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.

Notice

1. All information included in this document 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 Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.

   "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics 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 appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

R32C/100 Series

Interfacing the R32C/111 to an I²S Audio Codec

Introduction

I²S (Inter-IC Sound) is a serial bus standard primarily designed to allow digitised audio to be transferred between microcontrollers, processors and DSPs and audio CODECs (coder/decoders).

The R32C/111 Group of microcontrollers does not include a dedicated hardware peripheral, such as a Serial Sound Interface, capable of supporting I²S. It is, however, possible to implement an I²S, or similar digital audio interface, using general purpose microcontroller peripherals and software, with minimal impact on system performance.

This Application Note explains how to utilise a serial UART, multifunction timer channels and the Direct Memory Access Controller (DMAC) to implement a digital audio interface to a Wolfson Microelectronics CODEC. Audio samples are able to be recorded through a microphone and then to be played back through attached headphones.

Target Device

R32C/111 Group

Contents

1. Introduction to I²S

The audio data in an I²S interface is transferred over a 3 wire bus, consisting of a Serial Clock (SCK) line, a Word Clock, or Word Select (WS) line, and a Serial Data (SD) line. Commonly each bit of data is presented on SD by the transmitting device on the falling edge of SCK. The data is then transferred to the receiving device on the rising edge of the SCK (See Figure 1).

The transmitting device sends a word of data corresponding to an audio sample which can be of any length, but typically 16, 24 or 32 bits, with the most significant bit (MSB) transmitted first. As the least significant bit (LSB) is presented on SD, WS changes state to indicate to the receiver that the next bit to be transmitted is the MSB of the next word or sample. Usually successive words relate to the left and right channels in a stereo audio stream, hence WS is sometimes referred to as the Left-Right Clock (LRCLK).

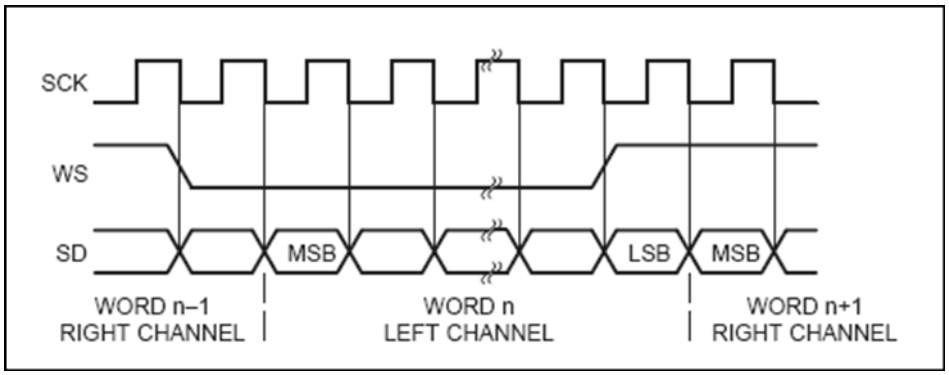


Figure 1: I2S Interface Timing

The use of the WS or LRCLK signal allows transmitters and receivers with different native word length to be used seamlessly in an audio system. If the transmitter sends words longer than the receiver can handle, the least significant bits of the sample are discarded. Conversely, if the receiver is expecting a longer word than is transmitted, the least significant bits are 'zero-filled'.

This means, for example, a 24-bit audio sampling analogue to digital converter (ADC) can send data to a processor that stores the samples as 16-bits and the 8 LSBs are discarded. The samples could then be sent to a 24-bit DAC to reproduce the audio, and the 8 LSBs will be auto-filled with zeros by the receiving DAC. This obviously introduces losses into the system, but the components do not need to be specially programmed to send or receive data of a defined length, as the I²S protocol handles this automatically.

While the SD signal is by definition generated by the transmitting device, the WS and SCK signals can be generated by the transmitter, the receiver or another controller in a system. Most I²S compatible devices can be programmed to either generate or accept WS and SCK, whether they are transmitting, receiving or both.

The I²S Specification defines limits for signal set-up, hold and delay times in relation to the serial clock period, but not the clock frequency itself. This allows the use of a wide range of audio sampling frequencies and resulting SCK bit rates to be used in an I²S system.

2. Configuring the R32C/111 Peripherals

The Renesas R32C/100 Series of microcontrollers includes devices designed for industrial, consumer and automotive applications. Designed as a 32-bit upward compatible version of the popular M16C architecture, its large on-chip memory integration, powerful CPU including Floating Point Unit and extensive on-chip peripherals make it suitable for general purpose applications that include some audio processing tasks.

The R32C/111 Group does not include dedicated hardware to support I²S interfacing. It is possible, however, to utilise general purpose peripherals to emulate such an interface and allow communication with external I²S compatible devices.

2.1 Generating the Serial Clock

The R32C/111 Group devices each include two 16-bit multifunction timer modules, Timer A and Timer B, with a total of 11 channels. Each channel can be configured in different modes, enabling functions such as periodic interrupt generation, event counting, pulse period measurement and pulse generation.

In Timer Mode a value is written to the Timer xi Register (where x = A or B and i = channel number) and is decremented each count of the timer clock source. When the timer underflows (0x0000 -> 0xFFFF) the initial value is automatically reloaded and the state of the TxiOUT pin is optionally inverted.

The CODEC chosen for the example application configuration has 24-bit ADCs and DACs and a programmable sampling rate of between 8 kHz and 96 kHz. The digital audio word length is also selectable to 16, 20, 24 or 32-bits. In our example code, we have chosen an audio sampling rate of 8 kHz and a word length of 16-bits. There is no MCU performance barrier to higher sampling rates and longer word length, however increasing either would create more sampling data, and therefore a larger storage requirement for a given audio recording time.

When recording audio, 16-bit samples are acquired by the CODEC and transmitted to the microcontroller. As there is both a left and a right sample word, there will therefore be 32 serial clock pulses every sample period. This gives a desired serial clock SCK period of:

$$T_{SCK} = 1 / (8000 \times 32) = 3.90625\mu s$$

In the example configuration we use Timer A, channel 0 (TimerA0) to generate the Serial Clock, SCK (sometimes also referred to as the Bit Clock, BCLK). The 16-bit value to be written to the TimerA0 Register is derived from the equation above, the system clock frequency and the timer clock source.

To generate the exact SCK period required for 8 kHz sampling, an appropriate clock source is needed for the R32C microcontroller. Using a 12.288MHz crystal, as suggested as suitable for 8kHz sampling frequency in the Wolfson CODEC datasheet, for both CODEC and microcontroller we can generate a CPU clock frequency through the PLL of (4 x 12.288) = 49.152MHz, and a peripheral clock of (CPU clock/2) = 24.576MHz

If we supply the peripheral clock directly to TimerA0, the timer will decrement with a period of:

$$T_{TIMERA0} = (1/ \text{Peripheral Clock}) = (1/24.576e6) \approx 40.69ns$$

To generate the underflow period for TimerA0 of half the desired SCK period we therefore need an initial value of:

$$(T_{SCK} / 2) / T_{TIMERA0} = ((3.90625e\text{-}6) / 2) / (4.069e\text{-}8) = 48$$

By configuring TimerA0 as described with a reload value of 47 (for 48 clock periods before underflow) and starting the timer (by setting the TA0S bit) we can generate the SCK or BCLK signal continuously and without further software intervention.

2.2 Generating the Word Select or Left-Right Clock

As can be seen from *'Figure 1: I2S Interface Timing'*, for 16-bit audio samples the Word Select signal is inverted every 16 serial clock pulses. We can again utilise the R32C/111 Group 16-bit multifunction timers to generate this signal.

In Event Counter Mode a value is written to the Timer xi Register (where again X = A or B and i = channel number) and can be programmed to decrement every time a pulse is seen on the TxiIN pin. When the timer underflows the value written is automatically reloaded, and again the state of the TxiOUT pin can be optionally inverted.

In our example application configuration we use Timer A channel 1 (TimerA1) to generate the Word Select signal, WS.

To count serial clock pulses we can connect the TA0OUT pin used for the SCK signal to the TA1IN pin of the second timer channel. Alternatively, and to avoid having to use the TA1IN pin, the Event / Trigger Select bits for TimerA1 can be set to Select the overflow (or underflow) of TimerA0 (See R32C/111 Hardware Manual Figure 16.9). Note that to make clearer the method of generating the WS signal in the example configuration this option is not employed.

By configuring this second timer channel as described above and using an initial value for TimerA1 Register of 15 (0x000F) we can generate the WS signal automatically and without further software overhead for as long as the SCK signal is generated.

