RX62N Group

ADPCM Audio Compression Record and Playback

R01AN0698EU0102
Rev. 1.02
August 10, 2011

Introduction

This document explains what Adaptive Differential Pulse Code Modulation is and how it is used for audio compression and decompression on an embedded RX62N based target platform. This is demonstrated with a project that includes source code.

Audio playback can easily be added to an embedded application without expensive hardware due to the following reasons:

- A D/A converter is not necessary since a PWM signal can be filtered for playback instead.
- Less memory is needed with 4:1 ADPCM encoding.
- Regular ‘parallel’ internal or external flash memory is not needed for audio data storage if serial flash memory is used.
- Audio re-programming for this external audio memory can be made simple with a low cost serial debug interface and a tool like Hew Target Server to automate audio data programming.

Target Device

RX62N.

The YRDKRX62N board is used in this application, which has an RX62N MCU, the R5F562N8BDFP. (100 pin, 96kB RAM, 512+32 kB flash memory).

Related Documents

- REJ06J0085-0100, Rev.1.00, Jun.15.2010. “M3S-S2-Tiny” for RX Family. Sound Data Compression/Expansion Software. Explains the usage of the audio data compression/expansion software library along with a simple sample program.
- When the YRDK CD has been installed, the Manual Navigator (located under Start⇨All Programs⇨Renesas High Perf. Embedded Workshop) will contain multiple documents including Quick Start Guide, RX hardware manual, board schematic etc.

Contents

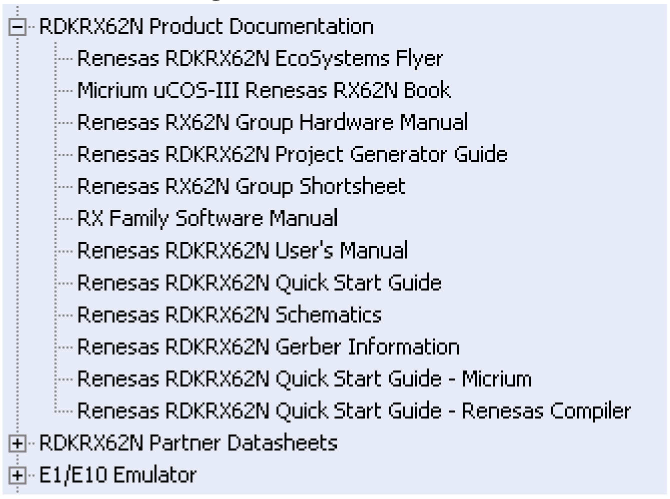
1. Overview

The purpose of this application note is to provide an example of how to perform ADPCM encoding and decoding on the YRDKRX62N board. A sample workspace with source code and libraries are provided and commented on throughout this document.

2. Tools and Documents

Please install the YRDK62N CD. This will give you:

- HEW with its source code Project Generator, compiler
- The Segger J-Link Debugger for the board, etc.
- Documentation for the board. This includes schematics, HW manual for MCU, User Manual, and third party device information. See Figure 1.



- RDKRX62N Product Documentation
  - Renesas RDKRX62N EcoSystems Flyer
  - Micrium uCOS-III Renesas RX62N Book
  - Renesas RX62N Group Hardware Manual
  - Renesas RDKRX62N Project Generator Guide
  - Renesas RX62N Group Shortsheet
  - RX Family Software Manual
  - Renesas RDKRX62N User's Manual
  - Renesas RDKRX62N Quick Start Guide
  - Renesas RDKRX62N Schematics
  - Renesas RDKRX62N Gerber Information
  - Renesas RDKRX62N Quick Start Guide - Micrium
  - Renesas RDKRX62N Quick Start Guide - Renesas Compiler
- RDKRX62N Partner Datasheets
- E1/E10 Emulator

Figure 1 *After installing the YRDK62N CD you will get documentation covering the board, the MCU and for third party peripherals used on the board.*

2.1 File Directory Content

See the header of source code file `ADPCM_Demo\sample_player\main_ADPCM.c` for directions on running the demo.

Also included with the source tree is a separate directory containing a project with which ADPCM encode and decode libraries can be built. These libraries are used by the playback/record sample code.

Here is how the code/libraries are laid out in the RX_RDK_ADPCM_Demo directory.

```
RX_RDK_ADPCM_Demo              Workspace directory of HEW
|
|-- ADPCM_Demo                 Project directory of HEW
|       |
|       |-- ADPCM_audiodata    ADPCM sound data
|       |
|       |-- ADPCM_Lib          Workspace directory of HEW
|       | |
|       | |-- ADPCM            Project directory for full ADPCM lib
|       | |         |
|       | |         |-- lib    Library storage for ADPCM with encode and decode
|       | |         |
|       | |         |-- Release  Configuration directory
|       | |
|       | |-- ADPCM_Decode     Project directory for ADPCM decode only
|       | |         |
|       | |         |-- lib           Library storage for ADPCM decode only
|       | |         |
|       | |         |-- Release       Configuration directory
|       | |
|       | |-- ADPCM_Encode          Project directory for ADPCM encode only
|       | |         |
|       | |         |-- lib           Library storage for ADPCM encode only
|       | |         |
|       | |         |-- Release       Configuration library
|       | |
|       | |-- src                   Source code to the ADCPM libraries
|       |
|       |-- BSP                     Directory for board specific support routines
|       |
|       |-- Debug                   Configuration directory
|       |
|       |-- RPDL                    R-PDL interrupt handler source code and library
|       |
|       |-- sample_player           Sample code demonstrating ADPCM encode and decode
|
|-- ADPCM_TOOL                      ADPCM conversion tool
        |
        |-- bin                     Tool to convert .wav files to ADCPM format
        |
        |-- doc                     Documentation for using ADPCM conversion tool
```

2.1.1 Directory Content Explanation

- **ADPCM_audiodata**
  Contains ADPCM encoded sample sound clips record at an 11 kHz sample rate. The file adpcm_sample.dat is currently linked into the demo build and resides in flash and can be used as audio source to test the decode/playback logic.
- **ADPCM_Lib**
  Contains the workspace, source code and library files associated with the ADPCM encoding and decoding algorithms.
- **BSP**
  Contains the Board Support Package software for the YRDKRX62N board. It includes the hardware initialization routines, drivers to write to the LCD, manage the on-board switches and an API layer to coordinate RSPI bus access.
- **RPDL**
  [See also 3 下の] Contains the necessary header files to reference the library functions, sample interrupt handlers, and the RPDL .lib file. RPDL is utilized throughout the demo code to perform various peripheral set up and modification as required to perform the record and playback functions.
- **sample_player**
  Contains the demo application source code.

Two workspaces are provided in this directory tree. One workspace, ADPCM_Demo, provides a sample program that demonstrates how to use the ADPCM libraries to perform recordings and playback on the YRDKRX62N board. The other workspace, ADPCM_Lib, provides projects to build three variations of the ADPCM library:

1.	**ADPCM**	Builds the full library with both encode and decode capability
2.	**ADPCM_Decode**	Builds the library with only the decode function
3.	**ADPCM_Encode**	Builds the library with only the encode function

3. RPDL

The Renesas Peripheral Driver Library is included in the workspace. RPDL is used to set up a peripheral using just a function call with the appropriate arguments. For example, this is used to set up the MTU used for recording in the sample code.

The manual for RPDL should be under your windows Start menu. ADPCM_Demo Workspace.

