



Technical notes on using Analog Devices DSPs, processors and development tools

Visit our Web resources <http://www.analog.com/ee-notes> and <http://www.analog.com/processors> or e-mail [processor.support@analog.com](mailto:processor.support@analog.com) or [processor.tools.support@analog.com](mailto:processor.tools.support@analog.com) for technical support.

## Programming S/PDIF on ADSP-2136x and ADSP-21371 SHARC® Processors

*Contributed by Aseem Vasudev Prabhugaonkar*

*Rev 2 – August 21, 2007*

### Introduction

This EE-Note provides a basic description of the Sony Philips Digital Interface (S/PDIF), its frame structure, and how to program S/PDIF on SHARC® ADSP-2136x and ADSP-21371 processors to transmit and receive data. This EE-Note also explains the Precision Clock Generator (PCG) setup required to provide necessary signals for an S/PDIF transmit section. Code examples are provided with this EE-Note to demonstrate programming of the S/PDIF transmit and receive sections of SHARC ADSP-2136x and ADSP-21371 processors. The examples consist of code that demonstrates talk-through and data loopback performed on S/PDIF interface.

### Basics of S/PDIF

S/PDIF is a standard that defines a serial interface for transferring digital audio data between various audio equipment like CD players, DVD players, and amplifiers. S/PDIF describes a serial, uni-directional, self-clocking interface for the interconnection of digital audio equipment for consumer and professional applications that use linear PCM coded audio samples. The data can be transferred between devices without having to convert it to an analog signal. This is the biggest advantage of S/PDIF. For instance, when audio is transferred from a CD player to an audio amplifier over an analog link, noise is introduced; filtering out this noise is not easy. This problem is overcome when audio data is transferred over a digital link instead of an analog link.

S/PDIF is used to transmit digital signals of many formats, the most common being the 48 kHz sample rate format used in digital audio tape (DAT) and the 44.1 kHz format used in CD audio. To support both systems, as well as others that might be needed, the format has no defined data rate. Instead, the data is sent using a bi-phase mark code, which has one or two transitions for every bit. This allows the original word clock to be extracted from the signal itself. S/PDIF is also polarity independent, which makes it a lot easier to work with.

S/PDIF was developed from the AES/EBU standard used in the professional audio field, which is commonly used in DAT systems. S/PDIF remained identical at the protocol level, but changed the physical connectors from XLR to electrical RCA jacks or optical TOSLINK, both of which cost less and are easier to use. Essentially, S/PDIF is a consumer version of the AES/EBU format. The S/PDIF interface specification consists mainly of the hardware and software. The software usually deals with S/PDIF frame format, and the hardware interface is the actual physical connection medium used to transfer data between

two devices. The various interfaces used for the physical medium could be TTL (Transistor-Transistor Logic), COAX (coaxial cable, 75-ohm cable connected with RCA plugs), and TOSLINK (an optical fiber connection). The S/PDIF signal is independent of polarity; hence, you do not have to worry about the absolute signal polarity.

## S/PDIF Bi-Phase Encoding

S/PDIF is a single-wire serial interface, and the bit clock is embedded within the S/PDIF data stream. The bit clock and the frame sync are recovered from the data stream at the receiver end. The simple coding of digital data (i.e., 1 as logic high, and 0 as logic low) is not an ideal format, because another signal (a bit clock) is required to transmit and sample data bits in the stream. This is where the bi-phase encoding comes in; the bit clock is embedded within the transmitted stream and recovered at the receiver. Bi-phase coded PCM has a mean voltage of zero, which eliminates DC on the interface, so the resulting data can be AC-coupled through a transformer or series capacitor. Each data bit in the S/PDIF stream has a time slot that begins with a transition and ends with a transition. The second transition serves as the beginning transition for the next time slot. If the data bit is a 1, an additional transition is made in the middle of the time slot; a data bit of 0 has no additional transition. Figure 1 illustrates bi-phase encoding. Unit interval (UI) is the shortest interval between transitions.