In the I²S specification, the first bit of valid data is not available on the Serial Data line until the second rising clock edge after the WS signal transition. This delay was originally intended to allow time for the receiving device to prepare for reception of the next sample. However this is rarely necessary in modern circuit design. There is sufficient time at most sampling speeds for the next sample to be received successfully with a transition immediately before the MSB of the next sample is presented on the Serial Data line.

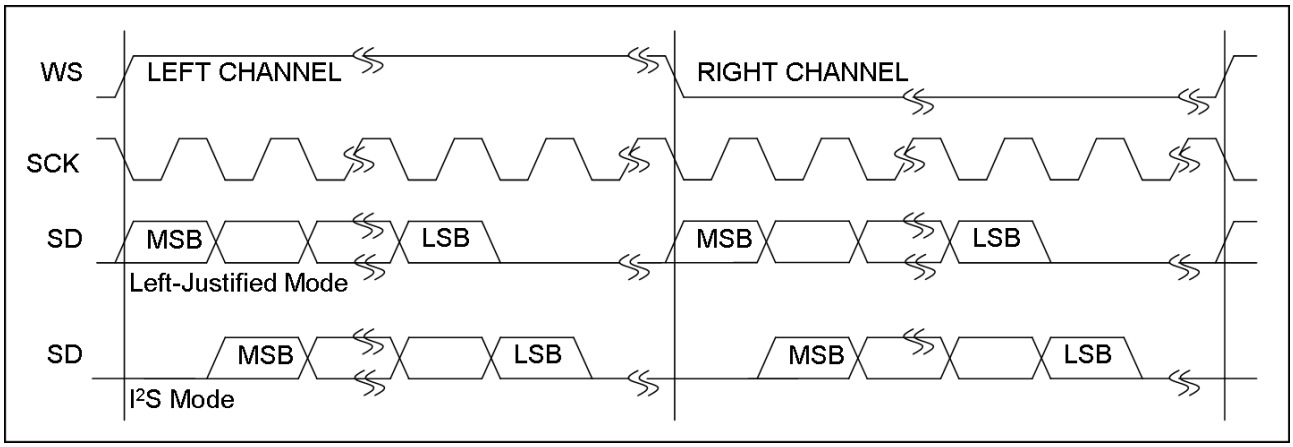


Figure 2: Left Justified Mode versus I²S Mode

Most I²S devices allow a 'left justified' mode to be used for the audio serial communications. This is identical to 'true' I²S communications, except the one clock period delay between the WS or LRCLK signal transition and the beginning of the next word is omitted. In our example application configuration the Left Justified Mode of the CODEC is used to simplify the timing on the audio interface.

If communications compliant to the original I²S Mode is required, a one clock period delay before transmission and reception of the MSB needs to be generated. This could be done by enabling the interrupt on the timer generating the Serial Clock signal and enabling the serial interface in the first timer Interrupt Service Routine. If there are no gaps between successive words on the serial data line(s), the serial interface can then be left enabled, the interrupt left disabled and communications will continue uninterrupted.

Implementation of this 'true' I²S Mode is not done in the example application code as it is unnecessary for communication with most I²S compatible devices.

Note also that Figure 2 shows that a delay of undefined length is allowed following each transmitted word in the I²S specification, before the transition on the WS line and commencement of the next word. This variable delay is not supported in this implementation, and continuous transmission and/or reception of data words is assumed.

One further point regarding the generation of the WS signal should be noted. The TimerA1 output will start as a 'L', indicating the Right channel of the stereo audio. As the order in which stereo samples are acquired by the CODEC ADC is Left channel and then Right channel, the first sample will not have a corresponding Left channel sample. As playback follows the same format, this has no effect on the audio quality or on storage and retrieval of samples.

2.3 Configuring the Serial Interface

On the R32C/111 Group of microcontrollers up to 9 Serial Interface units (UART0 to UART8) are available to the user. These can be configured in different modes to support a variety of synchronous and asynchronous serial communications formats.

In Synchronous Serial Interface Mode one bit of data is transmitted and received for every serial clock (CLK) pulse. This serial clock can be sourced internally, derived from the system clock and generated by the UART Baud Rate generator, or sourced externally from another clock source through the CLKi (where i = 0 to 8) pin.

2.3.1 Data Reception

The UARTi Receive Register is buffered by the UARTi Receive Buffer (UiRB). Immediately after a byte has been received, it is transferred to the UiRB Register so that the next byte can be received without a delay being required while the received data is read by the user code.

Furthermore, a UART receive interrupt request can be generated when the UARTi Receive Register contents are transferred to UiRB. If the UiRB is read immediately in response to this interrupt before the next byte has been completely received, continuous reception of data can take place.

If we configure UART0 in Synchronous Serial Interface Mode and select external clock, the TimerA0 output generating the SCK signal can be connected to the CLK0 pin. Serial transfers can then be synchronised with both the bit clock (SCK) and Word Select clock (WS). As long as the Receive Buffer is read quickly enough on every received byte, this synchronization will be maintained for as long as the SCK signal is generated.

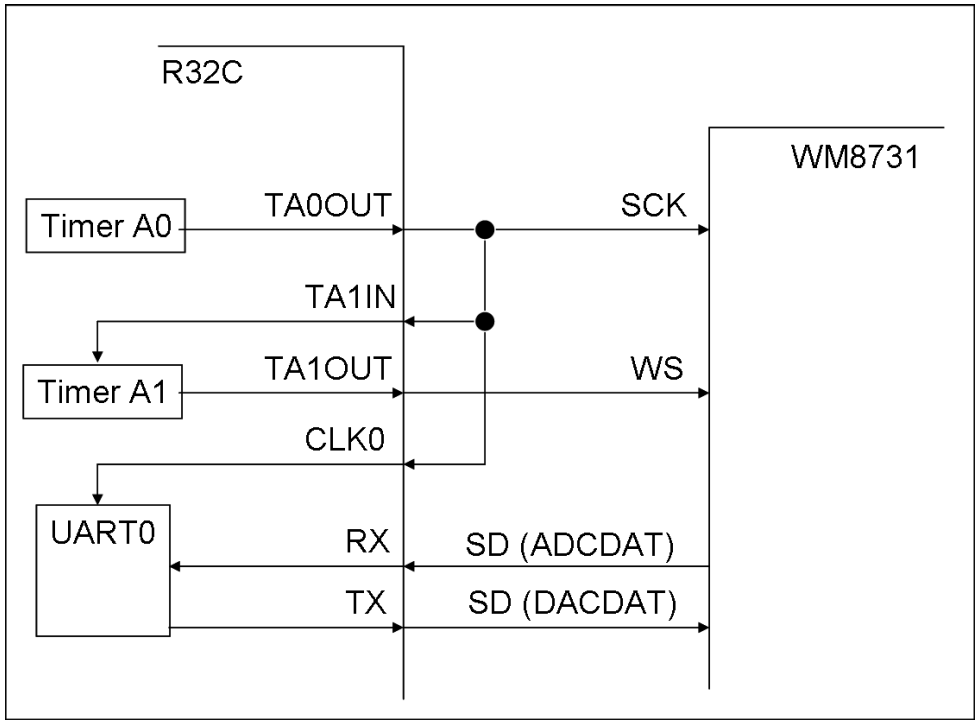


Figure 3: R32C Peripheral Configuration

During reception of data a dummy byte of data is normally required to be written to the UART Transmit Buffer U0TB before each byte is received (see R32C/111 Group Hardware Manual, Figure 18.21). Using Continuous Receive Mode this is not necessary, as reception is enabled by each read of the U0RB register. To enable initial reception the U0RB register therefore needs to be read before the SCK signal is applied to the CLK0 pin.

2.3.2 Data Transmission

The UARTi Transmit Register is buffered by the UARTi Transmit Buffer (UiTB). While one byte of data is being transmitted the next byte of data to be transmitted can be loaded into the UiTB, ready to transmit when the current byte has been transmitted.

Furthermore, the UART transmit interrupt can be configured to generate an interrupt request when the UiTB is empty. If the transmit buffer is written to immediately in response to this interrupt with the next byte to be transmitted, continuous transmission of data can take place.

2.4 Configuring the DMA

Use of the Direct Memory Access Controller (DMAC) on the R32C can significantly reduce the processing required by the CPU in an application. The DMAC allows transfer of data to and from registers and memory without interruption of the application code being executed by the CPU.

