To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

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

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

RENESAS

R8C Family

ADPCM Audio Solution for Subatomic Particle Board

Introduction

This application note presents a low cost embedded audio solution, comments on its features and on possible variations and limitations. It is also intended as a guide for how to reprogram the SPB with new audio.

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.

In this application note, together with the material in the accompanying 'Source' download (code and documentation), we will take a look at:

- What ADPCM is.
- An audio solution using ADPCM implemented for the Subatomic Particle Board (SPB).
- How to create ADPCM audio files.
- How to load audio files to external serial flash memory using a debug interface.

This application note and sample code is based on the ADPCM object code library from *REJ06j0047_s2tinyap "M3S-S2-Tiny: Sound Data Expansion Software for Tiny Microcontrollers"*. That application note documents the ADPCM decode functions of the library *s2_r8ctiny_e_v200.lib.*

Note: There is a 'Source' download package available adjacent to this application note containing an audio playback application, tools, and further documents. You are expected to download and be familiar with that content as you proceed through this application note.

Target Device

R8C, M16C. The document is written for the YR8CSPB, which has the R8C/25 MCU. Most of the content is also relevant for the M16C (YM16CSPB).

Contents

1. Related documents 3
2. Sampled Audio 3
3. Pulse Code Modulation & Compression 3
4. The SPB Hardware 5
5. PWM 7
6. Low Pass Filter 9
7. Audio Output Amplifier 10
8. Creating ADPCM Audio 10
9. How Playback Works 13
10. Loading New Audio, HEW Target Server 14
11. The ADPCM Library 17
12. Decoding Priority and Design 17
13. Processing Requirements 18
14. Alternative Design Lower Interrupt Frequency and CPU load 19

1. Related documents

The following documents can be found in the 'Source' download package that can be downloaded for this application note. They are in the*..\R8CSPB_AN\workspace\docs folder*.

- YR8CSPB User Manual V1.1.pdf
- YR8CSPB_Schematic_revc.pdf
- YR8CSPB_BOM_REVC.xls
- REJ06j0047_s2tinyap.pdf "M3S-S2-Tiny: Sound Data Expansion Software for Tiny Microcontrollers". Only the object library description is of interest. That application note includes a different audio solution not relevant for this application note.

There is also plenty of information on-line about the SPB at
http://www.renesasrulz.com/community/forum/subatomic-particle-board

2. Sampled 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.*

3. 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.

3.1 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 $3/5 * 2^8 = 153$ ($0x99_H$).

3.2 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.

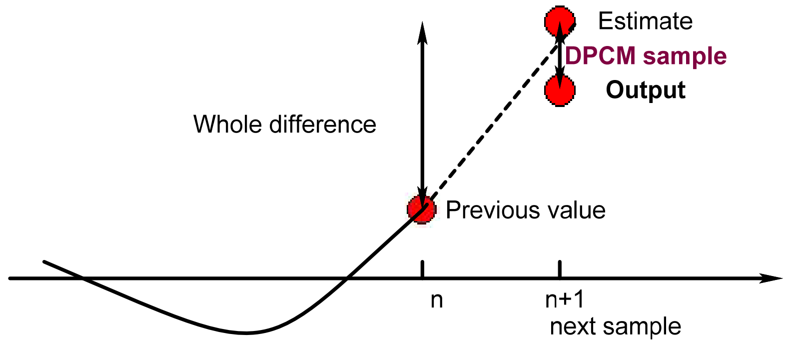

Figure 1. DPCM variant where the next PCM value is predicted both by the encoder and the decoder. The sample is then the offset of the prediction.

3.3 Adaptive Differential Pulse Code Modulation, 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. Audio playing time vs. memory space is addressed in 4.2 下の

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.

4. The SPB Hardware

The different blocks of the SPB are shown in Figure 2Figure 2 and Figure 3. Schematics, parts list, and User Manual for the SPB can be found in the 'Source' download.

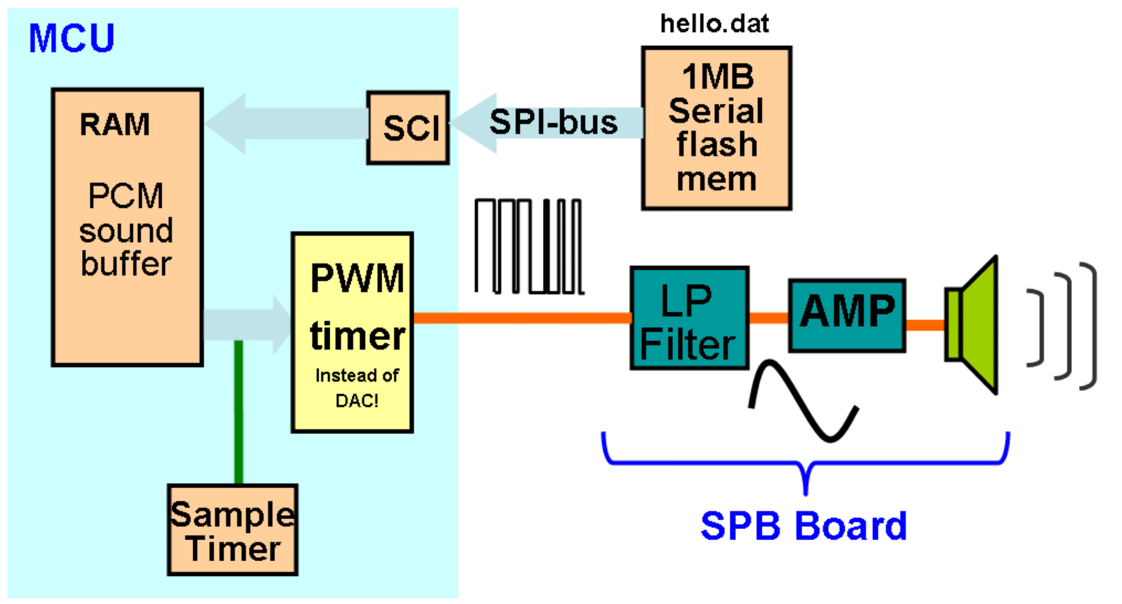

Figure 2. Functional block diagram of the YR8CSPB.

A Functional block diagram of the YR8CSPB can be seen in Figure 2, and a hardware block diagram in Figure 3. The blocks will be commented on individually.