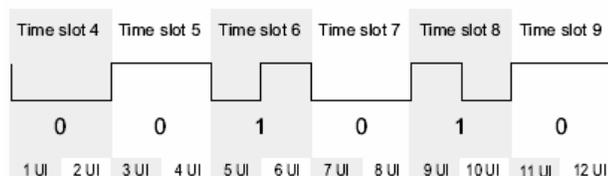


Figure 1. Bi-Phase encoding in S/PDIF stream

As noted in Figure 1, even if the data is a stream of 0s, there will be transitions at each time slot, and the direction of these transitions (signal polarity) becomes irrelevant.

## S/PDIF Frame Format

Each S/PDIF frame is 64 timeslots (128 UIs). Each frame consists of two sub-frames, which are 32 timeslots (64 UIs). The sub-frame comprises a preamble, 24 bits of audio data, and 4 bits that carry other information such as user data and channel status information. Figure 2 shows S/PDIF sub-frame format.

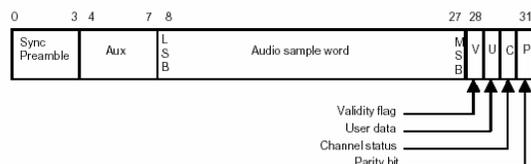


Figure 2. S/PDIF sub-frame format

The LSB of the audio data is driven first. The first four time slots of a sub-frame carry a distinctive data pattern called a preamble, which is used to indicate sub-frame and block starts. There are three preambles, each of which breaks the bi-phase coding rule by containing one or two pulses, which have duration of

3 UIs. Breaking this rule would mean that the pattern cannot occur anywhere else in the stream. Figure 3 illustrates S/PDIF frame and block format.

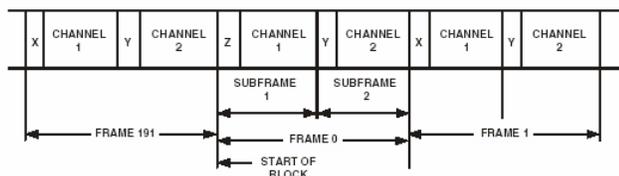


Figure 3. S/PDIF frame and block format

Each sub-frame begins with a 4-bit preamble. Preamble z indicates the start of a block and the start of sub-frame channel A. Preamble x indicates the start of a channel A sub-frame when not at the start of a block. Preamble y indicates the start of a channel B sub-frame.

## S/PDIF Transmitter and Receiver of ADSP-2136x and ADSP-21371 SHARC Processors

The S/PDIF transmitter in SHARC processors usually receives data from any serial input port, such as SPORTs, SRC, IDP, or other serial inputs. The serial data input to the transmitter can be formatted as left-justified, I<sup>2</sup>S, or right-justified with word widths of 16, 18, 20, or 24 bits. The serial data, clock, and frame sync inputs to the S/PDIF transmitter are routed through the Signal Routing Unit (SRU). These inputs, which are controlled by the SRU control registers, come from a variety of sources such as SPORTs, external pins, the precision clock generator (PCG), or the sample rate converter (SRC). The S/PDIF transmitter output may be routed to an output pin via the SRU and tied directly to the physical layer interface of an end-user product. The output is also available to the S/PDIF receiver for loopback testing through the SRU.

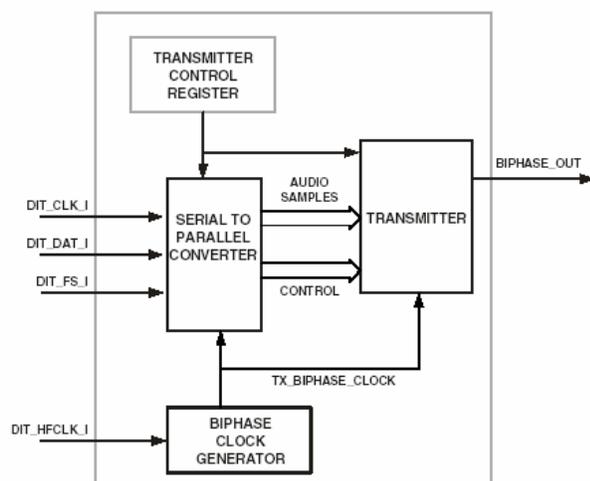


Figure 4. ADSP-2136x and ADSP-21371 processors S/PDIF transmitter block diagram

The S/PDIF receiver complies with all common serial digital audio interface standards, including IEC-60958, IEC-61937, AES3, and AES11. These protocols encompass everything from simple stereo streams