The R32C/111 Group has 4 DMA channels, each of which can be configured to transfer bytes, 16-bit words or 32-bit long-words, either individually or in blocks. Both source address and destination address can be anywhere within the 32-bit memory map. Source and destination addresses can also be specified as either fixed or forward, where the address is either constant or is incremented to the next byte, word or long-word after each transfer.

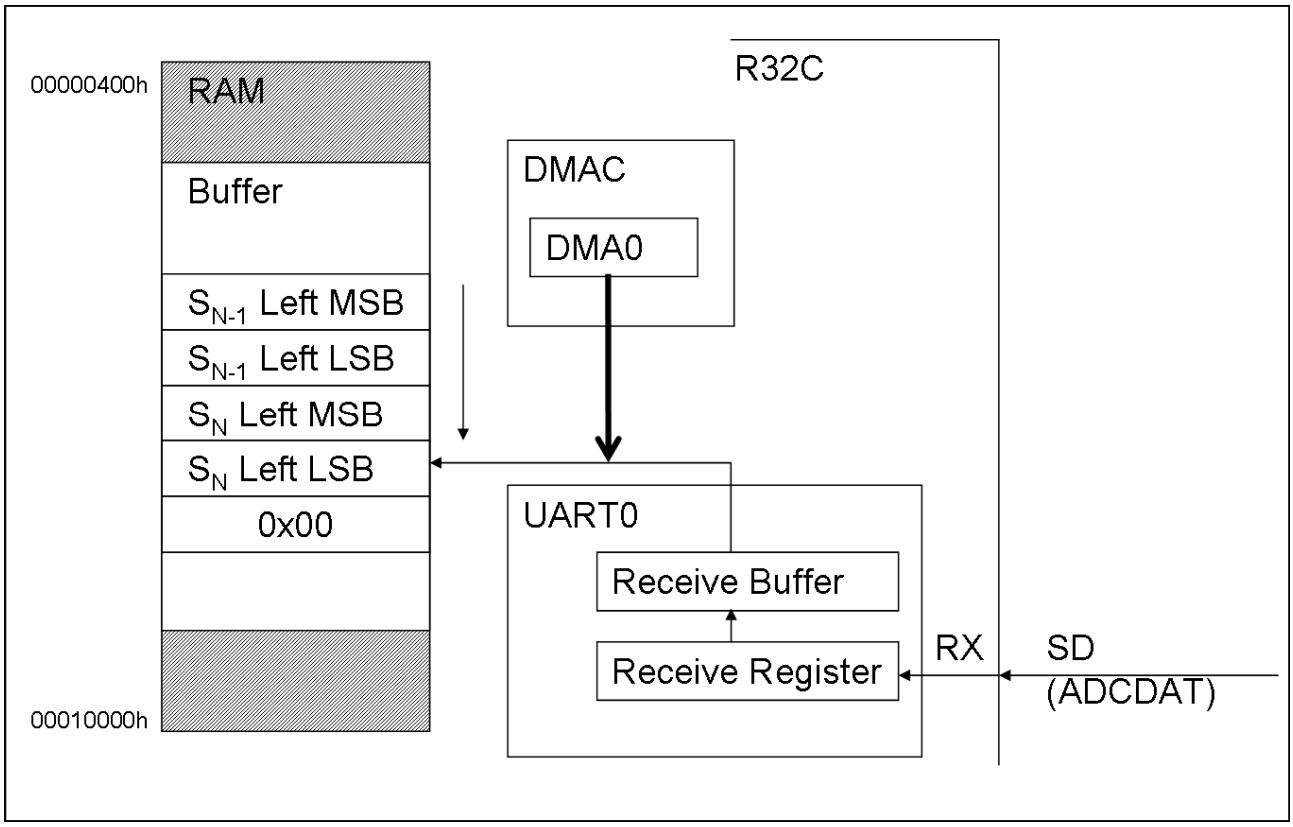


Figure 4: DMA Configuration for Recording Audio

Each DMA can be configured to transfer data in response to a particular peripheral interrupt request by selection of the trigger source using the DMA Request Source Select Registers.

2.4.1 Audio Recording

We can configure a DMA channel to transfer the contents of the UART0 Receive Buffer U0RB into RAM in response to the UART0 receive interrupt request. This will ensure the U0RB register is read immediately, and the receiving of bytes relating to audio samples can continue without interruption and loss of synchronisation. If we specify the destination address as incrementing, we can fill a buffer in RAM with the audio sample data.

Note that in our example application configuration, 16-bit samples are used, and the most significant byte is transferred and stored first. As this most significant byte is stored at the lower address in memory, the samples are therefore stored in 'big-endian' format. The R32C is a 'little-endian' architecture, where the byte stored at the memory address of a multi-byte data entity is interpreted as the least significant byte. The two bytes of each sample would therefore need to be reversed if any processing of the data was to be done. In the example application code, transfer of the sample data to the serial port for playback is done in the same order as receiving it during recording therefore this reversing of data bytes is not necessary.

2.4.2 Audio Playback

For playing back an audio recording through the simulated I2S interface it is necessary to transfer the audio samples back to the serial interface transmit buffer continuously and without interruption to maintain synchronisation between the serial data and WS (or LRCLK) signal.

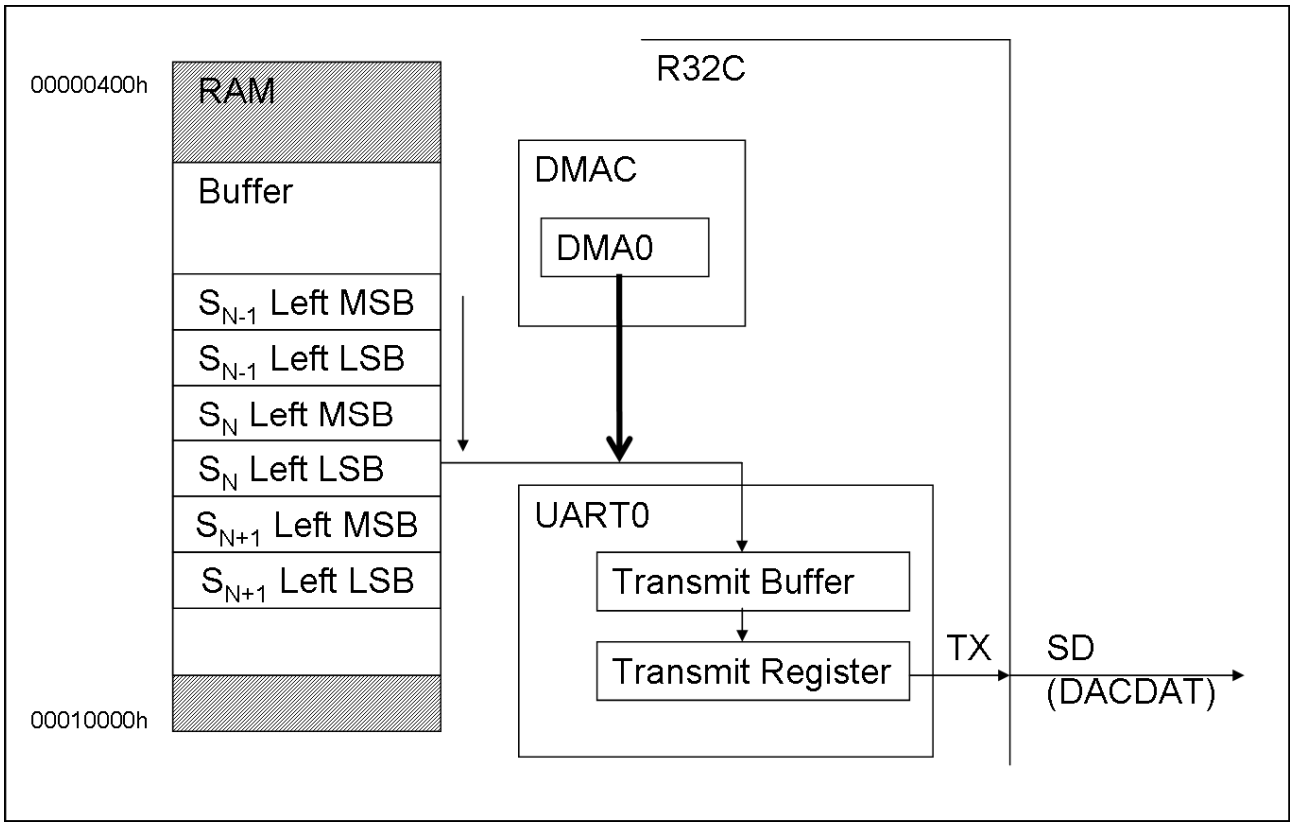


Figure 5: Configuration of the DMA for Audio Playback

This can be done by configuring a DMA channel with forward source, to automatically increment through the sample bytes stored in the RAM buffer, and fixed destination of the UART transmit buffer. The trigger source for the DMA is selected as the UART transmit interrupt request. Each time transmission of a byte of data is completed and the byte waiting in the transmit buffer is loaded into the transmit register, the next byte is automatically moved from the RAM buffer to the U0TB transmit buffer by the DMA to await transfer.