4. Sampled and Compressed Audio

Almost all electronic audio reproduction today is digital. This means that to record and reproduce audio it is necessary to sample the original waveform, store the samples, and later use the samples to re-create the original signal as closely as practical.

If the sampling rate is too low, the higher frequencies will be lost. Losing non-audible frequencies above 20 kHz is usually not an issue. "Phone quality" audio is sampled at 8 kHz, meaning that the highest reproduced signal will be 4 kHz. This is usually considered minimum quality.

The other aspect of audio digitizing is the sample width. The more bits per sample are used, the higher the quality of the signal. In this application note we will be using 16-bit audio technology. *Note that only 8-bit data is used in the audio samples provided.*

4.1 Pulse Code Modulation & Compression

The reason to use compression is to save memory space and communication bandwidth. We will show how ADPCM works, but first describe PCM and DPCM.

4.1.1 Pulse Code Modulation, PCM

Pulse Code Modulation is very straightforward. Each sample is coded "in itself" with its numerical value, with no relation to, or dependence on the previous or following sample values. With PCM, if a signal at one point in time is 3V, with the maximum output of 5V in the system, it will have a 16-bit PCM value of $3/5 * 2^{15} = 39322$ (or $999A_H$). For 8-bit PCM sampling the value would be 0x99, since $3/5 * 2^8 = 153$ ($0x99_H$).

4.1.2 Differential Pulse Code Modulation, DPCM

If the last audio output value is saved, it is only necessary to save the difference in output from the last sample. With the simplest type of DPCM encoding, the next value of the audio signal is simply an offset of the current. Since only the difference between the prediction and the actual sample is needed, less data needs to be saved.

With a more sophisticated DPCM encoding, the next PCM value is predicted by both the encoder and the decoder. The sample then *offsets* this *prediction*. The better the prediction, the fewer the bits needed to represent the same information.

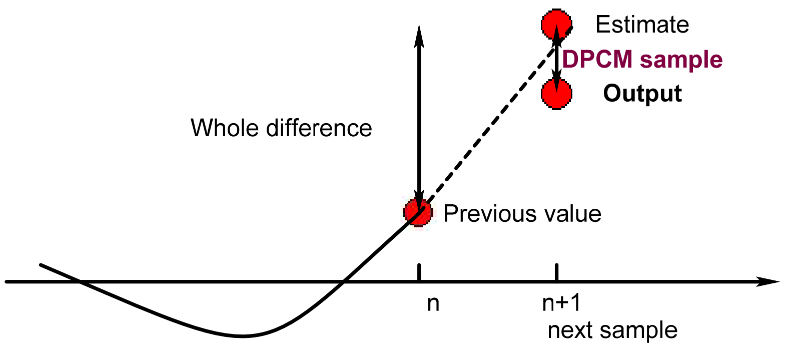


Figure 2 *DPCM variant where the next PCM value is predicted both by the encoder and the decoder. The sample is then the offset of the prediction.*

4.1.3 ADPCM

With ADPCM, a further reduction in data is achieved. The compression is 4:1. A 16-bit PCM sample is therefore encoded as a 4-bit value. An audio file of type PCM and of size 100 kB will therefore only take 25 kB of memory.

In ADPCM, the algorithm adapts the weight, or ‘stepsize’, of the unit step for each sample. For example, the quantization value of ‘1’ may translate to a change of 40 in the output PCM value for one sample, but for a later sample instead translate to a value of 72. This allows further reduction of data bandwidth compared to DPCM.

Each 4-bit sample encodes *a non-linear and varying difference* between each sample, encoded as

Bit #	3	2	1	0
	Sign	Magnitude (0 to 7)		

Each PCM sample is given by t*he Dialogic ADPCM Algorithm:*

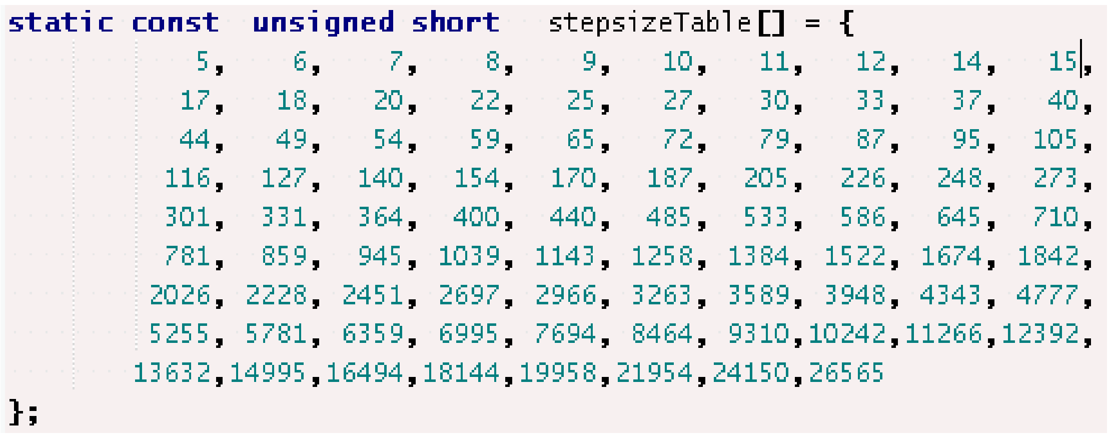
Output(n) = Output(n-1) +- stepsize x (b2 + b1 / 2 + b0 / 4 + 1/8)
Sign +- given by b3

The recursive formula yields the next PCM decoded output value based not only on the magnitude of the sample, but on the *weight* of the stepsize. This weight changes each sample depending on the *magnitude itself and two table lookups*. The magnitude can only be 0-7 units but the stepsize varies, or *adapts*. The end result in total PCM steps between samples can vary greatly.

The first table

ADPCM input mag-nitude	Table index change
7	8
6	6
5	4
4	2
3	-1
2	-1
1	-1
	-1

gives the number of steps to take within the stepsize table:



```
static const  unsigned short   stepsizeTable[] = {
         5,    6,    7,    8,    9,   10,   11,   12,   14,   15,
        17,   18,   20,   22,   25,   27,   30,   33,   37,   40,
        44,   49,   54,   59,   65,   72,   79,   87,   95,  105,
       116,  127,  140,  154,  170,  187,  205,  226,  248,  273,
       301,  331,  364,  400,  440,  485,  533,  586,  645,  710,
       781,  859,  945, 1039, 1143, 1258, 1384, 1522, 1674, 1842,
      2026, 2228, 2451, 2697, 2966, 3263, 3589, 3948, 4343, 4777,
      5255, 5781, 6359, 6995, 7694, 8464, 9310,10242,11266,12392,
     13632,14995,16494,18144,19958,21954,24150,26565
};
```

The use of both these tables is best illustrated with an example:

The stepsize at one sample point in time is 400. See the stepsize table. If the next sample has the magnitude 5, the index table’s row (marked yellow) says to increase the stepsize to a value four steps up in the stepsize table, so the stepsize becomes 586. Let's say the next sample's magnitude is 2, for which the table gives -1, meaning the stepsize should decrease to the value given one step down in the table from 586, and so stepsize for the next sample becomes 533. And so on.

5. Audio Memory Space

Audio can be stored on the MCU itself but could quickly fill up available memory space. Here is a calculation example for 16-bit PCM audio compressed 4:1 with ADPCM.

One byte of audio memory contains two samples of four bits each. With sampling rate 8 kHz (phone quality) we get

8000 samples/sec * ½ bytes/sample = 4000 bytes/sec

or 4 kB memory space per second @8 kHz.