to high sample rate, and compressed surround sound, such as those defined by Dolby or DTS. The input to the receiver is a bi-phase encoded signal that may contain any number of audio channels (compressed or linear PCM) or non-audio data. The receiver decodes the single bi-phase encoded signal to produce serial data, `LRCLK`, and high-frequency serial clocks. The receiver derives the clock from the on-chip digital phased-locked loop (PLL). In applications that are not sensitive to moderate jitter, the on-chip PLL can provide the clock source derived from the bi-phase encoded signal.

## S/PDIF Receiver Status Control Bits

It is important to understand the significance of some of the status control bits in `DIRSTAT` register and how to use them.

`DIR_NOSTREAM` – Asserted when there is no incoming data stream, such as when the cable is unplugged or the transmitter is not sending data. If asserted when the input is linear PCM, the receiver performs a soft mute (slowly and linearly decrementing the last valid audio sample to zero over a period of 4096 frames). During this time, the PLL three-states the charge pumps until the soft mute has been completed. If asserted for non-linear PCM audio data, the receiver sends out zeros after the last valid sample. This bit can generate interrupts.

`DIR_NOAUDIO` – Asserted (bit 1=1) when input data is non-PCM. Each sub-frame in a data frame is preceded by a preamble (96-bit-wide sync code), and the `DIR_NOAUDIO` bit is set once the complete preamble is detected. Since this bit can generate interrupts, it is used in to count the number of received frames, which, in turn, helps to detect frame sizes.

`DIR_LOCK` – This bit is set when the S/PDIF PLL is locked. The PLL can get unlocked when there is no input stream (the `DIR_NOSTREAM` bit is asserted) and de-asserts the `DIR_LOCK` bit. When the PLL is unlocked, audio data is handled according to bit setting of the lock error control bits (`DIR_LOCK`) in the `DIRCTL` register. This also can generate interrupts.

## S/PDIF Receiver PLL Lock

As mentioned earlier, the S/PDIF receiver contains an on-chip PLL. It is important that the PLL is locked in order to obtain correct data from the S/PDIF Rx. Once the receiver is locked, the `DIR_LOCK` bit in the `DIRSTAT` register is set, which can be polled to detect a LOCK condition. Another option is to use the `SPDIF_RX_LOCK_START` interrupt in the `DAI_IRPTL_H/L` register. These trigger the `DAI_INTH/L` interrupt once the receiver is locked. From this point on, the extracted output from the S/PDIF receiver becomes valid.

## S/PDIF Receiver DTS Frame Sizes

The SPDIF receiver supports the DTS stream. The DTS specifications support frame sizes of 256, 512, 1024, 2048, and 4096. But the on-chip SPDIF receiver supports only 256, 512, and 1024 DTS frames sizes. According to DTS stream format, a sync code is present at the beginning of each block that may contain 256, 512, 1024, 2048, or 4096 frames.



Checks for sync code after 1024 frames instead of 4096 frames

Figure 5. Block structure for 4096 frames

Since the SPDIF receiver hardware supports frame sizes up to 1024 only, it checks for the sync code after every 1024 frames while the sync code is actually present after every 2048 or 4096 frames, depending on the frame size of the received data. When a sync code is not found, the `DIR_NOAUDIO` bit gets de-asserted which can cause non-PCM data to be mistaken for a PCM audio data. So for DTS test kit frames with 2048 and 4096 frame sizes can be detected by adding the sync-detection logic in software by using a software counter to check for the DTS header every 2048 and 4096 frames, respectively. In this approach, the frame sync of the DIR receiver is used to count the number of frames. The steps to implement the same are as follows.