In our example application code the same DMA is used for both record and playback. This is possible as these activities do not happen simultaneously, and the DMA can be reconfigured each time it is used. In some audio signal processing applications simultaneous receiving and transmitting of audio samples may be required. In this case 2 separate DMA channels will need to be used.

3. Communication and Control of the Audio CODEC

In audio applications the I²S interface is used exclusively for serial digital audio data. In most cases an I²S compatible peripheral device will also needs to be configured for the task it is doing, and parameters such as sampling speed and clock configuration will need to be set. In the example configuration we use a WM8731 CODEC from Wolfson Microelectronics. This device allows programming of functionality through a separate Software Control Interface. On the WM8731 CODEC the state of a Mode pin is used to select between either 3-wire (SPI compatible) or 2-wire (I²C compatible) synchronous serial communications for this software control.

3.1 Simple I²C Communication

In our example application we use another Serial Interface on the R32C, UART2, configured in Special Mode 1 (I²C Mode) to perform this configuration and control of the WM8731. Ten 9-bit registers with 7-bit addresses from 0x00 to 0x09 are used to control the digital audio interface format, ADC, sampling rate, input selection, clock configuration, output volume and various other parameters including power down of different parts of the device. Setting of these registers is done by sending the 7-bit address followed by the 9 bits of configuration data for each register. A further register at address 0x0F is used to reset the WM8731 to its default state.

The example code for this I²C communication uses a subset of the 'I2C_Memory' Sample Code included as part of the R32C/111 Renesas Starter Kit HEW software installation. Only transmission routines are needed for our CODEC configuration, as the WM8731 registers are write-only, and no verification of contents can be done.

4. Hardware Configuration

In order to prove the functionality of the example software and allow it to be demonstrated, standard off-the-shelf hardware was chosen for the major component circuit boards.

The Renesas Starter Kit (RSK) for R32C/111 includes the R32C microcontroller itself, as well as supporting circuitry, several switches, LEDs and the E8a debugger interface connector. The specific device installed, part number R5F64112DFB, integrates 512k bytes of on-chip Flash Memory and 63k bytes RAM. The device also integrates many general purpose peripherals including 9 serial interface channels, 11 channels of 16-bit multifunction timer and 4 DMA channels.

For the audio CODEC, an Evaluation Board for the WM8731, part number WM8731EV1, was sourced from Wolfson Microelectronics. Note that only the small sub-module 'Customer Mini-Board', marked WM8731L-6061-FL28-M and available separately, is required for the example configuration used in this Application Note.

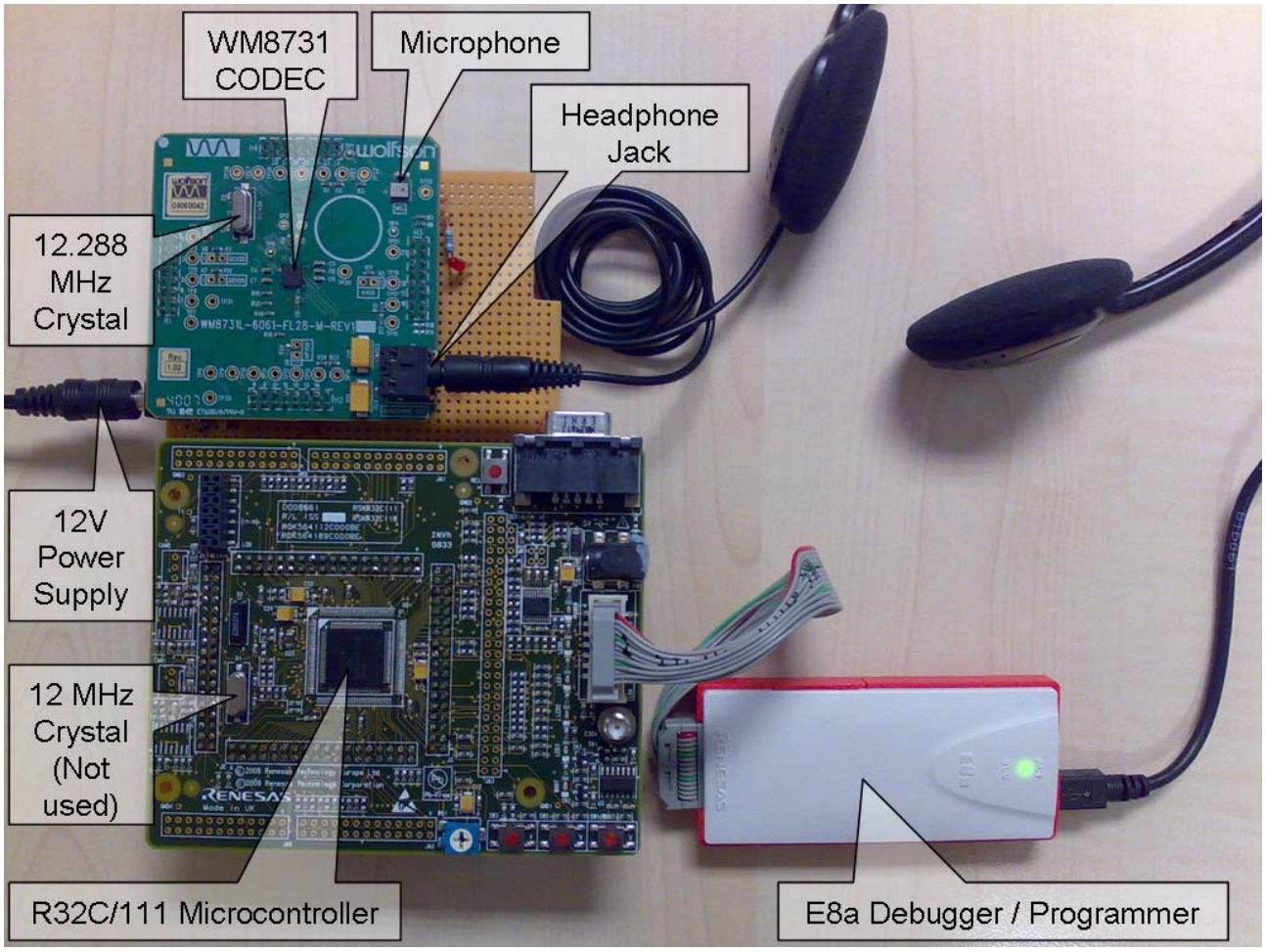


Figure 6: Photograph of Example Hardware Setup

A section of prototyping board was used to mount both boards together. Two 3V3 voltage regulators were used to provide separate digital and analogue supplies. Connections for the I²S digital audio interface and I²C interface between the RSK and the WM8731 mini-board, as well as the connections between the R32C timers and serial interface clock input were also implemented on the prototyping board (See Appendix: Interface Board Schematic).

Even small differences in the frequency or phase of the clock used for the R32C and WM8731 will cause noise to be introduced in the audio sampling, so a common clock, sourced from the WM8731 mini-board, is supplied to both CODEC and R32C/111 microcontroller (See MCU_CLK signal on the Appendix: Hardware Schematic).

The standard 12 MHz crystal on the WM8731 mini-board was changed to a 12.288 MHz to support standard sampling rates on the CODEC ADC (See WM8731 Datasheet, Table 18), and the CLKOUT output from the WM8731 was enabled. On the R32C RSK board the zero-ohm resistors R124 and R125 were removed to disconnect the 12 MHz crystal, and R120 fitted to connect the XIN pin to the corresponding header pin, CON_XIN.

Supply of the 12.288MHz crystal for the clock source is not strictly necessary. A 12 MHz clock source can be used if, for example, a sampling rate of 7.8125 kHz can be tolerated instead of the standard 8 kHz. This would also allow a standard crystal frequency to be used for the MCU CPU clock source and the rest of the peripherals. It should also be noted that either the R32C microcontroller or the CODEC can be directly connected to the reference crystal and can then provide a clock output for the other device.