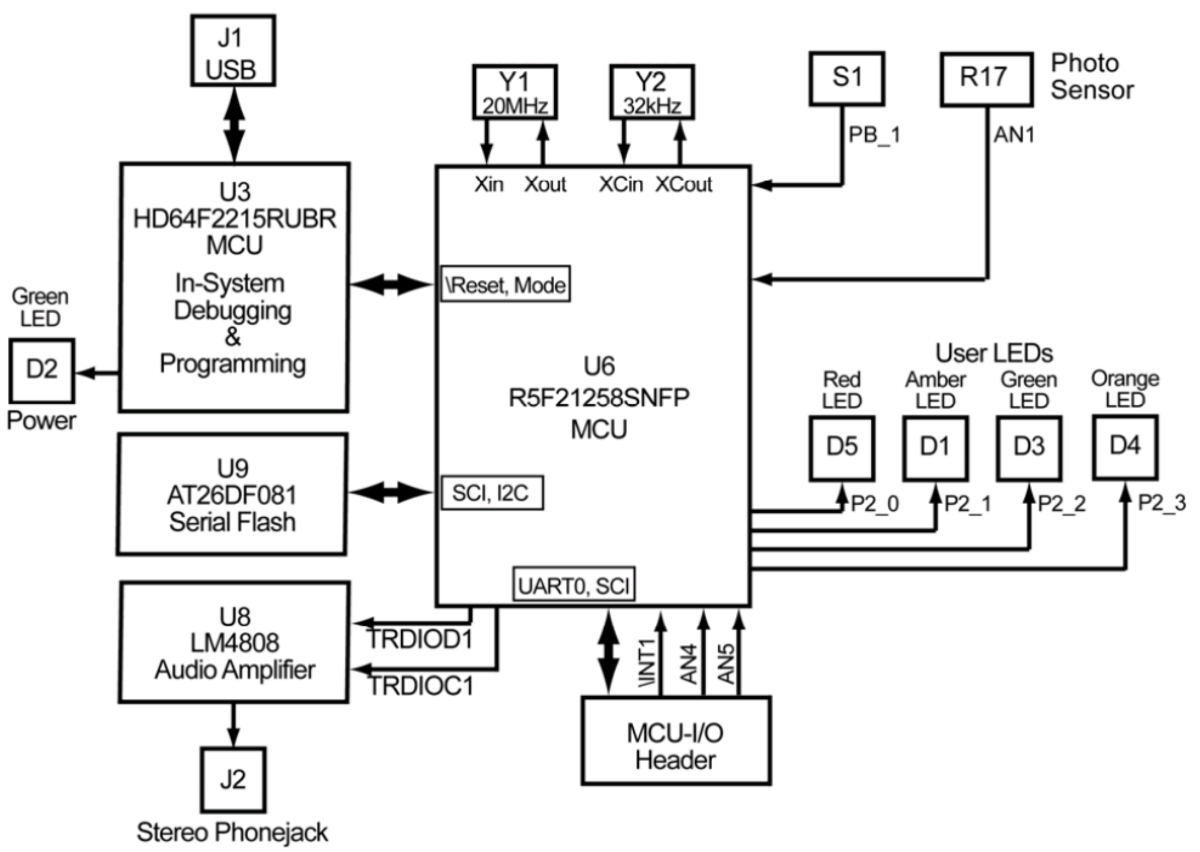

Figure 3. Hardware block diagram of the YR8CSPB.

4.1 System Clock

The sample playback software that comes with this application note uses the MCU's own Hi-speed On-Chip Oscillator by default and not the on-board 20 MHz crystal, since the human ear hardly needs the accuracy of a crystal. The crystal can easily be used as an alternative. This is easy to change as both are configured in the function *ConfigureOperatingFrequency*:

```
/* SELECT CPU CLOCK:
0 = XIN/XCIN clock. The main Xtal is 20 MHz.
1 = On-chip oscillator (OCO) clock. */
ocd2 = 1;
```

To use the 20 MHz crystal, assign ‘0’ to the bit ocd2. The crystal is connected to the Xin and Xout pins of the MCU.

4.2 External Audio Memory

Audio could 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.

The AT26DF081A is a 1 MB serial interface flash memory device connected to the MCU’s Clock Synchronous Serial I/O lines. The AT26DF081A is designed for use in a wide variety of high-volume consumer-based applications and is suited for data storage, eliminating the need for e.g. EEPROM devices.

The SSIO peripheral is used by the SPI protocol to read and write from the audio memory. The read and write functions are in the source file FlashSerial.c file, which calls the lowest level driver functions in file SPI_R8C25.c.

4.2.1 The Serial Flash interface

The Serial flash interface uses SPI communication using the Clock Synchronous Communication Interface. The MCU is set up for single master and the serial flash as single slave.

The pins on the MCU used for the SPI communication with the serial flash are:

SSCK: Clock I/O (p3_5)
SSI: Data In (p3_3)
SSO: Data I/O (p3_7)
SCS: Serial Chip-select (p3_4)

4.3 On-Board Debug Circuit

The SPB includes an on-board E8 debug circuit. As we shall see in 10 下の, this makes it simple for an end user to reprogram the on-board audio memory.

The E8 circuit enables communication between the PC and the R8C for debug and programming of the MCU and download of audio data to the serial flash. The E8 circuit connects to the PC Host as a USB Device, and to the R8C/25 MCU’s Mode and Reset pins. Note that if the Mode and Reset pins are connected to any other circuit, the debug/programming function is jeopardized.

More information on the debug circuitry is found in the YR8CSPB User Manual and Schematics. These are available from inside the Manual Browser, which is accessed from the PC Start Menu after installing the SPB CD.

5. PWM

To create an audio signal, a train of pulses with constant period but varying width is used. This is called Pulse Width Modulation. The audio modulates the pulse width from a minimum of 0% of the pulse period, to a maximum of 100%. The modulated signal is then filtered to reproduce the audio. This is shown in Figure 4.

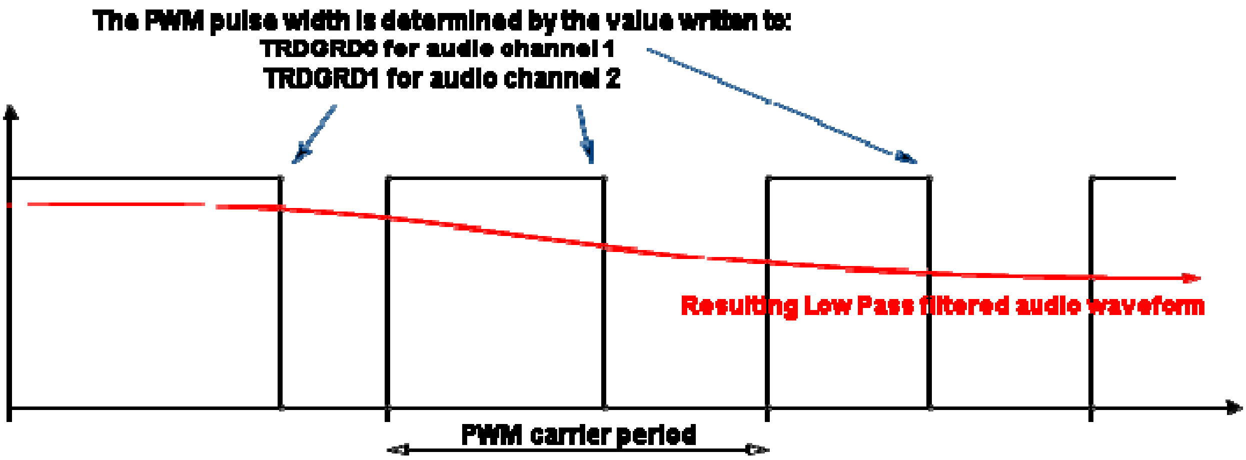

Figure 4. An audio signal is made by filtering a train of pulses with varying width. The width is determined by the value written to TRDGRD0 for audio channel 1, and TRDGRD1 for audio channel 2.