1. Initially enable the `DIR_NOAUDIO` interrupt for rising edge.
2. Route the frame sync output of the DIR receiver to another DAI interrupt. If another DAI interrupt is not available, the same interrupt can be switched for both functionalities.
3. Inside the `DIR_NOAUDIO` interrupt service routine, enable the DAI interrupt corresponding to the frame sync.
4. Increment a count inside the DAI frame sync interrupt service routine for every data frame.
5. When the count reaches 2048 or 4096 (depending on the frame size used), read the `DIR_NOAUDIO` bit.
6. If the `DIR_NOAUDIO` bit is high, the bit stream is OK. Now clear the count value and restart the counting. This will be repeated every 2048 or 4096 frames.
7. If the `DIR_NOAUDIO` bit is low, the bit stream does not have the sync code. For this case, disable the frame sync DAI interrupt and take the corrective action such as muting. Then re-enable the `DIR_NOAUDIO` interrupt for the rising edge. This will restart the counting process with respect to the next `DIR_NOAUDIO` interrupt.

The above approach can be implemented for any number of frame sizes.

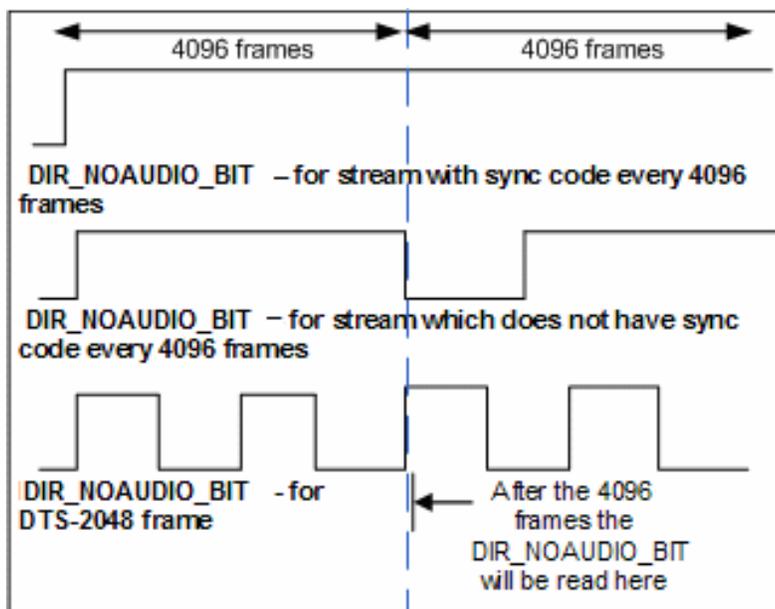


Figure 6. DIR\_NOAUDIO\_BIT functioning for different cases

For the first case in Figure 6, the DIR\_NOAUDIO bit will be tested for every 4096 frames and the software counter will be restarted.

For the second case in Figure 6, the DIR\_NOAUDIO bit is not available after 4096 frames; hence, the correction action should be taken and the DIR\_NOAUDIO interrupt should be re-enabled.

For the third case in Figure 6, the DIR\_NOAUDIO bit will be high for every 4096 frames and the software counter can be restarted. It will not be detected as an error condition.

## Interrupted Data Stream at the S/PDIF Receiver

When the SPDIF receiver data streams are interrupted, (e.g., when a cable is unplugged and reconnected), it is necessary to take extra steps to ensure that the SPDIF receiver's digital PLL will re-lock to the stream. The steps to accomplish this are described below.

1. Set up interrupts within the DAI so that the SPDIF RX can generate an interrupt when the stream is reconnected.
2. Within the interrupt service routine (ISR), stop and restart the digital PLL. This is accomplished by setting and then clearing bit 7 of the SPDIF receiver control register.
3. Return from the ISR and continue normal operation.

