ANALOG DEVICES **Engineer To Engineer Note** **EE-140**

Technical Notes on using Analog Devices' DSP components and development tools
Phone: (800) ANALOG-D, FAX: (781) 461-3010, EMAIL: dsp.support@analog.com, FTP: ftp.analog.com, WEB: www.analog.com/dsp

Using the ADSP-21160 Serial Ports in Multi-channel Mode.

Last Modified: August 2001

Introduction

The purpose of this document is to aid in and provide an example of using the ADSP-21160 serial ports in multichannel mode. The examples given were tested on two ADDS-21160M-EZLITEs. The code was developed using VisualDSP++1.0®. Two code examples are provided. The first is a simple example of how to use multichannel mode. The second example adds the Receive Register Comparison functionality.

Hardware Setup

The hardware connection of the two serial ports is described below:

DSP1		DSP2
DTx	←=======→	RTx
DRx	←=======→	DTx
RFSx	←=======→	RFSx
TFSx	unconnected	TFSx
TCLKx	←=======→	TCLKx
RCLKx	←=======→	RCLKx

Additionally, the TCLKx and the RCLKx of DSP1 should be tied together, and the TCLKx and RCLKx of DSP2 should be tied together. Essentially, all four clocks will be connected.

In this test case, DSP1 will be internally generating RCLK and RFS. All other frame syncs and clocks in this system are set to be externally generated. It is important to note that in multichannel mode, the frame sync and clock should always be generated by the RFS and the RCLK and not the TFS or the TCLK. The TFS pin is often left unconnected, although it can be used as a data valid pin. TCLK should always be configured as an input in multichannel mode. In this example, both hardware platforms are using serial port 0.

Software Description

Listed at the end of this document is code for DSP1 and DSP2. In this example, both DSPs have sixteen 32 bit channels set up in their serial port control registers. It is important to note that in multichannel mode, both the SRCTLx register and the STCTLx register must have the same serial word length programmed.

When using multichannel mode, any channel can be used to transmit or receive. In this example, DSP1 is transmitting on the first eight channels, and DSP2 is receiving on the first eight channels (set up in the MTCS0 and the MRCS0 registers). The DMA on DSP1 is set up to transmit 16 words. Since there are only 8 channels activated, there will be two frame syncs being generated. (The first frame sync is generated, then the first eight words are sent over the eight channels. Then the next frame sync is generated and the second eight words are sent over the same eight channels.) DSP2 is set up in the same way, except to receive.

When running these two programs, DSP2 should be started first. This allows time for the serial port to be configured before the frame sync and data starts coming from DSP1.

In this system, the clock is being generated by DSP1's SRCTL0 register. Therefore bit 10 of the SRCTL0 register is set. Since DSP1 is only transmitting data, and not receiving, bit 18 (SDEN) is set in the STCTL0 register to enable a DMA, but not the SRCTL0 register.

Note that in multichannel mode several bits of the SRCTLx registers and the STCTLx registers must be cleared. These are bit 0 (SPEN), bit 10 (ICLK), bit 13 (TFSR), bit 14 (ITFS), and bit 17 (LAFS) in the STCTLx. In the SRCTLx register, bit 0 (SPEN), bit 13 (RFSR), bit 17 (LAFS), bit 21 (D2DMA) and bit 22 (SPL) all must be cleared.

SPORT Receive Comparison Registers

When in multichannel mode, the serial ports can take advantage of the receive comparison registers. This enables the DSP to compare the data coming in to the serial port and compare it to a know value. Depending on the outcome of this comparison, the DSP will either save the word, or discard it. This functionality can be used for interprocessor communication when connecting together several DSPs via the serial port. (See the ADSP-21160 Hardware Reference, page 9-33 for a an example of how this can be done)

To enable the comparison registers, bit 15 (IMODE) in the SRCTLx register must be set. Bit 20 (IMAT) controls what happens to the incoming data word after comparison. If this bit is cleared, the data is accepted if the comparison is false. If the bit is set, the data is accepted if the comparison is true. Note that in single channel mode, both bit 15 and bit 20 have different functions. They assume their Receive Comparison Register control functionality when the serial port is programmed in multichannel mode.

The word to be compared to is placed in the KEYWDx register. Specific bits can be masked out of the comparison by setting the corresponding bit in the KEYMASKx register. The MTCCSx register, which is used for companding channel selection when not in comparison mode, is used to select the channels that will be compared with KEYWDx. If a bit in the MTCCSx register is cleared, then the corresponding channel does not go through the comparison process and instead is always accepted. If the bit is set, then the DSP performs the comparison on the corresponding channel.

Example code for DSP2 which uses the receive comparison registers is included at the end of this document. In this example, the data value that is placed in the KEYWD0 register is 0x33333333. Every bit in the MTCCS0 register is cleared, so every bit in every incoming word is compared to what is in KEYWD0. The DSP will then discard any word that comes in to the serial port that is not 0x33333333. If 0x33333333 is received by the serial port, it is saved to memory. Additionally, an interrupt can be set up to be triggered any time there is a KEYWDx match.