With 1 MB of memory one could therefore theoretically store

1000 kB / 4 kB/s = 250 seconds

or over 4 minutes of audio per MB @8 kHz.

6. Board and MCU Considerations

6.1 Jumper settings

6.1.1 On-board speaker output

The YRDKRX62N has a small speaker provided for convenience that may be used to check audio playback. A two-pin jumper, JP7, is provided to allow the audio signal from the RIGHT audio channel to be routed to the speaker. JP7 is located in the lower-right area of the PCB with the label SPK ENA.

- To hear audio playback on the speaker, connect the two pins of JP7 with a jumper.
- To disable audio output from the speaker, remove the jumper from JP7.

Note 1: Rev. 5 boards and later have an additional jumper, JP17, that routes the AUD_R signal to the speaker's amplifier. Pins 1-2 of this jumper are shorted with a trace on the back of the PCB by default which is correct for this application. However, if JP17 has been modified for other use, be sure to connect pins 1-2.

Note 2: The small size and low volume output from the on-board speaker requires listening closely.

6.1.2 Stereo output jack

The YRDKRX62N has a dual-channel audio amplifier that is well suited to listening with the use of stereo headphones plugged into the provided stereo output jack. Alternatively, the pre-amplified audio signal may be routed to an external amplifier with a stereo audio cable for greater output. Sound output will be from the RIGHT audio channel in this case.

Rev. 5 boards and later have three jumpers, JP17 (AUD_R / SPK SEL), JP18 (AUD_R/AUDIO SELECT), and JP19 (AUD_L/ AUDIO SELECT). Each of these jumpers has three pins (1-3). Pins 1-2 of these jumpers are shorted with a trace on the back of the PCB by default which is correct for this application. However, if the traces have been cut for other use, be sure to connect pins 1-2 on JP17 through JP19 with jumper shorting plugs (3-pin headers may need to be installed).

6.2 Board rework needed for recording

If working on a YRDKRX62N board with a revision lower than Rev. 5, a minor physical rework must be done to be able to record to RAM. If you only plan to do audio playback from flash, this is not necessary.

The area of the board affected by the rework is in the lower right hand corner between the potentiometer and the speaker. There are two changes that must be made and are denoted in the schematic capture below:

(a) **Rework 1**

- Add 15K resistor on the empty pad at R69
- Add 4.71K resistor on the empty pad at R70

(b) **Rework 2**

- Add 100K pull-up resistor as shown to 3V3A to set ADC to midpoint

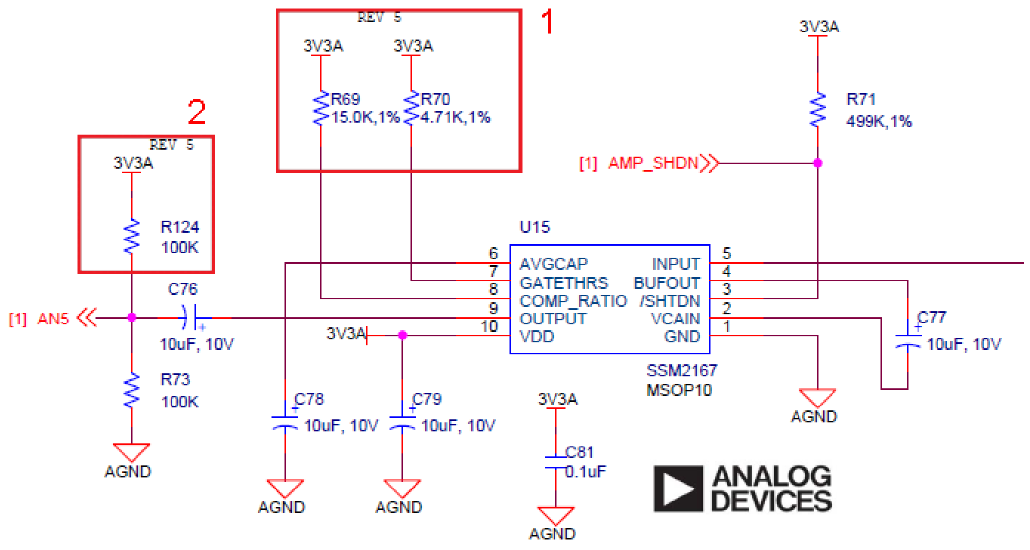


Figure 3 *Rework necessary to be able to use the recording circuit of the RDK are shown here. Rework 1 and 2 are marked with red rectangles.*

6.3 MCU Timers Used

The timers used are selected in file `syscfg.h` and are set to the following in the sample code:

```
/* MTU channel used for recording sample rate timing.     */
#define RECORD_SAMPLE_MTU       6

/* MTU channels used for carrier and sample rate timing.  */
#define SAMPLE_MTU              7
#define CARRIER_MTU             8
```

The Audio Right channel is connected to pin MTIOC8A. This pin carries the PWM signal of timer channel `CARRIER_MTU` running in PWM Mode 1. `TGRA` is used for the PWM period (the PWM carrier frequency) and the value set in TGRB specifies the duty cycle (sample value).

The Audio Left channel is connected to pin MTIOC8B, which if configured (not done in demo) can only run in PWM Mode 2.

An alternative software configuration of section 10 of the timers could save MCU resources by freeing up the DTC for other usage. In addition, this method would have the benefit that it could be used for both channels L+R (stereo) using the same MTU timer setup and PWM mode (Mode 2).

6.4 DAC

The DAC could also be used to playback audio instead of sung PWM. Comments on doing this are in section 10.

7. Demo Record & Playback

The ADPCM_Demo project in the RX_RDK_ADPCM_Demo workspace provides a sample recorder and playback program. Recordings utilize the Analog Devices ADMP401 omni-directional microphone to collect an analog signal and pass it to the 12-bit Analog/Digital Converter peripheral on the RX62N processor. Samples are collected at an 11.025 kHz rate, converted to PCM data, and then compressed to ADPCM.

Playback is a slightly more complicated process, requiring hardware to pace PWM adjustments. It starts by decoding a segment of the ADPCM data into an array used as input into the Data Transfer Controller (DTC) peripheral. The DTC is then started to transfer the PCM data to the PWM timer to modify the duty cycle. The DTC updates the PWM timer at the defined sample rate of 11.025 kHz. When the DTC has finished transferring the current segment of the PCM data, the processor is interrupted and another segment of decoded data is fed into the DTC. This process is repeated until the entire audio sample has been processed.

7.1 Demo State Machine

There is a simple state machine that is used to manage the playback and record processes in the provided sample code. There are 6 states:

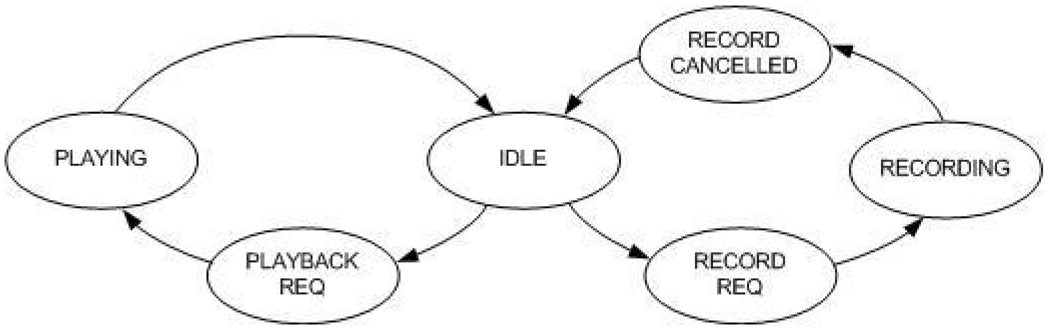


Figure 4 *ADPCM Demo State Machine*

SOUND_IDLE is the idle state when no buttons have been pressed and no ADPCM data is being played. It can only be entered from SOUND_PLAYING or SOUND_RECORDING_CANCELLED.