The PWM carrier frequency should be above 20 kHz so that it is not audible. The carrier must be filtered out by the Low Pass filter to leave only the audio signal. This is dealt with in section 6

5.1 Setting up the PWM Signal

For an alternative design that will use less CPU resources (but more TimerD resources), see 14 下の.

The PWM carrier frequency $f_{PWM\text{-}carrier}$ is determined by the setting of the PWM timer's period setting. In the source code, TimerD is used.

PWM period: 1/fk x (m+1)
Active level width: 1/fk x (m-n)
Inactive level width: 1/fk x (n+1)
fk: Frequency of count source
m: Value set in the TRDGRAi register
n: Value set in the TRDGRji register

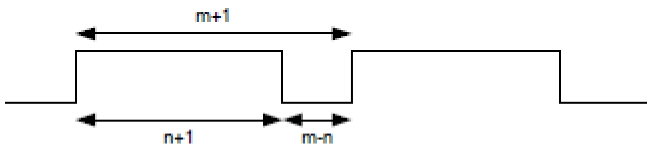

Figure 5. For an R8C's TimerD in PWM mode, the PWM period and duty cycle are set as in the figure. (Figure from the HW manual.) The PWM period is determined by TRDGRA1

The PWM period is set at initialization time in register TRDGRA1, and the duty cycle for the audio is written each sample period to TRDGRBi, Ci, or Di.

TRDGRC1 is used for the left channel in the solution, and TRDGRD1 is used for the right channel. This outputs audio to the MCU pins TRDIOC1 and TRDIOD1.

If the alternative method in section 14 is used to decrease the CPU load, the duty cycle will be written to TRDGRD0 for channel 1 and TRDGRD1 for a second audio channel. This will output the audio to ports TRDIOC1 and TRDIOD1. Note that TRDMR must be configured for buffered mode.

5.2 Setting Playback Sample Frequency

The timer reload value determines the PWM interrupt frequency; the 'audio' interrupt in the default solution. This is not the same as the sampling frequency since this solution does not use a PWM reload buffer. When not using a reload

(a) **Timer reload value**

For a different sampling frequency (the example clips have different sampling rates just to show how to switch playback frequency) we needed to solve the equation

APP = f1 / (PCF * fs) - 1

Where:

APP = PWM Period Setting, or timer reload value, and AUDIO_PWM_PERIOD in the source. This determines the PWM carrier period.
f1 = This is the system clock frequency. The timer is set to count f1.
fs = Sampling frequency.
PCF = This is an integer value given by
PWM carrier frequency / sampling frequency.
This must be a whole non-rounded integer, large enough so the carrier is not audible; above 20 kHz. The sample code's non-buffered solution requires the PWM interrupt be used in order to load a new sample right at the beginning of a PWM period. This is dealt with in detail in section 14.

In our case, the sample code is by default setup for **11** kHz:

f1 = **20 MHz**.
fs = **11,025 Hz**.
PCF = **3**. This will make the carrier frequency 3 * 11 025 Hz = 33 075 Hz, making the PWM Period Setting:

APP = (20 000 000 / 3 * 11025) – 1 = **605**.

(b) **Change the sample playback frequency:**

Open hwinit.c

Go to the InitTimer function.

Several of the new audio clips were sampled at a 16 kHz sampling rate. We need to change the playback frequency so these clips are replayed correctly. Using the formula to change to a 16 kHz sample rate to play the last six clips correctly we get
APP = (20 000 000 / 2 * 16000) – 1 = **624**.

Recompile and play the audio clips again and hear the clips at the correct rate!

6. Low Pass Filter

The outgoing PWM signal from the MCU's output pins must be filtered to rid the signal of the high frequency PWM carrier. This PWM carrier frequency must be above 20 kHz so that it is not audible to humans. As animals can hear above 20 kHz, a margin above 20 kHz is recommended. The carrier frequencies in the source code for the sampling frequencies 8, 11, and 16 kHz are all around 32 kHz. The carrier frequency is determined by the factor PCF, see 5.2.