5. Software Configuration

The example software is written under the Renesas High-performance Embedded Workshop (HEW) Integrated Development Environment (IDE), Version 4.06, and compiled using the Renesas NC100 C Compiler Version 1.01. The R32C/111 can be programmed with the example code through the E8a debug interface using HEW, and the code executed either stand-alone or under debugger control through the E8a.

All of the source code associated with the example configuration is contained within the HEW Workspace ‘R32C_I2S_Test’, within the project of the same name. The main source file ‘main_i2s_test.c’ contains the ‘main’ function controlling the execution flow of the example program. A flowchart showing the high-level functionality of the example code can be found in Appendix Section 8.2.

When the code is executed a press of SW1 on the RSK starts recording of audio through the microphone on the WM8731L-M mini-board. The ADC samples are stored in a buffer defined in R32C/111 internal RAM.

When the SW1 switch is pressed again, the audio is played back though the ‘Headphone Out’ connection on the WM8731L-M mini-board. Any further presses on SW1 will repeat the playback of the audio recording.

Our example application code uses 8 kHz sampling rate with 16-bit stereo sampling and a buffer size of 60k bytes. This gives an audio recording time of:

(61440 / (8000 x 4)) = 1.92 seconds

Before either recording or playback of audio, the WM8731 registers are written through the I²C interface to correctly configure the CODEC for the task. The file ‘codec.c’ contains the functions to configure the WM8731. The CODEC register addresses and the settings for all of the relevant bits in them are contained within the ‘codec.h’ header file. Parameters, such as the headphone output volume, can be adjusted by changing these definitions before building the project.

The low-level functions for driving the serial interface in I²C mode are contained within the file ‘iic_comms.c’. Note that these functions are stripped down for ease of use and understanding, and no error checking is employed as there would be in real application code.

Initialisation then takes place of UART0 for the data transmission or reception, TimerA1 for the Word Select generation, DMA0 for the data transfer and finally TimerA0 for the serial clock. As the serial clock synchronises all other activity, TimerA0 is started only when all other peripherals have been initialised and are ready.

Recording will then continue without CPU involvement until the buffer in RAM is full. During this time the program waits in a loop checking a flag ‘record_playback_complete’.

DMA transfers continue until the DMA0 Terminal Count Register decrements to 0. The DMA interrupt request is generated at this point. As we have set the DMA0 interrupt control register to assign a non-zero priority to this source (and therefore enabled the interrupt), the interrupt service routine is executed. This sets the flag ‘record_playback_complete’ which allows the code to continue execution.

When a record or playback is completed we need to stop TimerA0 and TimerA1. This ensures the CODEC digital audio interface is resynchronised when the next recording or playback takes place.

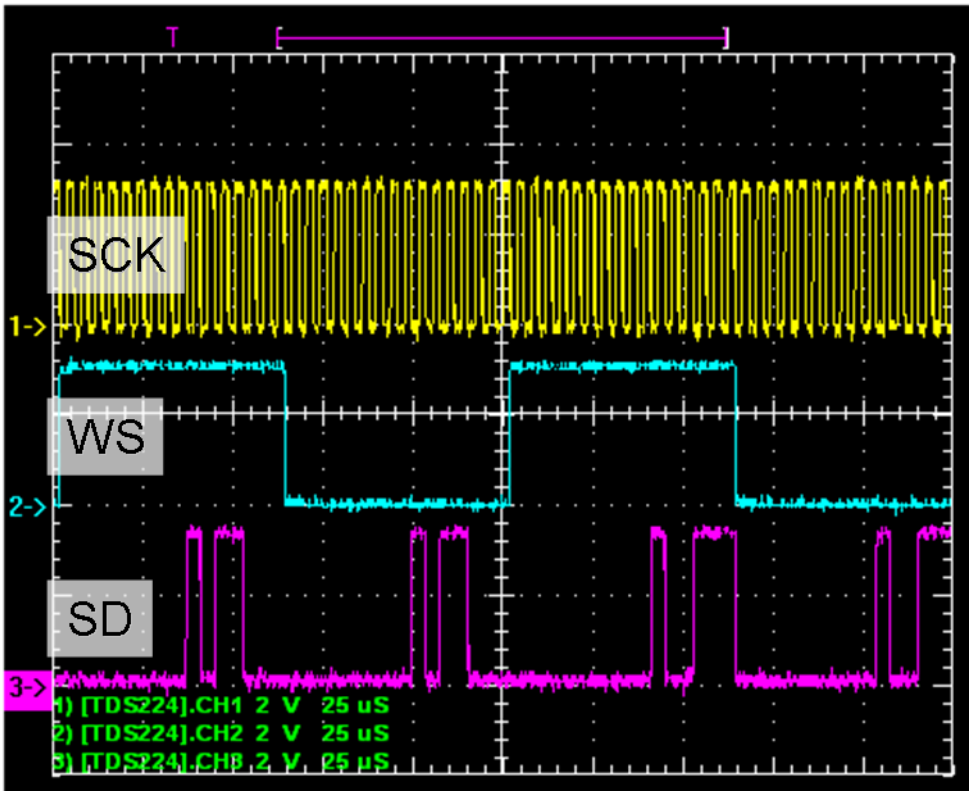


Figure 7: Oscilloscope Trace of Digital Audio Signals

Figure 7 shows an oscilloscope trace of the digital audio interface signals, with the SD trace being taken from the DACDAT signal during playback. There are four 16-bit ADC samples shown, 2 each for the Left and Right audio channels. As can be seen, each pair of samples has the same data, as each is taken from the same audio source. The SD data displayed represents, from left to right, sample data of 0x0058, 0x0058 and 0x0027, 0x0027 respectively.

6. Resources Used and Practical Implementation

There follows a summary of the resources used on the R32C/111 to implement this digital audio interface along with considerations for practical utilisation in a real application.

6.1 R32C Peripheral Resources Used

Table 1 below shows a summary of the peripheral resources on the R32C/111 microcontroller used in the implementation of the digital audio interface.

Table 1: R32C Hardware Resources Used

Peripherals	Number	Comments
Timer Channels	2	1 each for SCK and WS/LRCLK signals
Serial Ports	1	Same serial port can do transmit and receive
DMA Channels	1 or 2*	DMAC can be programmed for record or playback
I/O Pins	8**	2 Timer Out; 1 Timer In; UART Tx, Rx, CLKi; I2C SCK, SDA

*2 DMA channels are required for simultaneous transmission and reception on the digital audio interface

** This can be reduced to 7 if the TimerA1 Event Select bits are set to Select the overflow/underflow of TimerA0 (See R32C/111 Hardware Manual, Figure 16.9 'TRGSR Register')

6.2 R32C Memory Resources Used

Table 2 shows a summary of the R32C/111 microcontroller RAM and ROM resources required to implement the digital audio interface in the way described in this Application Note.

Table 2: R32C Memory Resources Used

R32C Memory Resource	Bytes	Notes
ROM	2700	Includes MCU initialisation and basic I2C routines
RAM	6	Does NOT include RAM buffer(s) for audio samples

In the example software the use of R32C/111 on-chip RAM for audio data storage is obviously both limiting in the length of the audio able to be recorded, and in the use of the limited on-chip RAM. In a real application where recording and playback of audio was required, the audio samples would likely be stored in non-volatile memory. Storing of audio samples during recording would fill each buffer alternately in a 'ping-pong' fashion. When each buffer was filled completely its contents would then be transferred to non-volatile memory, such as external serial EEPROM or Flash while the second buffer was filled.

Similarly, in an application where real-time signal processing of audio is done (e.g. noise cancellation), data is simultaneously received and sent to the audio CODEC. The RAM required for this would also be much smaller than in the example software, and a circular buffer of typically a few hundred bytes would be sufficient.

6.3 R32C Bus Bandwidth Resources Used

Table 3 below shows the bus bandwidth required to move data to and from the UART transmit and receive buffer registers during playback and record.

To calculate the number of bus cycles required to perform a DMA transfer from the UART0 Receive Buffer to a RAM buffer we can use the following equation:

CPU Cycles
= (8-bit Source SFR Read x Peripheral Clock Divider) + (8-bit Destination RAM Write) + 1
= (2 x 2) + (1) + 1
= 6 CPU cycles

If we calculate the number of CPU cycles required for a DMA transfer from RAM to UART0 Transmit Buffer, it also works out at 6 cycles, therefore the bus load when doing either receive / record or transmit / playback is the same, and when doing both is doubled.