SOUND_PLAYBACK_REQUESTED is entered when switch SW1 is pressed on the board. It can only be entered from SOUND_IDLE.

SOUND_PLAYING is entered when the "player" is started and remains in this state for the duration of the playback. There is no way to "cancel" playback. It can only be entered from SOUND_PLAYBACK_REQUESTED.

SOUND_RECORD_REQUESTED is entered when switch SW2 is pressed. It can only be entered from SOUND_IDLE.

SOUND_RECORDING is entered when the "recorder" is started. The state machine remains in this state while SW2 remains pressed and the recording buffer has not been filled. It can only be entered from SOUND_RECORD_REQUESTED.

SOUND_RECORDING_CANCELLED is entered when the current recording is to be stopped. This can be because SW2 has been released or the recording buffer has been filled. It can only be entered from SOUND_RECORDING.

7.2 Playback Selection from RAM or Flash

The sample code provides the ability to play back either a pre-encoded file that has been linked into flash with the build, or play a dynamically created audio clip from RAM generated by the recording process done in the demo. A preprocessor '#if' statement in `main_ADPCM.c` (see Figure 5) determines whether a pre-linked audio clip from flash, which comes with the workspace, is used as audio source, or a an audio clip from RAM. *Note that the latter(RAM) is only available after the user has first made a recorded audio clip using switch 2.*

```
if( g_SrcFlag == PLAY_FLASH_FILE )
{
#if PLAY_FROM_RAM
    /* The following code will play back a recording saved in memory    */
    ADPCM_Playback( (uint8_t *)&g_EncodedData[0], fsize );
#else
     /* The following code will play the prerecorded clip built in flash */
    fsize = ADPCM_DATA_SIZE;
    ADPCM_Playback( (uint8_t *)ADPCM_ADDR, fsize );
#endif
}
```

Figure 5 *Depending on this preprocessor'#if' statement in main_ADPCM.c, either a pre-linked audio clip from flash which comes with the workspace is used as audio source, or a clip available after a recording to RAM has been done - with PLAY_FROM_RAM set to 0 nothing will be played (silence) after startup until something is recorded.*

7.2.1 Playback from RAM

After user has recorded using switch 2, the audio data will be available in RAM (in the `g_EncodedData[])`. For this to happen, PLAY_FROM_RAM must be set to 1 as described in 7.2. To play the "dynamically" created ADPCM file, make the recording using Switch 2. Then simply press SW1 to listen.

7.2.2 Playback from Flash

If you wish to play a linked-in file in flash, you must change the code in main() to use the ADPCM data located at ADPCM_ADDR instead of the data at g_EncodedData[].

The data in ADPCM_ADDR comes from the file `adpcm_sample.dat` located in the ADPCM_audiodata directory. For this demo, the length of the file has been hardcoded by the constant ADPCM_DATA_SIZE.

If you wish to use a different audio file in your build, place your audio data file in the ADPCM_audiodata directory, you can either rename your file to the name that the build is expecting or change the name that the build is to pick up. In either case, you must also adjust the constant ADPCM_DATA_SIZE to the size of your new file.

(1) Creating Your Own ADPCM File on your PC

If you wish to create your own ADPCM file, included in this package is a tool called ADPCM.exe which encodes a WAVE file into ADPCM format. Select the WAVE file you wish to convert. It must meet the following specifications:

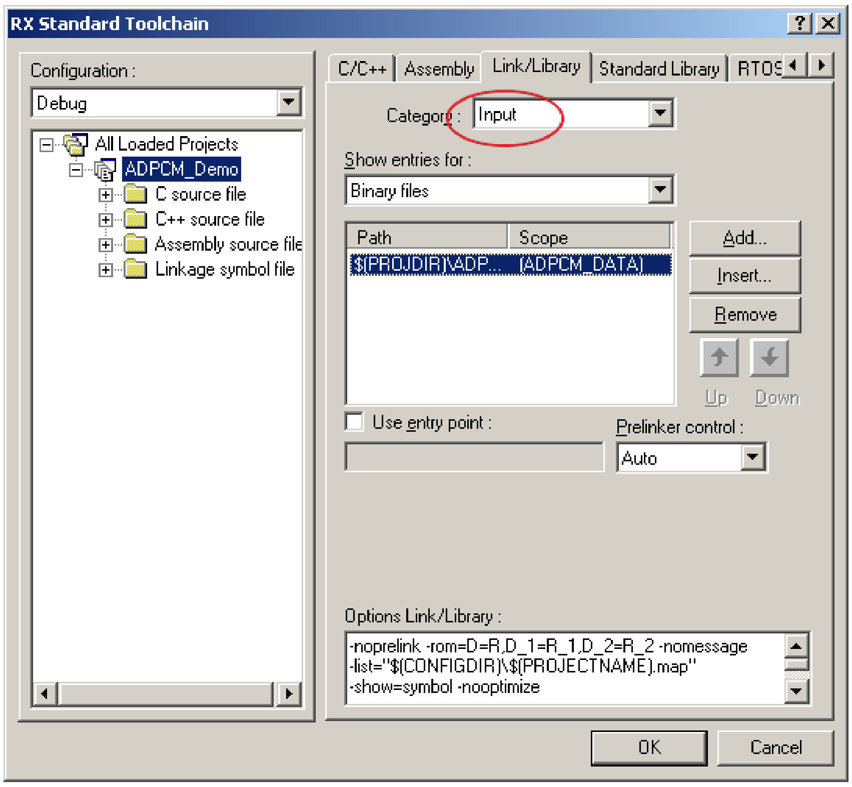
Quantization bits:	16
Channels:	Mono
Sampling Frequency	11.025 kHz
File Format	WAV

Double-click on the ADPCM.exe file in the ADPCM_TOOL/bin directory to start the tool. In the ADPCM_TOOL/doc directory is a PDF file that explains how to generate the file.

(2) Loading Prerecorded ADPCM to Flash

After you have encoded the file into ADCPM format, you need to change ADPCM audio files in the demo build by doing the following:

Select “Build” then “RX Standard Toolchain…”. In the window that appears, click on the “Link/Library” tab.



Configuration : Debug

Category : Input

Show entries for : Binary files

Path	Scope
$(PROJDIR)\ADP...	(ADPCM_DATA)

Use entry point :

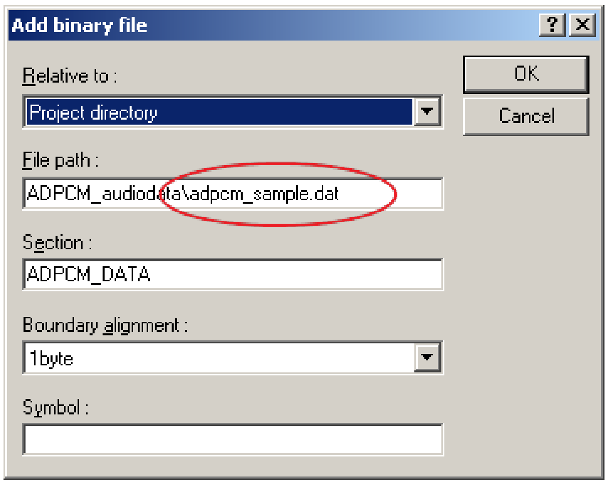
Prelinker control : Auto

Options Link/Library :

```
-noprelink -rom=D=R,D_1=R_1,D_2=R_2 -nomessage
-list="$(CONFIGDIR)\$(PROJECTNAME).map"
-show=symbol -nooptimize
```

In the “Category” pull down, select “Input” and then “Binary files” in the “Show entries for” pull down.