This method of resetting the digital PLL has been shown to provide extremely reliable performance when SPDIF inputs that are interrupted or unplugged occur momentarily. Listing 1 explains the procedure.

```

1. Initialize the No Stream Interrupt
/* Enable interrupts (globally) */
  BIT SET MODE1 IRPTEN;

/* unmask DAI Hi=Priority Interrupt */
  bit set imask DAIHI;
  ustat1 = DIR_NOSTREAM_INT;

/* Enable no-stream Interrupt on Falling Edge. Interrupt occurs when the stream is
reconnected */
  dm(DAI_IRPTL_FE) = ustat1;

/* Enable Hi-priority DAI interrupt */
  dm(DAI_IRPTL_PRI) = ustat1;

/* This ISR is triggered when the DIR sets no_stream bit */
  _DAIisrH:

2. Reset the Digital PLL inside ISR
/* Reading DAI_IRPTL_H clears interrupt */
  r8=dm(DAI_IRPTL_H);
  ustat2=dm(DIRCTL);

/* bit_7 disables D pll only */
  bit set ustat2 DIR_PLLDIS;
  dm(DIRCTL)=ustat2;

/*re-enable the digital pll */
  bit clr ustat2 DIR_PLLDIS;
  dm(DIRCTL)=ustat2;

```

*Listing 1. PLL re-lock code implementation*

## Interfacing S/PDIF Transmitter to SRCs



Consider the following while interfacing the SRC to the S/PDIF. The maximum data that is transferred through the SRCs is 24 bits. When a 32-bit clock and data is provided to the 128 dB SRC, the eight least significant bits (LSBs) are zero. However for a 140 dB SRC, the eight LSBs bits contain the ratio information. So, if the eight LSBs are not zero, the SPDIF transmitter channel status bytes will be affected.

A software workaround has been proposed to handle this situation. For 32-bit data, the 24 most significant bits (MSBs) of data of the SRC output are connected to the DIT pin. But for the eight LSBs, the DIT pin is connected low. This is implemented by connecting the frame sync of the SRC output to the DAI interrupt. The DAI interrupt is enabled for both edges.

Once the DAI interrupt is fired on a particular edge, within the DAI ISR,

- The SRC output is connected to the DIT pin.
- A timer with a count equal to 24 SCLK cycles is started.

Once the MSB 24 bits have been transmitted, the Timer interrupt is fired. Within the Timer ISR,

- The DIT input is connected to low.
- The timer is stopped.

Once the eight LSBs are transmitted and the FS changes edge, the DAI interrupt is fired again before transmitting the next frame.

## Consideration While Using the IDP with SPDIF

The Input Data Port (IDP) is another efficient method to transfer data to/from internal memory other than SPORTs. It is critical as to when the IDP is enabled; otherwise, it can lead to wrong data latch and bit shift.

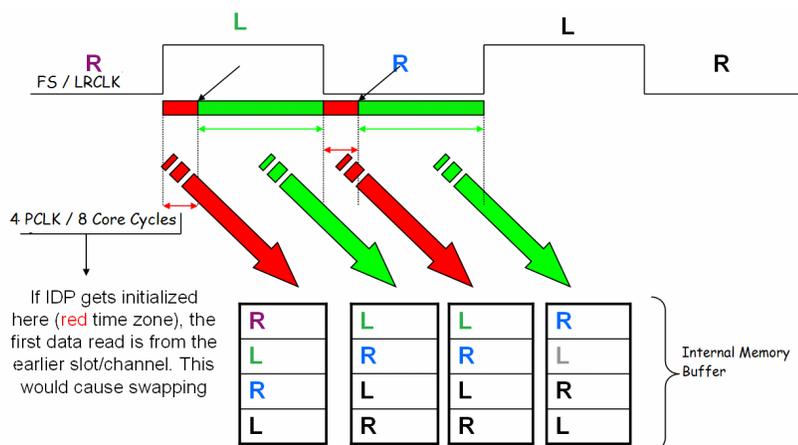


Figure 7: Considerations for IDP initialization

Data keeps moving through the IDP input shift register continuously, regardless whether the IDP is enabled. At the edge of the FS/LRCLK signal, the IDP checks for an enable bit. It takes about 4 PCLK (8 core) cycles for the IDP to actually test for IDP enable. If found enabled, it loads the data of the previous slot to internal memory (as shown in Figure 7). If the “L” or “R” channel slot should go to the first location of the internal memory buffer, enable the IDP during the green time zone. If the IDP is enabled during the red time zone, channel earlier to the one where it is enabled is written to the first location of the memory buffer. Therefore, if the intent is to start receiving data from the channel slot where the IDP is enabled, it should be enabled in the green zone within that channel slot.