Table 3: R32C Bus Bandwidth resources Used

Sampling Rate	Data Width	Period between DMA transfers (µs)	Bus Bandwidth
8kHz	16-bit	31.25	0.38%
8kHz	24-bit	20.83	0.57%
48kHz	16-bit	5.2	2.30%
48kHz	24-bit	3.47	3.45%
96kHz	16-bit	2.6	4.60%
96kHz	24-bit	1.74	6.90%

Note: 1. The above figures are calculated assuming a 50MHz CPU bus and 25MHz Peripheral bus
Each DMA transfer to or from the UART transmit or receive buffer registers takes 6 cycles or 120ns

It is worth noting that the bus bandwidth required to shift data to and from transmit and receive buffers using the DMAC, and the CPU bandwidth to then process this audio sample data will be broadly the same as that required for a dedicated hardware I2S peripheral.

We can see from this Table that even with the sampling rate of 8 kHz and sample word size of 16-bit used in the example software there is a very limited impact on the CPU bus bandwidth and therefore the vast majority of the CPU performance is still available for the rest of the application software.

6.4 Practical Implementation

For audio recording the figures in Table 3 do not include the CPU instructions required to manipulate the audio sample data once the RAM buffer has been filled. In a real application this might involve storing the data to non-volatile memory, signal processing and compression for receiving audio samples from the CODEC. Similarly for playback of audio, the data might need to be retrieved from off-chip non-volatile memory and decompressed before transmitting to the CODEC.

For example, a reasonable size for each ping-pong buffer might be 256 bytes, as many common Serial Flash devices have the same page programming size of 256 bytes. With the same 8 kHz sampling rate as above, 16ms would be available between buffers filling to store each buffer contents in the Serial Flash memory. A standard 16Mbit Serial Flash Memory would be large enough to hold over 2 minutes of audio without further compression

7. Conclusions

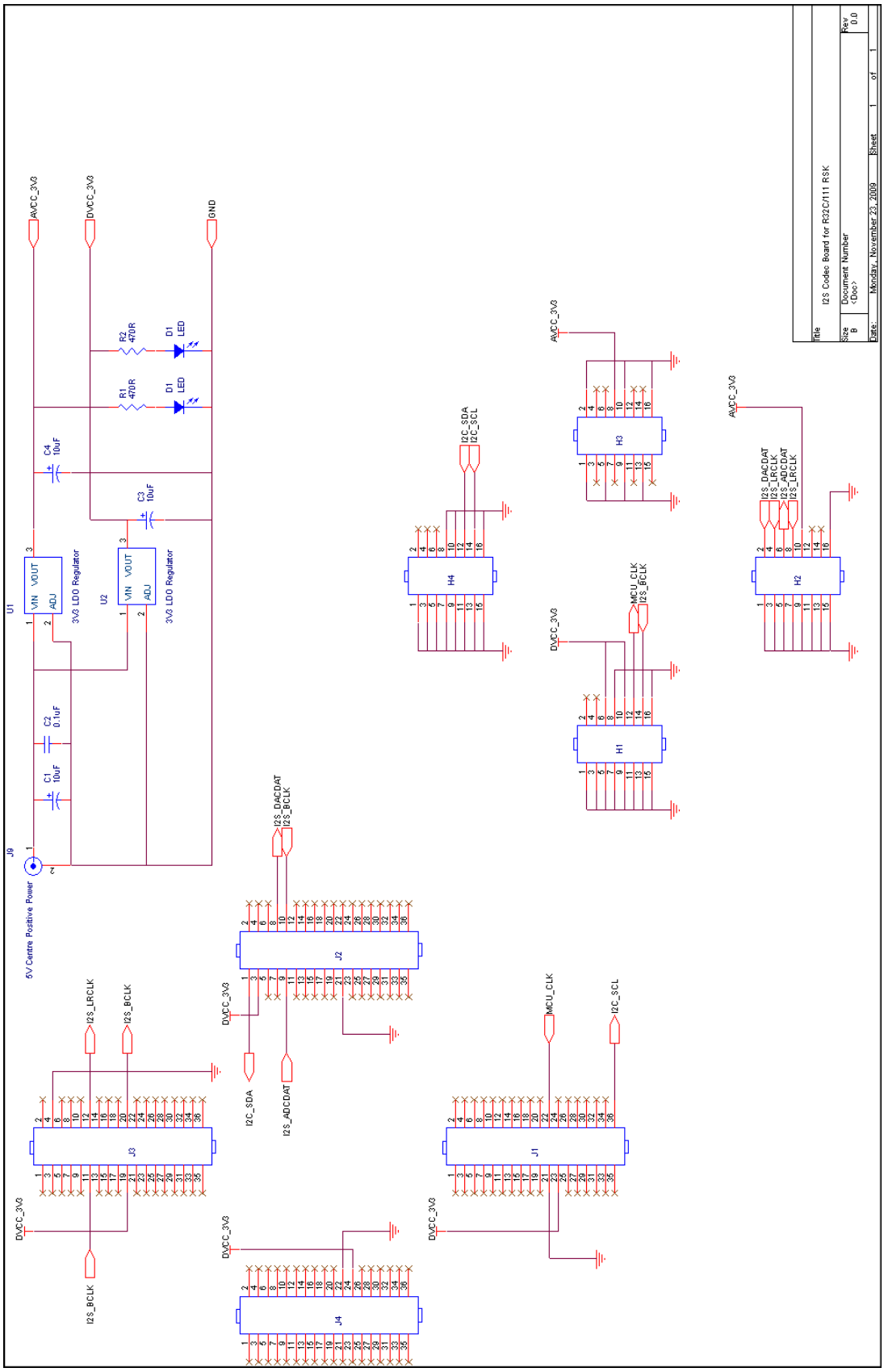
Although the Renesas R32C/111 Group are without a hardware peripheral capable of directly interfacing to a digital audio CODEC, it is fairly straightforward to utilise some of the plentiful timer and serial peripherals along with the DMA controller to simulate such a digital audio interface.

Such a simulated interface can be used for recording and playback of audio with minimal impact on CPU performance and no more overhead on the MCU buses than would usually be the case for a dedicated hardware peripheral.

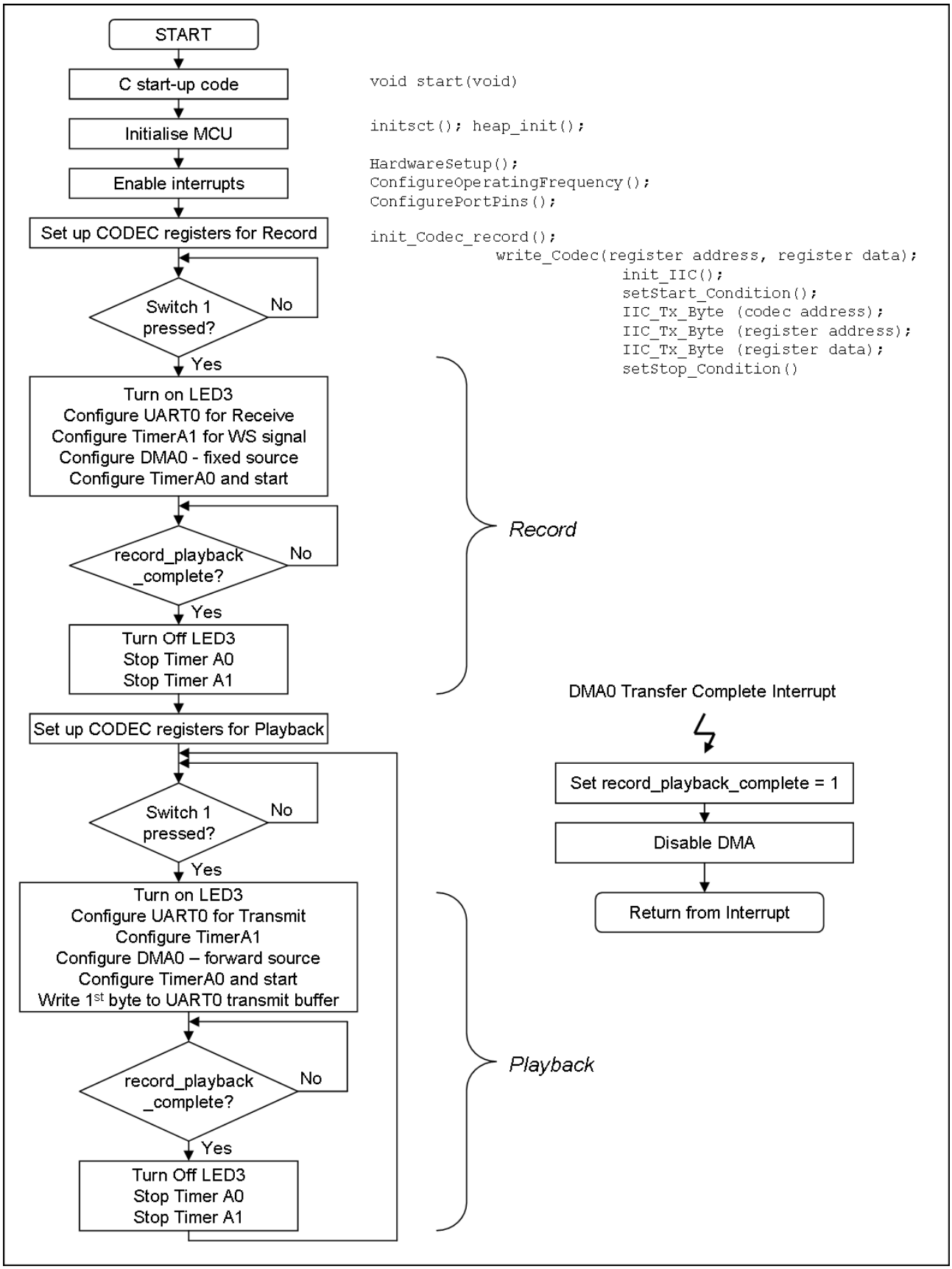
The described method therefore improves the suitability of the R32C Series for some digital audio applications and enables the cost-effective use of audio in a wide range of consumer and industrial user interface applications where this is an increasingly sought feature.