To modify the current entry, double-click on it (shown in blue in the above picture).



Relative to : Project directory

File path : ADPCM_audiodata\adpcm_sample.dat

Section : ADPCM_DATA

Boundary alignment : 1byte

Symbol :

Now, edit the file path for the file you wish to use and hit “OK”.

After saving your change, be sure you have updated the constant ADPCM_DATA_SIZE and then rebuild the project.

7.2.3 Decoding Process Detail (MTU and DTC)

When the pressing of switch SW1 is detected, function `SwitchPressCallback()`conditionally changes the state machine to SOUND_PLAYBACK_REQUESTED. `Main()` acknowledges the request, moves the state machine to SOUND_PLAYING and calls `ADPCM_Playback()` to play the ADPCM file.

`ADPCM_Playback()` decodes a portion of the data file and builds an array of PCM data to be fed into the PWM generator. This is done by utilizing the highly flexible RX62N **DTC** peripheral. It is set up to transfer data from the fixed length PCM data array directly to **MTU 8**, **CARRIER_MTU**, which is used to generate the PWM signal.

The pacing of the DTC is controlled by **MTU 7, SAMPLE_MTU**, set up to generate an interrupt at the sample rate (11.025 kHz). On every MTU 7 interrupt, the DTC will grab the next PCM data and update the register in the MTU which is used to control the duty cycle (TGRB). All of this occurs in the background, freeing up the MCU to allow the application to prepare the next portion of the file to be fed into the DTC.

When the DTC has transferred the last entry of the current array, the next MTU 7 interrupt is detected by the MCU and the next portion of already decoded data is fed into the DTC. The process continues until the entire file has been played.

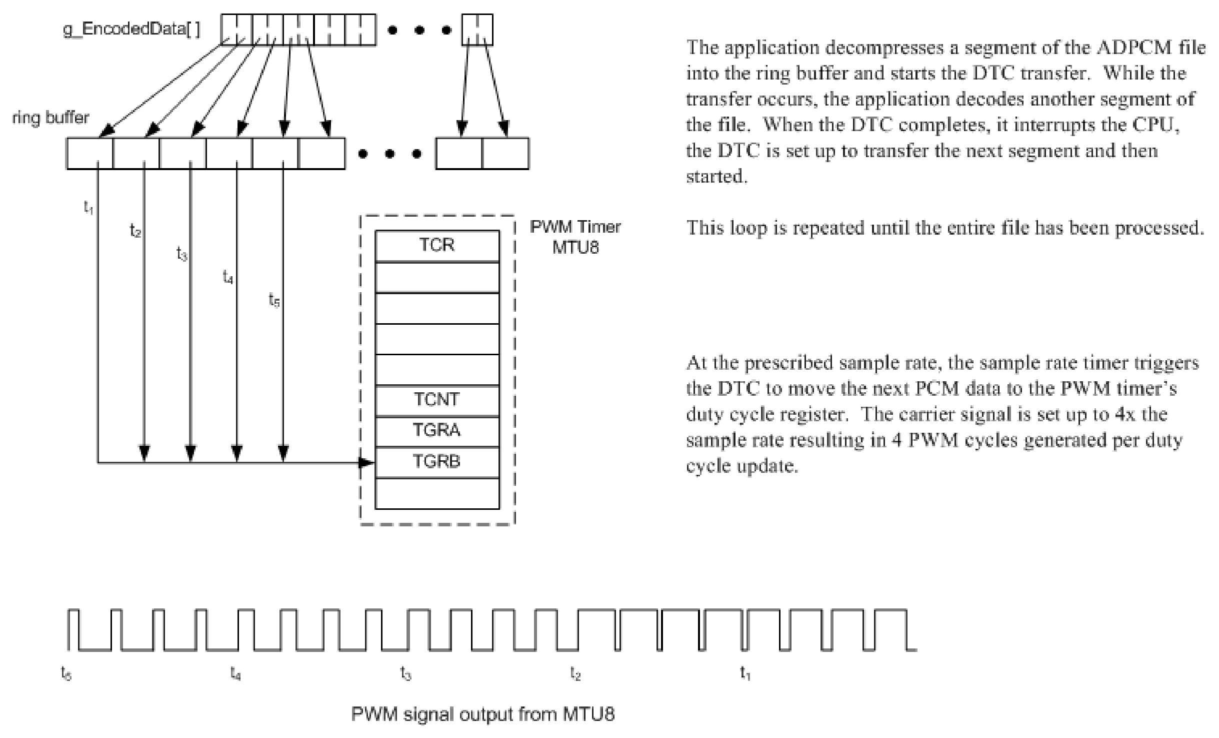

g_EncodedData[ ]
ring buffer
t1
t2
t3
t4
t5
PWM Timer MTU8
TCR
TCNT
TGRA
TGRB
The application decompresses a segment of the ADPCM file into the ring buffer and starts the DTC transfer. While the transfer occurs, the application decodes another segment of the file. When the DTC completes, it interrupts the CPU, the DTC is set up to transfer the next segment and then started.
This loop is repeated until the entire file has been processed.
At the prescribed sample rate, the sample rate timer triggers the DTC to move the next PCM data to the PWM timer's duty cycle register. The carrier signal is set up to 4x the sample rate resulting in 4 PWM cycles generated per duty cycle update.
t5
t4
t3
t2
t1
PWM signal output from MTU8

Figure 6 *Playback Process Detail.*

7.3 Recording with the board to RAM

Open the HEW workspace, build the project with PLAY_FROM_RAM set to 1 as described in 7.2. Load the image and hit Reset/Go. The LCD should display the project name and simple instructions to use switch 2 to record. When ready to make a recording, press and hold SW2 and then speak into the microphone located next to the speaker. Release the button when done recording. The recording will also stop when the internal memory space allocated for the recording is full.

LED14 will light while the recording is being made and is shut off when the recording process stops.

7.3.1 Encoding Process Detail

When a Switch 2 press is detected, `SwitchPressCallback()` will change the state machine to SOUND_RECORDING_REQUESTED. Function `main()` acknowledges the request, changing the state machine to SOUND_RECORDING and calls `ADPCM_Record()` to perform the actual recording.

The same MTU (MTU 8) is used for running the PWM recording as is used playback and is set up using RPDL (see 3 上の). It interrupts the processor at the a rate of 11.025 kHz, or every 91 us. The. The MTU interrupt is created by RPDL and will trigger the callback routine provided by the user to the RPDL API, in our case `RecordSampleRateTimerCallback()`. When this callback gets control, it reads the current value from the 12-bit ADC peripheral and save it in a local array until four values have accumulated. These are then converted to four 16-bit PCM values and saved in the global array `g_InputData[]`. `ADPCM_Record()` is then notified that there is data ready for compression.

When `ADPCM_Record()` detects the notification, it calls `EncodeData()` to perform the ADPCM conversion. This is then done by a function call to the ADPCM library.