It is difficult to predict when the IDP is enabled relative to the frame sync edge. The IDP may get enabled during the red time zone or the green time zone – this is difficult to predict. This can cause channel misalignment when the data is moved to internal memory.

### Solution 1

Using the DAI interrupt to sense the FS/LRCLK edge ensures the time from the edge of FS/LRCLK where IDP can be enabled. This can be mapped to a DAI high-priority interrupt so that it gets serviced immediately. With this approach, you can ensure that the IDP always gets enabled in the same time zone. Channels are never misaligned in internal memory.

## Solution 2

Software application code can also read the channel status information of IDP buffers to find out whether the IDP is working properly. If the channel status information shows L->R->L->R..., the IDP is working properly. A channel status of R->L->R->L... means that there is a shift in the words received and the IDP must be re-enabled.

## Example: Loopback from Transmitter to Receiver

This section discusses programming the ADSP-2136x and ADSP-21371 SHARC processors' S/PDIF transmitter to receive data, bit clock, and frame sync from a SPORT to generate a bi-phase encoded data stream. The Precision Clock Generators (PCG) are used to provide an over-sampling clock, frame sync, and bit clock to the SPORT and S/PDIF TX. The DIT signal is the output of the S/PDIF transmitter, and the bi-phase encoded data stream is sent over this signal. The DIT pin can be routed to any external DAI pins. The S/PDIF receiver receives bi-phase encoded data on DIR, and it is an input signal to the S/PDIF receiver.

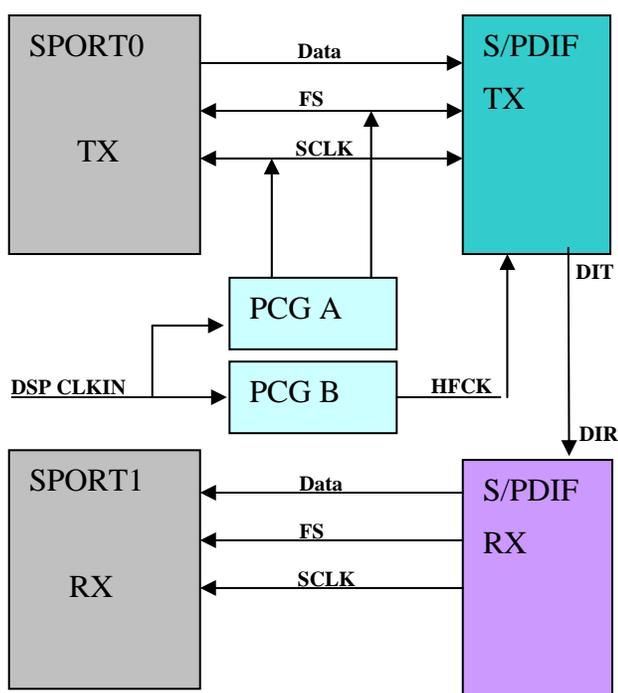


Figure 8. S/PDIF TX – RX loopback setup

The DIR pin can be routed to any of DAI pins or receive data from the DIT signal for internal loopback. SPORT0 is configured to transmit data in I<sup>2</sup>S format with a data word length of 32 bits. SPORT0 and S/PDIF TX receive the bit clock and frame sync from PCG A. PCG B supplies an over-sampling HFCLK clock signal to S/PDIF TX. The transmitter is configured for an over-sampling ratio of 256 x FS. Hence, for a specific frame sync (FS) frequency, HFCLK should be 256 x FS, and the bit-clock should be 64 x FS. The over-sampling ratio in S/PDIF transmitter is determined by the DIT\_FREQ bits of the DITCTL register. The DIT\_SMODE\_IN bits, which determine the serial data input format, are configured for I<sup>2</sup>S format. Refer to your processor's *Hardware Reference* <sup>[1][2]</sup> for details on the DITCTL register bits.