8. Appendix

8.1 Interface Board Schematic



8.2 Code Flowchart



8.3 Code Listing

```
/***********************************************************************
* DISCLAIMER:
* The software supplied by Renesas Technology Europe Ltd is
* intended and supplied for use on Renesas Technology products.
* This software is owned by Renesas Technology Europe, Ltd. Or
* Renesas Technology Corporation and is protected under applicable
* copyright laws. All rights are reserved.
*
* THIS SOFTWARE IS PROVIDED "AS IS". NO WARRANTIES, WHETHER EXPRESS,
* IMPLIED OR STATUTORY, INCLUDING BUT NOT LIMITED TO IMPLIED
* WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* APPLY TO THIS SOFTWARE. RENESAS TECHNOLOGY AMERICA, INC. AND
* AND RENESAS TECHNOLOGY CORPORATION RESERVE THE RIGHT, WITHOUT
* NOTICE, TO MAKE CHANGES TO THIS SOFTWARE. NEITHER RENESAS
* TECHNOLOGY AMERICA, INC. NOR RENESAS TECHNOLOGY CORPORATION SHALL,
* IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR
* CONSEQUENTIAL DAMAGES FOR ANY REASON WHATSOEVER ARISING OUT OF THE
* USE OR APPLICATION OF THIS SOFTWARE.
***********************************************************************/

/* Copyright (C) 2010. Renesas Technology Europe, All Rights Reserved */

/*""FILE COMMENT""****************************** Technical reference data **
* File Name    : main_i2S_test.c
* Version      : 1.00
* Device    : R32C/111 (R5F64112DFB)
* Tool Chain   : HEW, R32C Toolchain v1.01
* H/W Platform : RSKR32C111
* Description  : Demonstration of implementation of I2S interface on R32C111
*
*  Note: This file contains a compiler directive "#pragma interrupt
*  functionname" which instructs the compiler to treat the following
*  function as an interrupt. The compiler will save all registers
*  used in the function and replace the normal RTS instruction with an REIT
*  instruction at the end of the function.
*
************************************************************************
/

/***********************************************************************
*
* History      : 06.01.2010 Ver. 1.00 First Release
*""FILE COMMENT
END""*****************************************************/

/***********************************************************************
*
User Includes (Project level includes)
************************************************************************
/
/* Following header file provides a structure to access on-chip I/O
    registers. */
#include "sfr111.h"
/* Following header file provides common defines for widely used items. */
#include "rskr32c111def.h"
/* Following header file provides prototypes for the functions defined in this
```

```
  file.   */
#include "main_i2s_test.h"
#include "iic_comms.h"
#include "codec.h"

/***********************************************************************
*
Constants
***********************************************************************
/


/***********************************************************************
Global Variables
***********************************************************************
/

/* Declare buffer in RAM for audio recording */
unsigned char gI2S_Buff[BUFFER_SIZE];

char record_playback_complete = 0;

/* DMAC Count register */
unsigned long dct0;
#pragma DMAC dct0 DCT0

/* DMAC source address register */
void _far *dsa0;
#pragma DMAC dsa0 DSA0

/* DMAC destination address register */
void _far *dda0;
#pragma DMAC dda0 DDA0

/* DMAC mode register */
unsigned long dmd0;
#pragma DMAC dmd0 DMD0

/***********************************************************************
User Program Code
***********************************************************************/

/*""FUNC COMMENT""*****************************************************
* Outline      : main
* Description  : Main program.
* Argument     : none
* Return value  : none
*""FUNC COMMENT END""*************************************************/

void main(void)
{
  ENABLE_IRQ

  while(1)
  {
    /* Initialise WM8731 register settings */
    init_Codec_record();
```

```
    /* Wait for switch to be depressed to record */
    while (SW1 == 1){ };

    /* Wait for switch to be released */
    while (SW1 == 0){ };

    /* 50ms delay to eliminate noise from switch at start of recording */
    delay(500000);

    record_audio();

    /* Change Codec settings (e.g. mute microphone, speaker on) via I2C
    interface for playback of audio */
    init_Codec_playback();

    while(1)
    {
       /* Wait for SW1 for playback */
       while (SW1 ==1){};

       playback_audio();
    }
  }
}
/***********************************************************************
End of function main
***********************************************************************/

/*""FUNC COMMENT""*************************************************************
* Outline      : record_audio
* Description  : Init I2S interface and streams audio samples into buffer
* Argument     : none
* Return value  : none
*""FUNC COMMENT END""*********************************************************/

void record_audio(void)
{
  record_playback_complete = 0;

  /* Red LED indicates recording in progress */
  LED3 = LED_ON;

  /* Configure UART0 for reception of digital audio data  */
  uart0_init();

  /* Configure TimerA1 to generate WS / LRCK */
  timerA1_init();

  /* Configure DMA1 to move data from UART0 Receive Buffer
  to RAM Buffer when triggered by receive interrupt */
  dma0_init_record();

  /* Configure TimerA0 to generate SCK / BCLK */
  timerA0_init();

  while (record_playback_complete==0){};
```

```
  /* Red LED off indicates recording complete */
  LED3 = LED_OFF;

  /* Stop TimerA0 for BCLK */
  ta0s = 0;
  /* Stop TimerA1 for LRCLK */
  ta1s = 0;

}
/***********************************************************************
End of function record_audio
***********************************************************************/

/*""FUNC COMMENT""*************************************************************
* Outline      : playback_audio
* Description  : Init I2S interface and streams audio samples from buffer
* Argument     : none
* Return value  : none
*""FUNC COMMENT END""*********************************************************/

void playback_audio(void)
{
   record_playback_complete = 0;

   /* Green LED indicates playback in progress */
   LED0 = LED_ON;

   /* Configure TimerA1 to generate I2S LRCK */
   timerA1_init();

   /* Configure UART0 for transmission */
   uart0_init();

   /* Configure DMA0 to move data from RAM buffer to
   UART0 Transmit Buffer when triggered by TXEPT interrupt */
   dma0_init_playback();

   timerA0_init();

   /* Start serial comms - first byte must be initiated manually as
   DMA0 transfer triggered on transmit register empty signal*/
   u0tb = (char)gI2S_Buff[0];

   while (record_playback_complete==0){};

   /* Green LED off indicates playback complete */
   LED0 = LED_OFF;

   /* Stop TimerA0 for BCLK */
   ta0s = 0;
   /* Stop TimerA1 for LRCLK */
   ta1s = 0;
}
/***********************************************************************
End of function playback_audio
***********************************************************************/
```

```
/*""FUNC COMMENT""************************************************************
* Outline      : timerA0_init
* Description  : Configures Timer A0 in Timer Mode with TA0OUT inverted on
*        underflow to generate BCLK for I2S interface
* Argument     : none
* Return value  : none
*""FUNC COMMENT END""*************************************************************/

void timerA0_init(void)
{
  /* Timer 0 mode register,
  b1:b0      - TMOD1:TMOD0 - 00 (timer mode)
  b2         - Reserved    - 0
  b4:b3      - MR2:MR1     - 00 (TA0IN pin is programmable I/O port)
  b5         - Reserved    - 0
  b7:b6      - TCK1:TCK0   - 00 (f1 as frequency source) */
  ta0mr = 0x00;

  /* Configure pin P3_0 as TA0OUT */
  p3_0s = 0x01;

  /* Timer A0 start and reload value
  Want to generate 8kHz * 32 = 256 kHz output => 3.90625us period