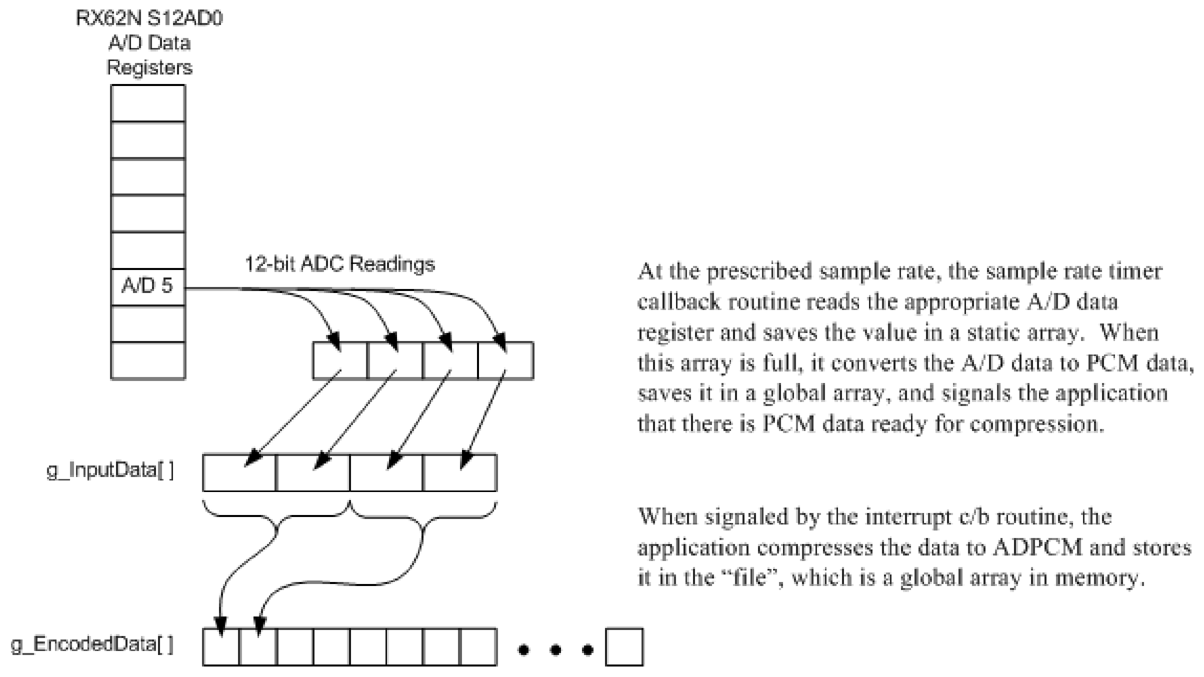

RX62N S12AD0
A/D Data
Registers
A/D 5
12-bit ADC Readings
g_InputData[ ]
g_EncodedData[ ]
At the prescribed sample rate, the sample rate timer callback routine reads the appropriate A/D data register and saves the value in a static array. When this array is full, it converts the A/D data to PCM data, saves it in a global array, and signals the application that there is PCM data ready for compression.
When signaled by the interrupt c/b routine, the application compresses the data to ADPCM and stores it in the "file", which is a global array in memory.

Figure 7 *The data array g_EncodedData[…]*

The ADPCM encoded data is written to the data array `g_EncodedData[]` which is defined as a 27,563 (6BABh) byte array. This provides 2*27,563=55,126 ADCPM samples. At 11.025 kHz, this allows for a recording of 5.00 seconds. Longer recordings can be made by modifying the constant MAX_DATA_LENGTH.

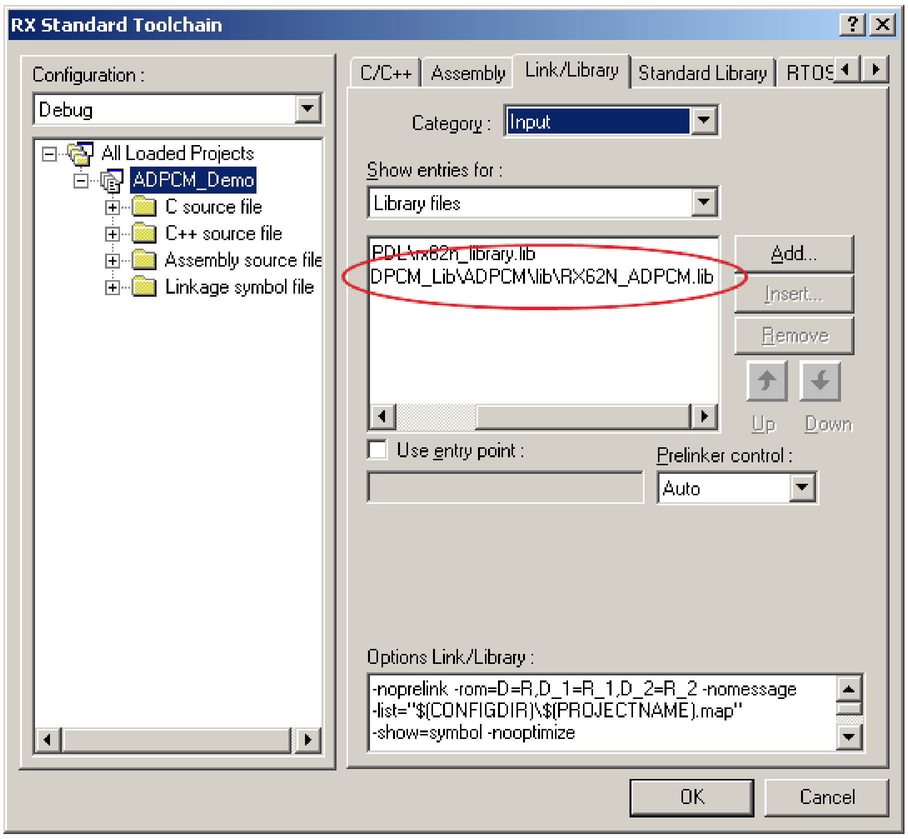
7.4 Selecting an ADPCM Encode/Decode Library

As mentioned earlier, there are three versions of the ADPCM libraries

- Decode only
- Encode only
- Encode and decode.

One version of the library must be linked into the demo build. By default, the full library (encode and decode) is linked in.

To change the library, select "Build" then "RX Standard Toolchain…". In the window that appears, click on "Link/Library".