PCG A provides bit clock and frame sync signals to SPORT0 and S/PDIF TX. To comply with I<sup>2</sup>S format, the PCG\_CTLA1 register should also be programmed appropriately with the FSAPHASE\_HI bits. This value should align the rising edge of FS with the falling edge of SCLK from PCG A for I<sup>2</sup>S mode. For programming details, control register bit settings, and so on, refer to main.c in the associated .ZIP file. This file also has information on PCG, SPORT, and SRU configuration. Note that some of these signals are also brought out on DAI pins to capture them on an oscilloscope. The screen captures were taken on an oscilloscope during S/PDIF transmission.



Figure 9. SPORT0 and S/PDIF TX signals



Figure 10. SCLK and FS relationship



Figure 11. FS, SCLK, DIT and HFCLK of S/PDIF TX

The screenshot displays the VisualDSP++ development environment for the ADSP-21364 EZ KIT. The main window shows the C source code for `main.c`, which includes initialization for SPDIF RX and TX, and SPORT0 and SPORT1. The code is as follows:

```

SubFrmCnt = 0;
//-----
PCG_A_B_OnDAI();//---Connect the PCG signals to DAI pins to watch on
InitSRU();
SPDIF_RxToSport1Sru();
PCG_A_B_Config();

interrupts(SIG_P6_Sport0TxIsr);//set up sport0 interrupt
interrupts(SIG_P3_Sport1RxIsr);//set up sport1 interrupt

SPDIF_RX_Init();//Initialize SPDIF RX
SPDIF_TX_Init();//Initialize SPDIF TX

Sport0Init();//Initialize the SPORT0 for I2S mode
Sport1Init();//Initialize the SPORT1 for I2S mode

//Wait for SPORT interrupts
while(1)
{
}

//Initialize SPORT0
void Sport0Init()
{
    *pSPCTL0 |= SPEN_A | SLEN32 | OPMODE | SPTRAN; //SPORT0 config
                                                    //External cloc
                                                    //I2S mode
                                                    //Transmitter
                                                    //Word length 3
}

//Initialize SPORT1
void Sport1Init()
{
    *pSPCTL1 |= SPEN_A | SLEN32 | OPMODE; //SPORT1 config
                                                    //External cloc
                                                    //I2S mode
                                                    //Receiver
                                                    //Word length 3
}

//Initialize SPDIF Transmitter
void SPDIF_TX_Init()
{
    *pDITCTL = DIT_EN | DIT_SMODEIN0;//configuration of SPDIF TX
}
    
```

The left sidebar shows the project structure for "SPDIF TX - Rx Loopback". The "SPDIF Receive" window displays register values for DIR and DIRCHANL. The "Data(M) Memory [Hexadecimal]" window shows the SPDIF\_Rx\_Buffer memory dump, which contains a series of 0x12345600 values. The "Disassembly" window shows the assembly code for the `main` function, including instructions for setting up registers and jumping to initialization routines.

Figure 12. Viewing received data and S/PDIF RX register values in VisualDSP++® development tools

## Example: Talk-Through Code

The following code example demonstrates the receipt of data on S/PDIF RX from an external source such as a DVD player. The bi-phase encoded PCM stream is received by S/PDIF RX. SPORT1 is used to receive data from S/PDIF RX. SPORT0 is used to transmit this data to S/PDIF TX. S/PDIF TX encodes this data into a bi-phase stream, which is provided to AVR.