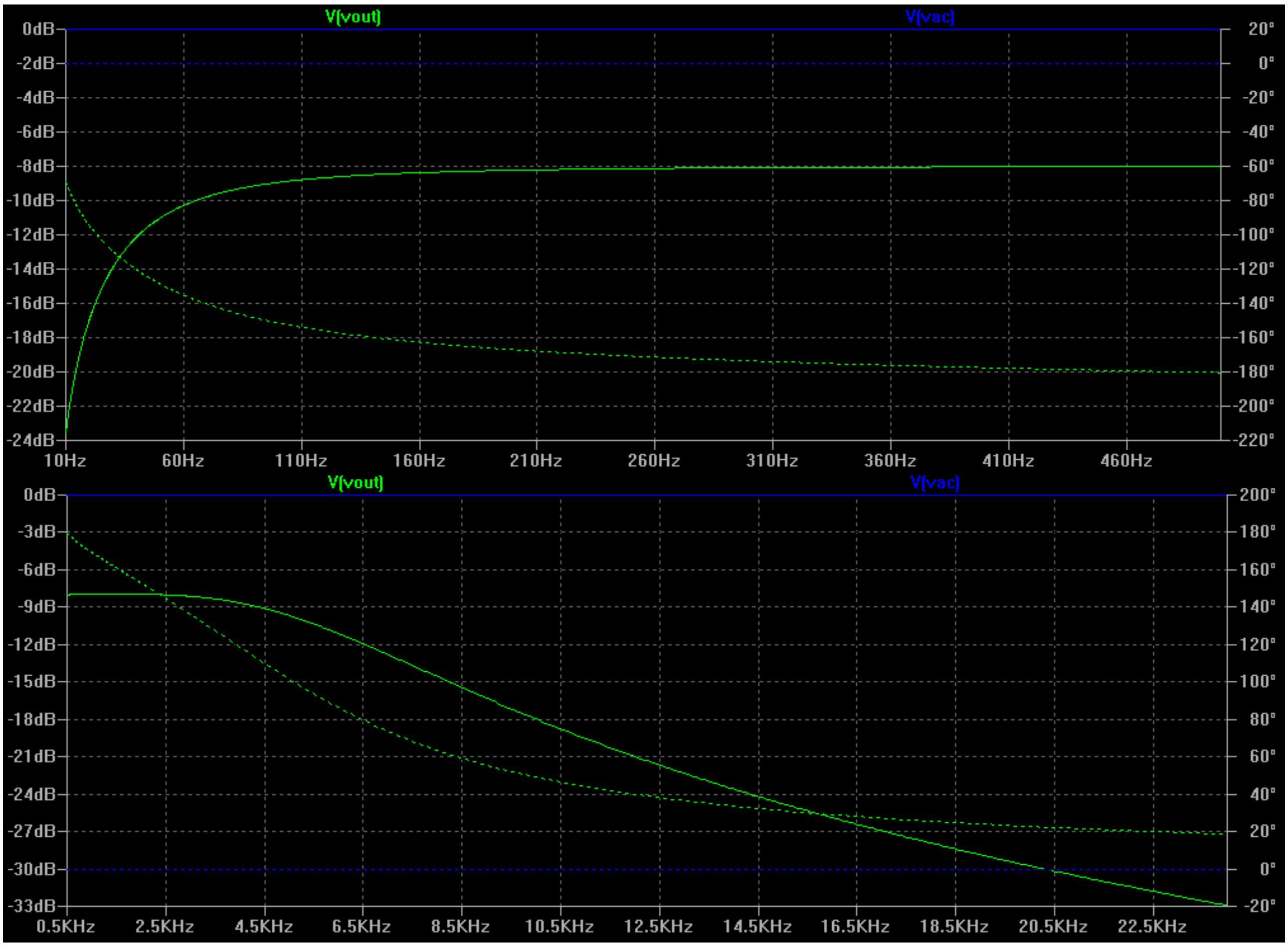

Figure 6. The audio output low pass filter on the SPB simulated in Linear Technology's LTSpice. The solid line shows frequency response and the dotted line phase shift for the output voltage relative to the input voltage. The top graph shows the frequency response for 10–500 Hz, and the lower graph for 500 Hz-24 kHz.

The highest sampling frequency is 16 kHz in the provided source code. According to the sampling theorem, the highest frequency that can be sampled is 8 kHz. The filter starts to slope right before 8 kHz and has a cut-off frequency just above 5 kHz.

The LTspice model for the LP-filter is included in the source download.

7. Audio Output Amplifier

The two-channel audio circuit uses the operational amplifier LM4808, which has two integrated 105 mW headphone amplifiers, suited for low-power portable systems.

The audio is fed to the amplifiers via pins TRDIOC1 (audio left) and TRDIOD1 (audio right).

8. Creating ADPCM Audio

In this chapter we will go from single audio clip WAV-files to a multiple audio clip, target-loadable file, that is both indexed and searchable by audio clip name.

The WAV-format is the default uncompressed audio format for Windows and is supported on almost all computer systems.

8.1 Converting WAV-files to ADPCM-files

Located in the folder ..*\R8CSPB_AN\tools\wav2adpcm*, the program *wav2adpcm.exe* converts a WAV-file into an ADPCM file. The program uses the syntax

>wav2adpcm.exe "filename1.wav" "filename1.aud"

assuming *filename1.wav* exists in the current directory. *filename1.aud* will be created in the current directory and will be ADPCM encoded.

8.2 Creating a Multi Audio File For External Memory

It is more practical to create one custom file containing multiple sound clips for downloading to the external flash than to download them one by one. In addition, it is difficult to keep track of all the file information separately within the target, or from a particular location within the serial memory. The utility *bin_to_mo.exe* in ..*\R8CSPB_AN\tools\wav2adpcm* concatenates multiple ADPCM-files and adds a file information section at the beginning of the resulting file.

The syntax is

>bin_to_mot.exe -m "file1.aud" " file2.aud" " file3.aud" … outputfile.mot

Note that the last argument, the serial memory output file, has no quotes.

The utility attaches a header at the beginning describing each audio clip's name, offset address within the file and its individual length. Each record, one for each sub-file is 32 bytes long and is made up of

Name [24 bytes]:	A string of max 24 characters, including the Null character at the end of the name string.
Start address [4 bytes]:	Start offset-address within the MOT-file.
Size [4 bytes]:	Size of the audio clip within the MOT-file.