Category : Input

Show entries for : Library files

```
PDL\rx62n_library.lib
DPCM_Lib\ADPCM\lib\RX62N_ADPCM.lib
```

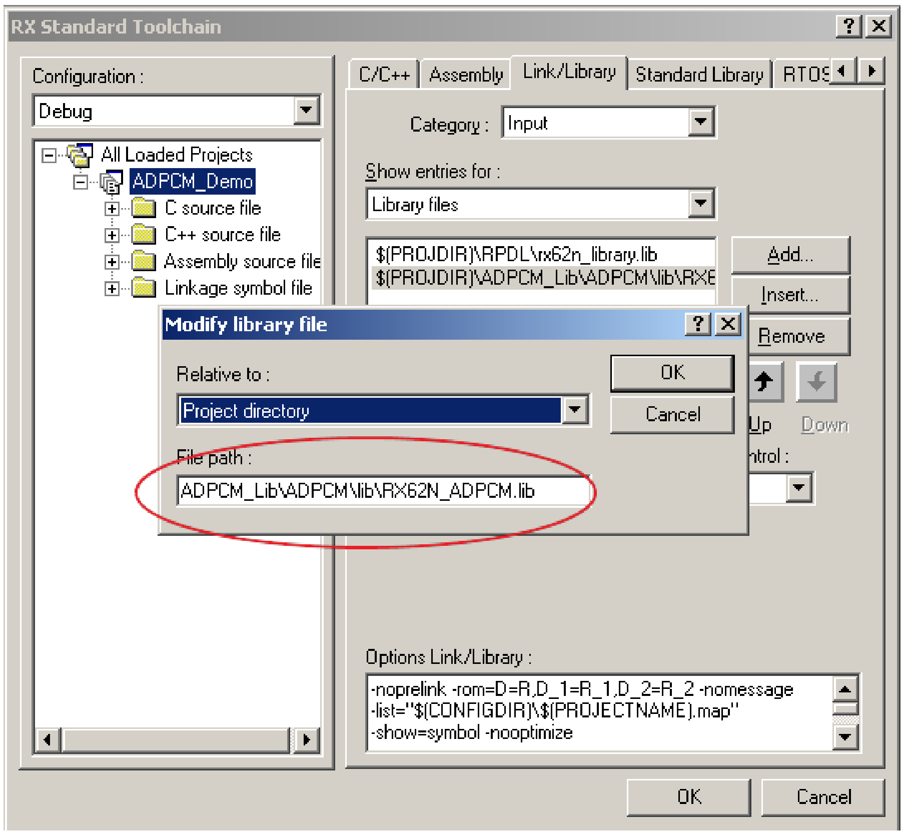
Use entry point :

Prelinker control : Auto

Options Link/Library :

```
-noprelink -rom=D=R,D_1=R_1,D_2=R_2 -nomessage
-list="$(CONFIGDIR)\$(PROJECTNAME).map"
-show=symbol -nooptimize
```

Figure 8 *In the "Category" pull down, select "Input" and then "Library files" in the "Show entries for" pull down. Double click on the entry circled in red above.*



Category : Input

Show entries for : Library files

```
$(PROJDIR)\RPDL\rx62n_library.lib
$(PROJDIR)\ADPCM_Lib\ADPCM\lib\RXE
```

Relative to : Project directory

File path : ADPCM_Lib\ADPCM\lib\RX62N_ADPCM.lib

Options Link/Library :

```
-noprelink -rom=D=R,D_1=R_1,D_2=R_2 -nomessage
-list="$(CONFIGDIR)\$(PROJECTNAME).map"
-show=symbol -nooptimize
```

Figure 9 *In the "Modify library file" pop up, modify the file path and name to the library file to be included in the demo build.*

8. The “ADPCM_Lib” Workspace

Precompiled libraries

Also included with the demo code is a workspace to build your own ADPCM library. As mentioned earlier, three projects are provided to custom build the library for your requirements. However, these libraries have already been built and are included in the distribution.

These libraries are named and located at:

- RX_RDK_ADPCM_Demo/ADPCM_Lib/ADCPM/lib/RX62N_ADCPM.lib
- RX_RDK_ADPCM_Demo/ADPCM_Lib/ADCPM_Decode/lib/RX62N_ADCPM_Decode.lib
- RX_RDK_ADPCM_Demo/ADPCM_Lib/ADCPM_Encode/lib/RX62N_ADCPM_Encode.lib

The ADCPM_Demo project is configured to link in RX62N_ADPCM.lib directly from the directory it was built in.

ADPCM Source Code

The source code to build the various ADPCM libraries is located ADPCM_Lib/src.

The key functions are written in assembler and reside in the following files:

adpcm_encoder.src	Compression routines
adpcm_decoder.src	Decompression routines
adpcm_table.src	Step and index tables used for both compression and decompression

Also included is a file called version.c, which reflects the current version of the encoding algorithm.

8.1 Encoding Library Functions

8.1.1 Initialization (R_apdcm_initEnc)

Prototype

```
void R_adpcm_initEnc(adpcm_env *wenv);
```

Explanation

This function initializes the sound data that is to be compressed.

This function has to be executed only once for completing the initialization before compressing consecutive ADPCM data.

The adpcm_env structure data type is declared in the header file r_s2_lib.h. Before executing this function, declare the adpcm_env type structure variable that is to be passed as an argument. Value need not be set to the argument before this function is executed.

Return value

None

8.1.2 Compression (R_adpcm_encode)

Prototype

```
int16_t R_adpcm_encode(int16_t smpln, adpcm_env *wenv);
```

Explanation

This function encodes (compresses) 16-bit PCM data to 4-bit ADPCM data. Encoded data is stored in the memory area initialized in the function R_adpcm_initEnc. The argument smpln specifies the number of bits into which ADPCM data is to be encoded. Value of smpln must be in multiples of 4. The size of the output memory location depends upon the smpln variable.

Return value

int32_t - If value of smpln is not in multiples of 4, -1 is returned, else 0 is returned

8.1.3 Refresh (R_adpcm_refreshEnc)

Prototype

```
void R_adpcm_refreshEnc(int16_t *inputAddr, uint8_t *outputAddr,
                        adpcm_env *wenv);
```

Explanation

This function refreshes internal PCM input data buffer, and refreshes the output ADPCM data buffer. The argument inputAddr specifies the memory area where input data is stored. The argument outputAddr specifies the memory area where ADPCM data is stored. When this function and function R_adpcm_encode are called in succession, R_adpcm_encode encodes from the top of the memory area pointed out by inputAddr, stores the encoded data from the top of the memory area pointed out by outputAddr.

Table 1: Arguments for the initialization function

Argument	Type	Explanation
inputAddr	int16_t *	Top address of PCM data area.
outputAddr	uint8_t *	Top address of ADPCM data (compressed data) area.
wenv	adpcm_env *	Pointer to the encode management structure.

8.2 Decoding Library Functions

8.2.1 Initialization (R_adpcm_initDec)

Prototype:

void R_adpcm_initDec(adpcm_env *wenv);

Explanation

This function initializes the sound data to be expanded.

This function has to be executed only once before expanding consecutive ADPCM data.

The adpcm_env structure data type is declared in the header file r_s2_lib.h. Before executing this function, declare the adpcm_env type structure variable that is to be passed as an argument. Value need not be set to the argument before this function is executed.

Return value

None

8.2.2 Decompression (R_adpcm_decode)

Prototype

```
int16_t R_adpcm_decode( int16_t smpln, adpcm_env *wenv );
```

Explanation

This function converts the encoded 4bit ADPCM data into decoded 16bit PCM data. Decoded data is stored in the memory area initialized in the function R_adpcm_refreshDec. The number of 16 bit samples decoded by each call to this function is determined by smpln and must be a multiple of 2.

Define the output sample buffer before calling this function. The size of this buffer depends upon the smpln variable. Required output memory will be size × 16 bit.

E.g. If size = 16, define int16_t buffer[16] to allocate storage for output sample buffer.

Return value

If the conversion size of the ADPCM data is an odd number, -1 is returned, else 0 is returned.