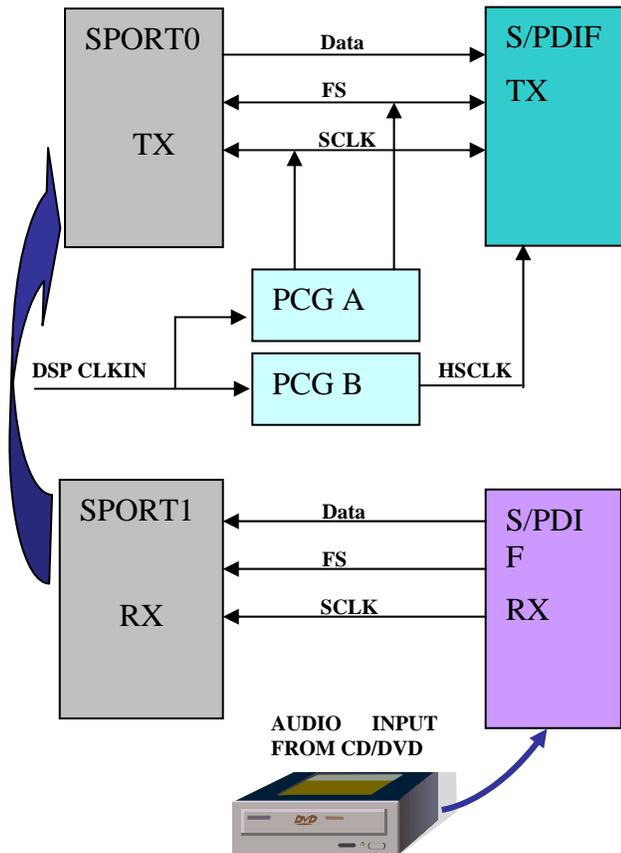


Figure 13. S/PDIF talk-through setup

For programming details, control register bit settings, and so on, refer to `main.c` in the associated .ZIP file. This file also has information on PCG, SPORT, and SRU configuration. Note that some of these signals are also brought out on DAI pins to capture them on an oscilloscope.

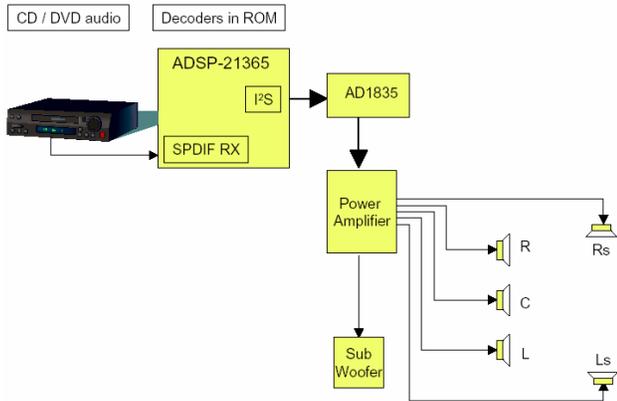


Figure 14. Example system configuration using S/PDIF

The block diagram in Figure 14 shows an example system configuration. The SHARC processor receives the S/PDIF stream over its receiver, implements decode or post-decode algorithms on the received data, and then sends the decoded audio data over serial ports in I<sup>2</sup>S or TDM mode to the codec.

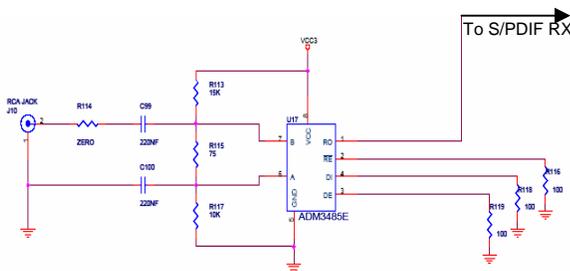


Figure 15. Suggested S/PDIF RX input circuitry

The circuit diagram in Figure 15 shows a suggested receive circuitry in which an ADM3485E (RS-485 transceiver) is used.

## References

- [1] *ADSP-2136x SHARC® Processor Hardware Reference*. Rev 1.0, October 2005. Analog Devices, Inc.
- [2] *ADSP-21368 SHARC® Processor Hardware Reference*. Rev 1.0, September 2006. Analog Devices, Inc.
- [3] *ADSP-21364 Processor EZ-KIT Lite Evaluation System Manual*. Rev 3.2, July 2007. Analog Devices, Inc.
- [4] *ADSP-21369 Processor EZ-KIT Lite Evaluation System Manual*. Rev 2.1, August 2006. Analog Devices, Inc.
- [5] *TN-26 the AES3 and IEC60958 Digital Interface TECHNOTE*. Julian Dunn, Audio Precision.
- [6] *International Standard, IEC, 60958-1*. First edition 1999-12.

## Document History

Revision	Description
<i>Rev 2 – August 21, 2007 by Deepa Venkataraman</i>	Made the EE-note generic for the ADSP-2136x and ADSP-21371 SHARC processors, and updated title accordingly.
<i>Rev 1 – April 1, 2005 by Aseem Vasudev Prabhugaonkar</i>	Initial version.