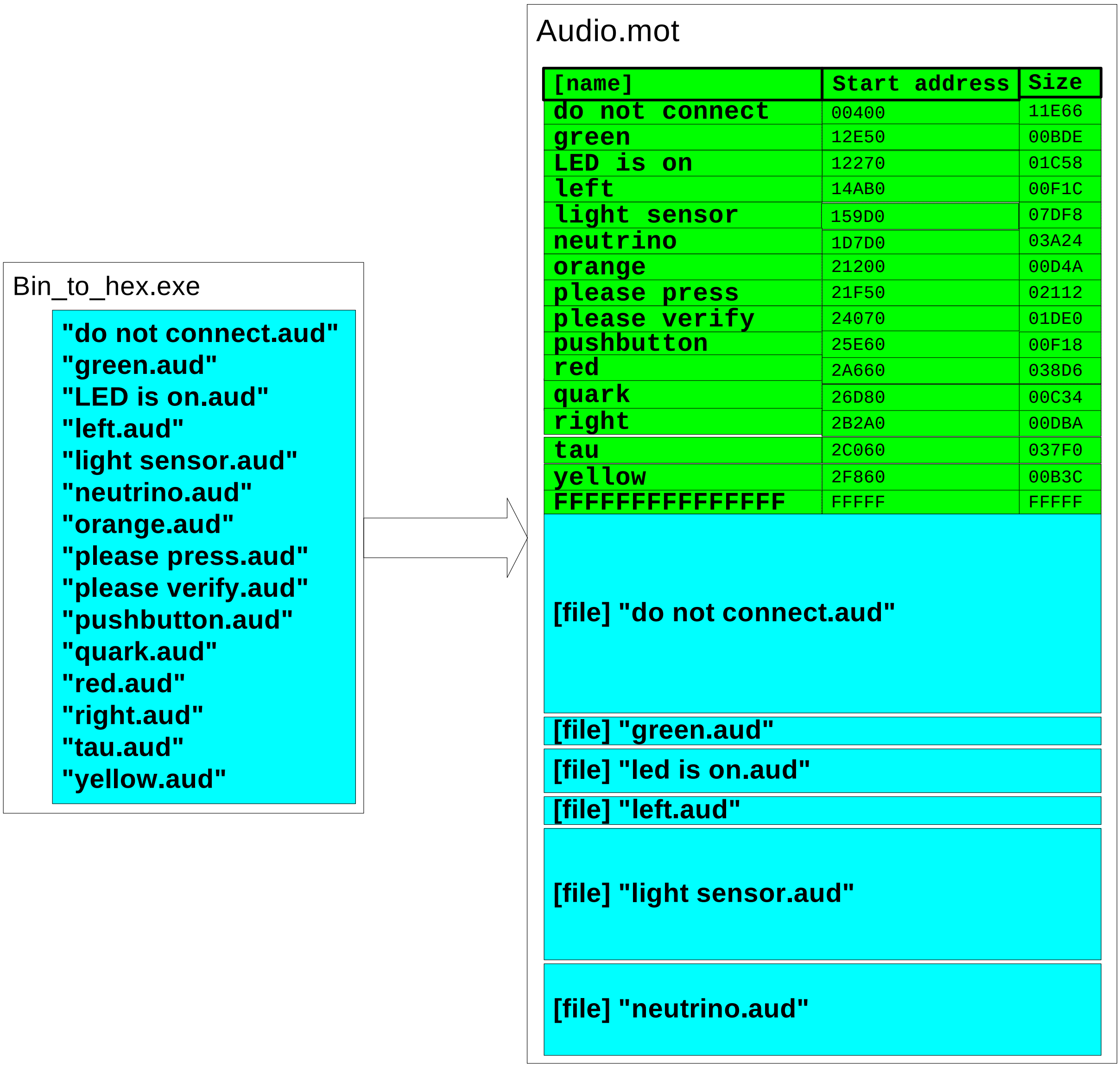

Figure 7. The Bin-to-Mot utility concatenates multiple input files into one output file. It attaches a header (green) at the beginning of the output file so that the individual sub-files can easily be located. The audio clips in this figure are the default clips that come with the SPB when shipped.

In the folder ..*\R8CSPB_AN\tools\wav2adpcm* there are 12 Jetson cartoon WAV-format audio clip examples that can be used to create a file similar to that of Figure 8. There is also a batch file *convert_spb_jetsons.bat* that can be used to quickly convert all audio clips with one action instead of doing it one-by-one as in section 8.1 上の. This makes the whole process convenient by going from multiple wav-files to one 'audio library'. After the batch file is run, only one download action is needed, and the files are neatly organized and searchable.

Open *convert_spb_jetsons.bat* for editing by right clicking it. After opening it, you will see all the commands used to make a loadable audio file. The last line uses *bin_to_mo.exe* to concatenate the ADPCM-files into one larger file.

(a) **Create example audio load file**

Run the batch file to create the same audio load file type as above, but for the provided Jetson audio clips. We will download it in section 10 to the serial flash memory chip.

8.3 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.

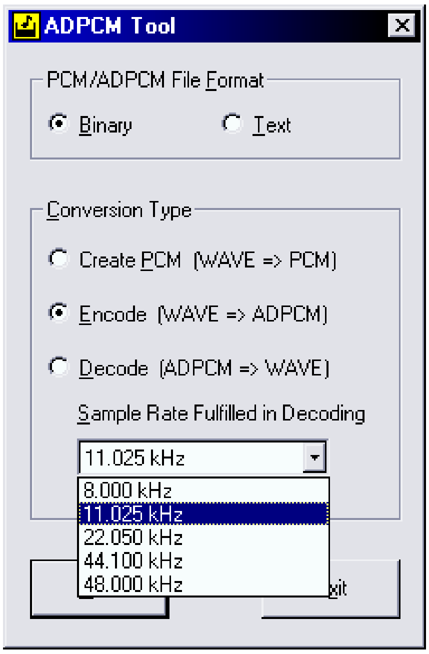

Figure 8. The ADPCM Tool is a GUI for converting back and forth between ADPCM and WAV formats. After selecting what format types to convert between and sampling frequency, press 'Go' to select the file to convert.

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.

9. How Playback Works

9.1 Open HEW and Connect to the SPB

Here are the steps to program the serial chip with audio.

(a) **Open the workspace**

Go to the folder *..\R8CSPB_AN\workspace* and double-clicking on the R8CSPB_AN.HWS file. This will start HEW (High-performance Embedded Workshop).

(b) **Check**

- That version 5.43 or higher of the *Renesas M16C Standard Toolchain* is selected. This can be seen under *Tools -> Administration -> Toolchain*.
- That E8 V2.10 or higher is installed. This is equivalent to version 2.07 or more of the R8C_E8_SYSTEM debugger. Check under *Tools -> Administration -> Debugger Components*.

(c) **Connect to Target**

Select the E8 Debug session (switch from Default if this is the active session).
If you do not see the Emulator Setting window, press connect (*Debug->Connect*).

In the Emulator Mode tab make sure that these settings are selected:
- MCU Group: *R8C/25*
- Device: *R5F21258*.

- Mode: *Erase Flash and Connect*
- Select power supply: *Power Target from Emulator*, and *3.3 V*.

The *Firmware Location* tab determines where the target debug kernel is to be located. Leave the settings untouched. Only select '*Enable advanced setting*' if you have added code to the application.
- Communication baud rate: *500 kbps*.

Click OK.

(d) **Download firmware to target**

Compile and link the source code (F7).

Download the program to the SPB by right clicking and selecting *Debug->Download* (or just double-click on the file).

Now you can run the SPB ADPCM demo with the debugger.

9.2 Read the Serial Flash Audio File Decode and Playback

In debug mode we will now look at how audio clips are selected, watch the data come in from the external memory to the SPI interface, decode the SPI buffer ADPCM-data, and write it to the PWM output.

(a) **Run the code**

In HEW with the target connected as above, select *Debug->Reset+GO* (Shift+F5). Make sure the code is running by checking that the red and yellow LEDs are blinking.

(b) **Connect headphones/speakers to the SPB.**

(c) **Playback**

Press the board's pushbutton to play back one file in the serial flash.

Pressing it again while audio plays on one channel will cause another audio clip to play on the other channel.

(d) **Modify play order**

Open the file Streamingaudio.c and find the string array audio_file_array[].

Rearrange the order in which the audio files are listed and thus played.

Recompile (F7), press *Reset->Go* (Shift+F5), and listen to the playbacks by pressing the pushbutton.

(e) **Observe search for audio clip**

Go to the function *FlashFind()* in StreamingAudio.c, this should be around line number 360.

Set a breakpoint in the function right after the call to *FlashReadData()*, by double clicking in the event gutter to create a blue dot. (A red breakpoint in the next gutter is written to flash memory and is slower, but allows for more breakpoints.

Press the pushbutton (with the code running). The debugger should now stop at the breakpoint.

Highlight the *SPI_buffer* variable and add it to the watch window (Ctrl+W creates the watch window, then drag the variable to it.)