  CPU clock is 4 x XIN = 4 x 12.288 MHz = 49.152 MHz
  Peripheral clock is CPU clock / 2 = 24.576 MHz => clock period 40.69ns
  Clocks per half BCLK period = ((3.90625us / 2) / 40.69ns) = 48 */
  ta0 = 47;

  /* start timer A0*/
  ta0s = 1;

}
/*****************************************************************************
End of function timerA0_init
*****************************************************************************/

/*""FUNC COMMENT""************************************************************
* Outline      : timerA1_init
* Description  : Configures Timer A1 in Event Counter Mode with BCLK (from
*        TimerA0) as count source and toggle TA1OUT on underflow to
*        generate LRCK for I2S  erface
* Argument     : none
* Return value  : none
*""FUNC COMMENT END""*************************************************************/

void timerA1_init(void)
{
  /* Timer 1 mode register,
  b1:b0      - TMOD1:TMOD0  - 01 (event counter mode)
  b2         - Reserved     - 0
  b3         - MR1       - 0 (count on falling edge)
  b4         - MR2       - 0 (Increment /decrement from UDF register)
  b5         - MR3       - 0
  b6         - TCK0      - 0 (reloading)
  b7         - TCK1      - 0               */
  ta1mr = 0x01;
```

```
  /* Configure pin P3_3 as Input */
  pd3_3 = 0;

  /* Confirm Port 3 is input to Timer A */
  ifs00 = 0;

  /* Confirm count source as TA1IN */
  ta1tgl = 0;
  ta1tgh = 0;

  /* Configure pin P3_2 as Output */
  pd3_2 = 1;

  /* Configure pin P3_2 as TA1OUT */
  p3_2s = 0x01;

  /* Timer A1 starts and reload value */
  /* Want to generate LRCK with toggle every 16 BCLKs */
  ta1 = 15;

  /* start timer A1 */
  ta1s = 1;

}
/***********************************************************************
End of function timerA1_init
***********************************************************************/

/*""FUNC COMMENT""*************************************************************
* Outline      : uart0_init
* Description  : Configures UART0
* Argument     : none
* Return value  : none
*""FUNC COMMENT END""*********************************************************/

void uart0_init(void)
{
  /* dummy variable for UART Continuous Receive Mode enable */
  volatile char dummy;

  /* Port pin configuration */

  /* Configure the port pin p6_1 (CLK0) as input.   */
  pd6_1 = 0;
  p6_1s = 0x03;

  /* Configure the port pin p6_3 (TXD0) as output.  */
  pd6_3 = 1;
  p6_3s = 0x03;

  /* Configure the port pin p6_2 (RXD0) as input.   */
  pd6_2 = 0;

  /* UART0 transmit/receive mode register
    b2:b0 - SMD12:SMD1 - 001 (Synchronous Serial Mode selected)
         b3   - CKDIR      - 1 (External clock selected)
```

```
        b4  - STPS    - 0 (1 Stop bit)
        b5  - PRY     - 0 (No parity used)
        b6  - PRYE    - 0 (No parity used)
     b7  - IOPOL      - 0 (TXD, RXD polarity not inverted)  */

  u0mr = 0x09;

  /* UART0 transmit/receive control register 0
    b1:b0 - CLK01:CLK0 - 00 (f1 clock)
        b2  - CRS     - 0 (CTS/RTS function select bit (Set to 0))
     b3  - TXEPT      - 0 (Clear transmit register empty flag)
        b4  - CRD     - 1 (Disable CTS/RTS function)
        b5  - NCH     - 0 (CMOS Data output selected)
        b6  - CKPOL      - 0 (Transmit data at falling edge of clock)
     b7  - UFORM      - 1 (MSB first selected) */

   u0c0 = 0x90;

  /* UART0 transmit/receive control register 1
  b0      - TE     - 1 (Enable transmission)
  b1      - TI     - 0 (Clear the 'transmit buffer empty' flag)
  b2      - RE     - 1 (Enable reception)
  b3      - RI     - 0 (Clear the 'Receive complete' flag)
  b4      - U0IRS     - 1 (Interrupt on transmission complete (TXEPT=1))
  b5      - U0RRM     - 1 (Enable Continuous Receive Mode)
  b6      - UOLCH     - 0 (non-inverse data logic selected)
  b7      - UOERE     - 0 (Error signal output disable) */

   u0c1 = 0x35;

  /* Unused in synchronous serial mode. Set to 0.   */
  u0smr = 0x00;

  /* Unused in synchronous serial mode. Set to 0.   */
  u0smr2 = 0x00;

  /* UART0 Special Mode Register 3
     Bit b3 - NODC - Set to 0 in order to select 'CLKi is CMOS output'
     Remaining bits not applicable in synchronous serial mode. Set to 0. */
  u0smr3 = 0x00;

  /* Unused in synchronous serial mode. Set to 0.   */
  u0smr4 = 0x00;

  /* Read dummy value from UART receive buffer to allow reception in
  Continuous   Receive Mode. Reception of further bytes enabled by DMA
  tranfers from u0rb */
  /* This has no effect on UART transmissions */
  dummy = u0rb;

}
/***********************************************************************
End of function uart0_init
***********************************************************************/

/*""FUNC COMMENT""*************************************************
* Outline      : dma0_init_record
```

```
* Description  : Configures the DMAC channel 0 in fixed source and
*          forward destination mode for audio recording
* Argument  : none
* Return value  : none
*""FUNC COMMENT END""***************************************************/

void dma0_init_record(void)
{
  /* DMA0 Mode Register
  b1:b0   - MD01:MD00    - 01 (Single transfer)
  b3:b2   - BW01:BW00    - 00 (8 bit mode)
  b4      - USA0      - 0  (Fixed Source)
  b5      - UDA0      - 1  (Forward Destination)
  b7:b6   - Reserved     - 00 */
  dmd0 = 0x21;

  /* Set DMA transfer count */
  dct0 = BUFFER_SIZE;

  /* Set DMA Source Address as UART0 receive buffer */
  dsa0 = (unsigned long *)&u0rb;

  /* Set DMA Destination Address */
  dda0 = (unsigned long *)&(gI2S_Buff);

  /* DMA1 Request Source Select UART0 receive interrupt request */
  dm0sl = 0x0F;

  /* Enable and set priority of DMA0 complete interrupt */
  dm0ic = 0x06;

}
/***********************************************************************
End of function dma0_init_record
***********************************************************************/

/*""FUNC COMMENT""*********************************************************
* Outline      : dma0_init_playback
* Description  : Configures the DMAC channel 0 in forward source and
*          fixed destination mode for audio playback
* Argument  : none
* Return value  : none
*""FUNC COMMENT END""*****************************************************/

void dma0_init_playback(void)
{
  /* DMA0 Mode Register
  b1:b0   - MD01:MD00    - 01 (Single transfer)
  b3:b2   - BW01:BW00    - 00 (8 bit mode)
  b4      - USA0      - 1  (Forward Source)
  b5      - UDA0      - 0  (Fixed Destination)
  b7:b6   - Reserved     - 00 */
  dmd0 = 0x11;

  /* Set DMA transfer count */
  dct0 = BUFFER_SIZE - 1;
```

```
  /* Set DMA Source Address as data to transmit over I2S. First byte
  transmitted is dummy byte, so point to next byte to maintain alignment*/
  dsa0 = (unsigned long *)&(gI2S_Buff+1);

  /* Set DMA Destination Address */
  dda0 = (unsigned long *)&u0tb;

  /* DMA0 Request Source Select UART0 transmit interrupt request */
  dm0sl = 0x0E;

  /* Enable and set priority of DMA0 complete interrupt */
  dm0ic = 0x06;
}
/***********************************************************************
End of function dma0_init_playback
***********************************************************************/

/*""FUNC COMMENT""*************************************************
* Outline      : _dma0_complete
* Description  : Interrupt handler for DMA0 complete.
* Argument  : none
* Return value  : none
*""FUNC COMMENT END""*********************************************/

#pragma interrupt _dma0_complete(vect=8)
void _dma0_complete(void)
{
  record_playback_complete = 1;

  /* Disable DMA0 */
  dm0sl = 0x00;
}
/***********************************************************************
End of ISR _dma0_complete
***********************************************************************/
```

Website and Support

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

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

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.