8.2.3 Refresh (R_adpcm_refreshDec)

Prototype

```
void R_adpcm_refreshDec(uint8_t *inputAddr, int16_t *outputAddr,
adpcm_env *wenv);
```

Explanation

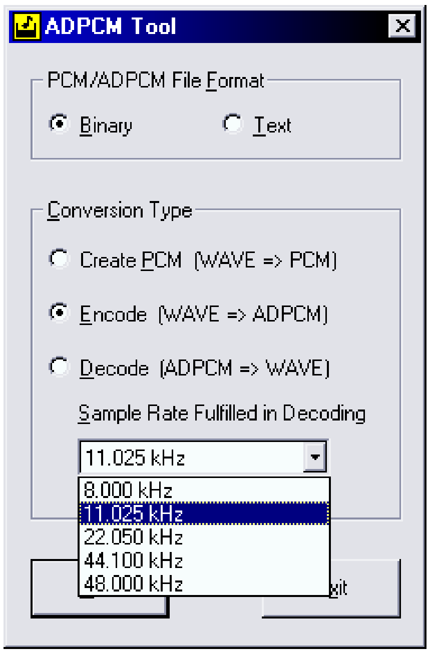
This function refreshes internal ADPCM input data buffer and refreshes the output PCM data buffer. The argument inputAddr specifies the memory area where input data is stored. The argument outputAddr specifies the memory area where PCM data is stored. When this function and function R_adpcm_decode are called in succession, R_adpcm_decode decodes from the top of the memory area pointed out by inputAddr, and stores the decoded data from the top of the memory area pointed out by outputAddr.

Return value

None

9. Single File Audio Format Conversion GUI

There is a windows GUI tool ADPCM.exe in the folder ..\R8CSPB_AN\tools\ADPCM-tool that also comes with this application note download. It can be used to create ADPCM encoded files from WAV-files one at a time. It can also be used to decode ADPCM files to WAV files. See the manual for ADPCM-Tool.


ADPCM Tool

PCM/ADPCM File Format: [✓ Binary] Text

Conversion Type: Create PCM (WAVE => PCM) [✓ Encode (WAVE => ADPCM)] Decode (ADPCM => WAVE)

Sample Rate Fulfilled in Decoding: 11.025 kHz

Figure 10. *The ADPCM Tool is a GUI for converting back and forth between ADPCM and WAV formats. After selecting what format types to convert between and sampling frequency, press 'Go' to select the file to convert.*

The ADPCM-tool GUI will only convert one audio clip at a time.

There are plenty of other audio conversion tools available on-line, however there are different ADPCM formats, not always compatible with each other.

10. Appendix A MTU Interrupt ISR instead of DTC - PWM Mode 2 - Stereo

An alternative software configuration of the timers with the YRDK62N board Rev.5 and higher could save MCU resources by freeing up the DTC for other usage.

The MTU8 compare match interrupt request signal TGIA8A is used in the default code to trigger the DTC. The DTC then writes the audio samples to the PWM output timer register (MTU8.TGRB) at the sampling rate. Instead of triggering the DTC, the above TGIA8A signal's interrupt service routine could be used instead. In addition to freeing up the DTC, this method has the benefit that it could be used for both channels L+R (stereo) using the same MTU timer setup and PWM mode (Mode 2). This means simpler code if both channels are used.

It is strongly recommended to use the timer registers in buffered mode so that the new value written to the PWM output timer register at the beginning of each PWM count period, and not a random time within the PWM period, which could result in audible disturbances.

Here is how the timers would be set up to do this, using MTU6 for the PWM duty cycles and another timer "MTUx" for the PWM period:

- Set TMDR register for buffered mode so TGRC is buffer for TGRA, and TGRD is buffer for TGRB.
- Set the TMDR register for PWM Mode 2.
- Set TBTM for buffered mode and compare match.

PWM Duty cycle:

- Duty cycles for L+R channels set in the buffer registers MTU6.TGRC and MTU6.TGRD.

PWM Period:

- TSYR set for synchronous operation. This would make for synchronous operation of TCNT in MTU6 with another MTU; "MTUx".
- MTUx set to clear TCNT with one of its TGRs, for example MTUx.TGRA.

To do this, changes to the jumper settings and must be made. This is only possible in Rev. 5 of YRDK62N and higher. Jumpers JP18 and JP19 must be set to use the MTU6 output pins MTIOC6A and MTIOC6B instead of the MTU8 pins (MTIOC8A and MTIOC8B).

11. Appendix B
Using DAC instead of PWM

Instead of using a PWM signal generated by timers, the on-chip DAC could be used instead.

If working on a YRDKRX62N board with a revision higher than Rev. 5, there is a jumper JP17 place reserved on the board which if used would connect the R audio channel to the DA1 pin, instead of a PWM source. To route the DAC audio out to the speaker on the YRDKRX62N board, it is necessary to cut the trace that shunts JP17 pins 1 and 2. The DA audio comes out on JP17 pin 3. Connect this pin instead to pin2 to route the DAC audio to the speaker. A 3-position jumper header may be installed at JP17 so that the audio can easily be switched between PCM audio or DAC audio.

The DAC uses the same data that was extracted from the data stream for the PCM R channel, so the pointer to the audio data buffer for the DAC should be the same as for loading the PWM output value (perhaps scaled). This data is normally 16-bit format, and since no scaling to 10-bit values has been done for the DAC, there is the possibility that the data may go out of the DAC range if the encoded signal amplitude level is too great. But as long as the data values remain below 10-bits then the audio will reproduce correctly.

Initializing the DTC will be the same except that the destination address (DAR) will be the DAC data register.

```
/* Define location where DTC is to write DUTY cycle values. */
#define DTC_DESTINATION (uint16_t *)&DA.DADR1
```

Below is an example of DA initialization.

```
/***************************************************************************
Function Name   : R_InitDA
* Declaration   : void R_InitDA(void)
* Description   : This function initializes DAC1
* Arguments     : none
* Return value : none
***************************************************************************/
static void R_InitDA( void )
{
  PORT0.DDR.BIT.B5 = 0;
  PORT0.ICR.BIT.B5 = 0;

  /* Cancel D/A Converter Module Stop state. */
  SYSTEM.MSTPCRA.BIT.MSTPA19 = 0;

  /* Init D/A Data Register. */
  DA.DADR1 = 0x0020;

  /* D/A Control Register. */
  DA.DACR.BYTE = 0x9F;   /* b7 DAOE1 - D/A Output Enable 1 - enabled.
                            b6 DAOE0 - D/A Output Enable 0 - disabled.
                            b5 DAE - D/A Enable 0 - independent on each channel
                            b4:b0 Reserved - Set to 1.*/

  /* D/A Data Placement Register. */
  DA.DADPR.BYTE = 0x00;  /* b7 DPSEL - Data Placement Select aligned to LSB.
                            b6:b0 Reserved - Set to 0.*/
}
```

Website and Support

Renesas Electronics Website
http://www.renesas.com/

Inquiries
http://www.renesas.com/inquiry

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Jun.7.11	—	First edition issued
1.01	July 25, ‘11		Changed 6.1, 6.3, 7.2.1, 7.3.
1.02	August 10, ‘11		Changed 6.1, 6.3

General Precautions in the Handling of MPU/MCU Products

The following usage notes are applicable to all MPU/MCU products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Handling of Unused Pins

   Handle unused pins in accord with the directions given under Handling of Unused Pins in the manual.

   — The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on

   The state of the product is undefined at the moment when power is supplied.

   — The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.
   In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.
   In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

   — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals

   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.

   — When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.

5. Differences between Products

   Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.

   — The characteristics of an MPU or MCU in the same group but having a different part number may differ in terms of the internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-65030, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
1 harbourFront Avenue, #06-10, keppel Bay Tower, Singapore 098632
Tel: +65-6213-0200, Fax: +65-6278-8001

Renesas Electronics Malaysia Sdn.Bhd.
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics Korea Co., Ltd.
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141