Press the small +sign in the watch window to see the content. This is the ADPCM data for this audio clip.

10. Loading New Audio, HEW Target Server

We will now look at how to load a multiple audio clip file, such as the one created in section 8.2 上の, to the SPB's serial flash chip. This will be done using Hew Target Server. HTS comes with a HEW installation and provides an extension to HEW so that Windows application development tools available on the market can send commands to - and receive results - from HEW. HEW can in this way operate in conjunction with other applications.

Included in the software download package is this *Serial Memory Loader* HEW workspace which uses HTS to program the serial flash audio memory. This workspace is located in the folder *..\R8CSPB_AN\tools\SerialMemLoader\YR8CSPB*. It is opened by the GUI program

HTS_External_Programmer.exe included in this package. The *SerialMemLoader* workspace will be opened and execute on the SPB. It will and receive the MOT-file data from HEW Target Server. Using the SPI interface (clocked serial interface) *SerialMemLoader* will be directed by HTS to write the file to the serial flash chip blocks.

Here are the steps to erase and write new audio clips to the external serial memory chip.

(a) **Configure HEW Target Server**

Run the batch command file *C:\Program Files\Renesas\Hew\registerserver.bat* that should be on your PC if you have installed HEW and any Toolchain.

(b) **Register HEW Target Server**

From within HEW, go to Tools -> Administration -> Search Disk -> Start, select HEW target server -> Register. Press ‘Close’ then ‘OK’.

(c) **Close HEW before proceeding**

(d) **Load New Audio**

Start *HTS_External_Programmer.exe* from ..\R8CSPB_AN\tools\SerialMemLoader\Loader-GUI.

Press Select Workspace and browse to ..\R8CSPB_AN\tools\SerialMemLoader\YR8CSPB and click on SerialMemLoader.hws.

Press ‘Select File’.

Browse to the file you want to download to the serial flash. For this application note we will use the demo *spb_audio_jetsons.mot* which you create by running *convert_spb_jetsons.bat* in section 8.2 上の.

Press ‘Open Workspace’

Connect as before when the Emulator Settings window pops up. If the Emulator Settings window does not appear, switch from the default session to the E8 session

Figure 9. Switching from the default session to the E8 session

If the debugger pops open a window asking you for the location of a source file, locate the file from the correct directory and press OK.

The Open Workspace button should change its name to Program after connection. Press it. There is about a 10 second delay before programming starts while the serial flash memory chip is erased. The serial flash chip should then be written to.

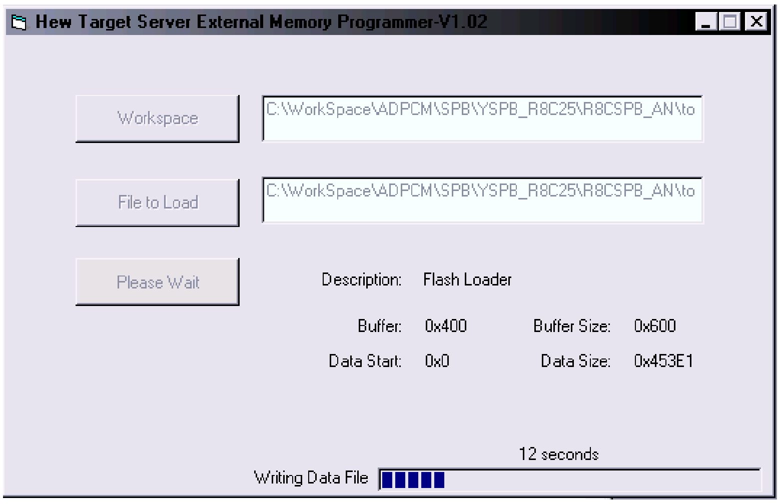

Figure 10 HTS_External_Programmer.exe. After erasing is finished, programming takes place.

If the Disconnect button on HTS_External_Programmer is highlighted the new sound file should be in place in the serial flash chip. Do not close HEW.

Press Disconnect.

Close HTS_External_Programmer which will close HEW.

10.2 Playback the New Audio tracks

(a) **Re-open the workspace.**

(E.g. Start HEW, File -> Recent Workspaces -> …)

(b) **Change audio file names and number of audio files**

We need to change audio_file_array[] and NUMAUDIOFILES to reflect the new content in the serial flash data. Examine the batch file's output names to get the correct name strings. Only include the name of the file, that is, without the '.aud'. In our case we need to set

```
#define NUMAUDIOFILES to 11
/* Array with names of audio files in media (serial flash) to be played (in
order). */
char far * const audio_file_array[] =
{
   /* 11 kHz */
   "jetsnbel",
   "greatest",
   "jetphone",
   "lateschool",
   "workwork",
   /* 16 kHz */
   "report2myoffice",
   "importantassignment",
   "wrongbutton",
   "reallydaddy",
   "jetcar",
   "fired"
};
```

Download the X30-file again.

Change the sampling frequency to 16 kHz to hear the last 6 audio clips correctly. This is explained in 5.2 上の. When you have done this, the first clips will be played at the wrong speed instead.

11. The ADPCM Library

The object library *s2_r8ctiny_e_v200.lib* must be linked together with the code in order to decode ADPCM data. Before calling any of the library functions, an ADPCM structure variable of type *T_ADPCM*, found in *adpcm4.h*, must be allocated.

11.1 The ADPCM Library Data structure

ADPCM structure variable	Explanation
input	Top address of ADPCM data area.
output	Top address of expanded PCM data area.
Size	Number of samples to be decoded.
id	Do not set. For internal library use.
vp	Do not set. For internal library use.

There are two function calls to the ADPCM library. One function for initializing the library with the data structure's location, and one function does the actual decoding.

11.2 ADPCM Initialization

The ADPCM data structure's address must be provided to the library by means of the initialization function.

```
void adpcm_init4( T_ADPCM *pADPCM );
```

This function must be executed prior to any ADPCM data decoding calls. All you need to provide is the address to the structure. The structure members need not be assigned any values until actual decoding starts.

11.3 ADPCM Decode

After the ADPCM structure has been initialized, decoding can be done by any number of calls to the decode function.

```
int adpcm_decode4( T_ADPCM *pADPCM );
```

Before <u>each</u> call to the function, the ADPCM structure members input, output and size must be set.

- Input Set this to the beginning, or top address, of the ADPCM data area to be decoded.
- Output Set this to the address where you want the resulting PCM data to be output.
- Size Number of samples to be decoded. It must always be an even number.

12. Decoding Priority and Design

The code to extract ADPCM audio data from the SPI interface and convert it to PCM data resides inside the PWM interrupt in the sample code. The reason is explained here.

Without a reload buffer the PWM channel must be serviced without delay. This is explained in 14. Since the CPU could potentially be busy executing other functions - or interrupts - when more PCM data is needed, we must ensure that decoded data is always available.

The solution uses two PCM buffers, each containing multiple samples. See Figure 11. When one PCM buffer is empty, the other buffer must be refilled with a higher priority than that of the main application. The 'main' non-interrupt source code has no priority at all unless an RTOS is used. Therefore decoding must take place within a high priority interrupt routine.

Figure 11 shows the structure of the PWM timer audio interrupt. As decoding takes place inside the interrupt routine, we still *must* make sure that data samples are written to the PWM at the right time, even it the buffer being refilled is not full yet. The interrupt routine is therefore reentrant, that is, it can interrupt itself, write to the PWM output (which is done right at the beginning of the interrupt) and then return (to itself) and continue to decode data and write it to the partially filled buffer.

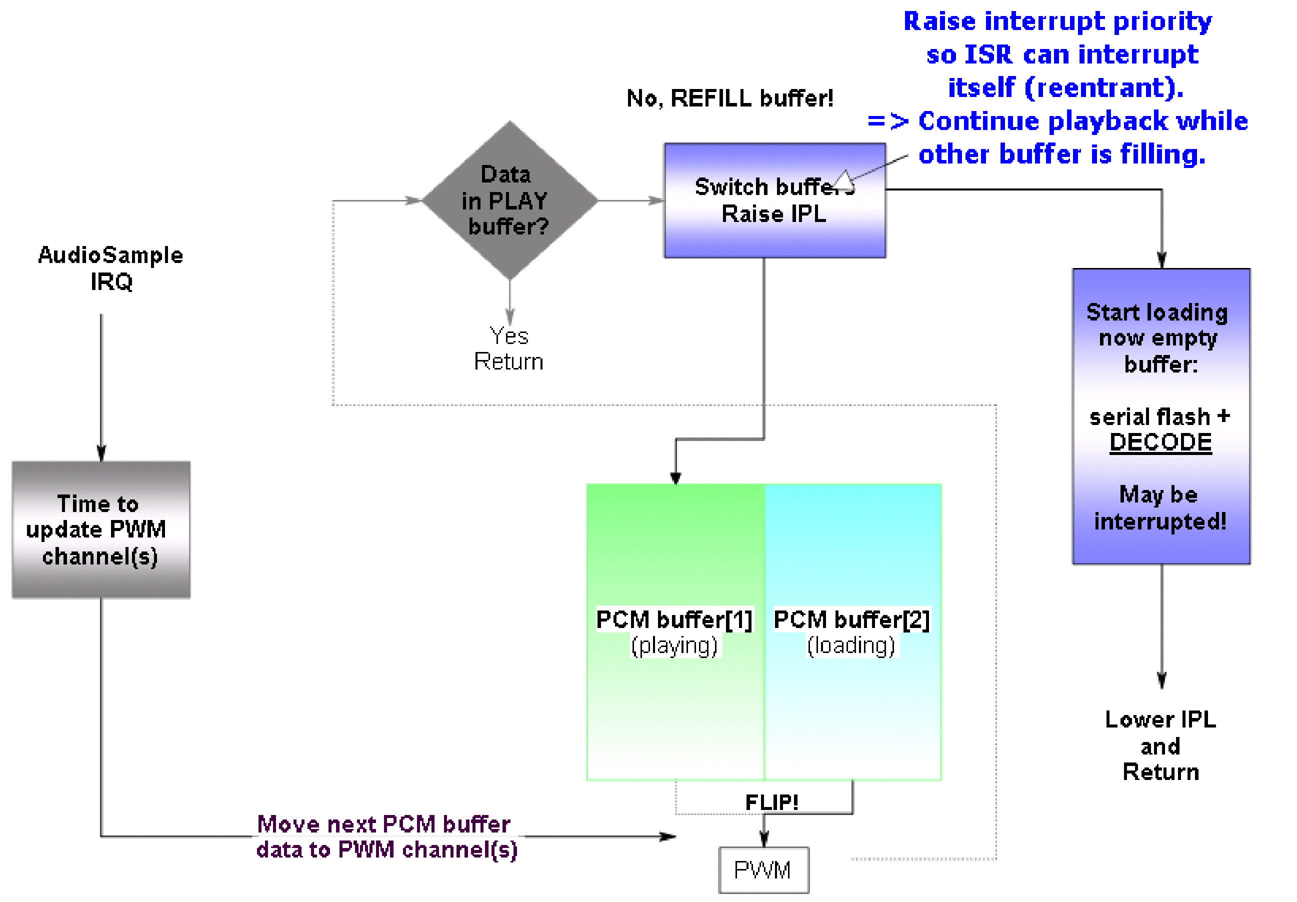

Figure 11. Interrupt activity changes when one PCM buffer becomes empty. The interrupt then decodes new data from the SPI interface and writes it to the empty buffer until it is full with new ADPCM audio data samples.

13. Processing Requirements

The approximate CPU-load, as a result of the PWM interrupts and audio decodes, can be found by adding a long counter variable to the main program's while(1) loop, and just increment the counter. This loop should not do anything else. After running the application for a certain time, i.e. 10 seconds, the counter is read. After this value is noted, add the interrupt code and this time run the audio. After running the application for the same amount of time, the counter is read again. The approximate CPU load is then

CPU load = (counter value idling - counter value running audio) / counter value idling

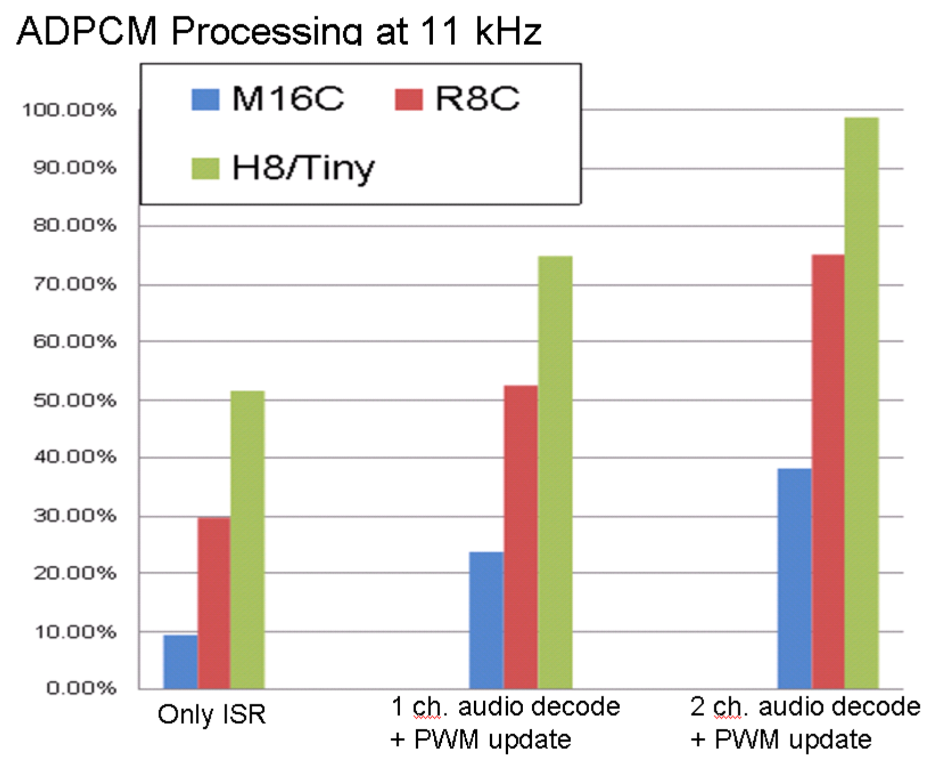

Figure 12. ADPCM processing benchmarks showing CPU load using 11 kHz audio. For the R8C/25 we see that only 25% of the CPU is available for other tasks when audio is playing on two channels (stereo). To achieve better duty cycle values use the alternative method discussed in 14.

If instead PWM buffered mode (see next section 14 下の) were to be used, the MCU need only drop in a new sample each sample period, instead of as for the default solution at an exact multiple of the PWM frequency at the beginning PWM interrupt. This will result in a decrease the CPU load compared to above data.

The M16C uses this buffered method by default in its SPB sample code (not included in this application note).

14. Alternative Design Lower Interrupt Frequency and CPU load

With the system configured to run in Reload Buffer Mode, the interrupt frequency and therefore the CPU load, can be lowered. This alternative design is outlined here.

In the current solution which is non buffered, the PWM interrupt must be taken and the service routine immediately write the new PWM value - at the beginning of the PWM timer cycle. If the new sample were written randomly within the PWM cycle, it would sometimes occur after the PWM counter passes the new sample's value, resulting in a faulty 100% pulse width in that pulse. This is audible and would lower the quality of the sound. The PWM frequency must therefore be an exact multiple of the sample frequency. In addition, since the PWM interrupt must be taken instead of just a sample frequency interrupt, the number of interrupts is much higher in the current solution; the total number being the PWM frequency instead of the sampling frequency.

Using a reload buffer, one per channel, the sample update can be run by a 'regular' timer at the sampling frequency, and not the PWM frequency, since the new sample can be written at any time within the PWM cycle. This means that the PWM interrupt need not be used at all. If the PWM interrupt (a multiple of the sampling frequency) is not used, the number of interrupts is reduced considerably. This reduces the CPU load.

TimerRA for example can be set up to interrupt at the sampling frequency. At the interrupts it should write to the registers TRDGRD0 (buffer for TRDGRB0) and TRDGRD1 (buffer for TRDGRB1) if two audio channels are used. To summarize:

- TRDGRD0 can be used as buffer register for the TRDGRB0 register to get one buffered audio channel on pin TRDIOB0.
- TRDGRD1 can be used as buffer register for the TRDGRB1 register to get the second buffered audio channel on pin TRDIOB1.

This should increase performance considerably. This is not done in the sample code since the audio amplifier input is connected to TRDIOC1 and TRDIOD1.

Website and Support

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

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

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Feb 1, 2010	—	First release, after RTA AE review.
1.01	Mar 26, 2010	1	Title M16C removed.
		3	Link to RenesasRulz modified.
		9	5.2.b changed.

Notes regarding these materials

1. This document is provided for reference purposes only so that Renesas customers may select the appropriate Renesas products for their use. Renesas neither makes warranties or representations with respect to the accuracy or completeness of the information contained in this document nor grants any license to any intellectual property rights or any other rights of Renesas or any third party with respect to the information in this document.
2. Renesas shall have no liability for damages or infringement of any intellectual property or other rights arising out of the use of any information in this document, including, but not limited to, product data, diagrams, charts, programs, algorithms, and application circuit examples.
3. You should not use the products or the technology described in this document for the purpose of military applications such as the development of weapons of mass destruction or for the purpose of any other military use. When exporting the products or technology described herein, you should follow the applicable export control laws and regulations, and procedures required by such laws and regulations.
4. All information included in this document such as product data, diagrams, charts, programs, algorithms, and application circuit examples, is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas products listed in this document, please confirm the latest product information with a Renesas sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas such as that disclosed through our website. (http://www.renesas.com)
5. Renesas has used reasonable care in compiling the information included in this document, but Renesas assumes no liability whatsoever for any damages incurred as a result of errors or omissions in the information included in this document.
6. When using or otherwise relying on the information in this document, you should evaluate the information in light of the total system before deciding about the applicability of such information to the intended application. Renesas makes no representations, warranties or guaranties regarding the suitability of its products for any particular application and specifically disclaims any liability arising out of the application and use of the information in this document or Renesas products.
7. With the exception of products specified by Renesas as suitable for automobile applications, Renesas products are not designed, manufactured or tested for applications or otherwise in systems the failure or malfunction of which may cause a direct threat to human life or create a risk of human injury or which require especially high quality and reliability such as safety systems, or equipment or systems for transportation and traffic, healthcare, combustion control, aerospace and aeronautics, nuclear power, or undersea communication transmission. If you are considering the use of our products for such purposes, please contact a Renesas sales office beforehand. Renesas shall have no liability for damages arising out of the uses set forth above.
8. Notwithstanding the preceding paragraph, you should not use Renesas products for the purposes listed below:
   (1) artificial life support devices or systems
   (2) surgical implantations
   (3) healthcare intervention (e.g., excision, administration of medication, etc.)
   (4) any other purposes that pose a direct threat to human life
   Renesas shall have no liability for damages arising out of the uses set forth in the above and purchasers who elect to use Renesas products in any of the foregoing applications shall indemnify and hold harmless Renesas Technology Corp., its affiliated companies and their officers, directors, and employees against any and all damages arising out of such applications.
9. You should use the products described herein within the range specified by Renesas, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas shall have no liability for malfunctions or damages arising out of the use of Renesas products beyond such specified ranges.
10. Although Renesas endeavors to improve the quality and reliability of its products, IC products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Please be sure to implement safety measures to guard against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other applicable measures. Among others, since the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
11. In case Renesas products listed in this document are detached from the products to which the Renesas products are attached or affixed, the risk of accident such as swallowing by infants and small children is very high. You should implement safety measures so that Renesas products may not be easily detached from your products. Renesas shall have no liability for damages arising out of such detachment.
12. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written approval from Renesas.
13. Please contact a Renesas sales office if you have any questions regarding the information contained in this document, Renesas semiconductor products, or if you have any other inquiries.