Example Code for DSP1

```
/*Code for DSP1*/
/*In this example, DSP1 functions entirely as a transmitter*/
#include <def21160.h>
#define N 16             /* Size of buffer */

.section/DM     seg_dmda;
.VAR source[N]=
                0x11111111, 0x22222222, 0x33333333, 0x44444444,
                0x55555555, 0x66666666, 0x77777777, 0x88888888,
                0x99999999, 0xAAAAAAAA, 0xBBBBBBBB, 0xCCCCCCCC,
                0xDDDDDDDD, 0xEEEEEEEE, 0xFFFFFFFF, 0x00000000;

/* PM interrupt vector code */
.section/pm   seg_rth;
Reserved_1:     rti; nop; nop; nop;
Chip_Reset:     idle; jump start; nop; nop;

.section/pm seg_pmco;
start:
        /*clear receive and transmit serial port registers*/
        r0=0x0;
        dm(STCTL0)=r0;
        dm(SRCTL0)=r0;

        r0 = 0x000000FF;            /*Enable 1st 8 channels for transmit */
        dm(MTCS0) = r0;

        r0 = 0;
        dm(MTCCS0) = r0;            /* no companding on transmit */
        dm(MRCCS0) = r0;            /* no companding on receive */
        dm(MRCS0) = r0;             /* receive on no channels */

        r0=0x0;
        dm(TDIV0)=r0;
        r0 = 0x03e70001;      /* 20MHz SCLK, 20KHz RFS  */
        dm(RDIV0)=r0;

        /*configure the DMA*/
        r0=source;
        dm(II2) = r0;   /*internal memory location */
        r0=1;
        dm(IM2) = r0;  /* stride is 1 */
        r0=16;
        dm(C2) = r0;   /*move 16 words*/
```

```
          r0 = 0x000401F0;   /*Multi channel, DMA enabled, no chaining,32 bit words, externally */
          dm(STCTL0) = r0;     /*generated clock, 0 multi channel frame delay*/

          r0 = 0x0F8465F0;   /*set an internally generated clock and receive frame sync*/
          dm(SRCTL0) = r0;     /*Sets off the serial port*/

wait:     idle;              /* deluxe thumb-twiddling loop */
          jump wait;
```

Example Code for DSP2

```
/*Code for DSP2*/
/*In this example, DSP2 functions entirely as a receiver*/
#include <def21160.h>
#define N 16            /* Size of buffer */
.section/DM     seg_dmda;
.VAR dest[N]=           0x00000000, 0x00000000, 0x00000000, 0x00000000,
                                  0x00000000, 0x00000000, 0x00000000, 0x00000000,
                                  0x00000000, 0x00000000, 0x00000000, 0x00000000,
                                  0x00000000, 0x00000000, 0x00000000, 0x00000000;

/* PM interrupt vector code */
.section/pm   seg_rth;
Reserved_1:     rti; nop; nop; nop;
Chip_Reset:     idle; jump start; nop; nop;

.section/pm seg_pmco;
start:
          /*clear receive and transmit serial port registers*/
          r0=0x0;
          dm(STCTL0)=r0;
          dm(SRCTL0)=r0;

          r0 = 0x000000FF;            /*Enable 1st 8 channels for receive */
          dm(MRCS0) = r0;

          r0 = 0x00000000;
          dm(MTCCS0) = r0;             /* no companding on transmit */
          dm(MRCCS0) = r0;             /* no companding on receive */

          dm(MTCS0) = r0;             /* transmit on no channels */
```

```
        /*Clear RDIV or TDIV because they will be externally generated*/
        r0 = 0x0;
        dm(TDIV0)=r0;
        dm(RDIV0)=r0;

        /*configure the DMA*/
        r0=dest;
        dm(II0) = r0;   /*internal memory location */
        r0=1;
        dm(IM0) = r0;  /* stride is 1 */
        r0=2;
        dm(C0) = r0;   /*move 16 words*/

        r0 = 0x000001F0;   /*No DMA enabled because not transmitting anything*/
        dm(STCTL0) = r0;      /*0 multi channel frame delay*/

        r0=0x0F8401F0;     /*externally generated frame sync and clock.  DMA receive enabled*/
        dm(SRCTL0) = r0;      /*Sets off the serial port*/

wait:   idle;               /* deluxe thumb-twiddling loop */
        jump wait;
```

Example Code for DSP2 that implements Receive Comparison Registers

```
/*This is the Receive end of a multi-channel serial port connection that uses the receive*/
/* comparison registers*/
/*This program can be tested by using the code for DSP1 as the transmit SPORT (given above)*/
#include <def21160.h>
#define N 16              /* Size of buffer */
.section/DM     seg_dmda;
.VAR dest[N]=         0x00000000, 0x00000000, 0x00000000, 0x00000000,
                              0x00000000, 0x00000000, 0x00000000, 0x00000000,
                              0x00000000, 0x00000000, 0x00000000, 0x00000000,
                              0x00000000, 0x00000000, 0x00000000, 0x00000000;

/* PM interrupt vector code */
.section/pm   seg_rth;
Reserved_1:     rti; nop; nop; nop;
Chip_Reset:     idle; jump start; nop; nop;

.section/pm seg_pmco;
```

```
start:
        /*clear receive and transmit serial port registers*/
        r0=0x0;
        dm(STCTL0)=r0;
        dm(SRCTL0)=r0;

        r0 = 0x000000FF;
        dm(MRCS0) = r0;                 /*Enable 1st 8 channels for receive */
        dm(MTCCS0) = r0;               /*do comparison on first 8 channels */

        r0=0x0;
        dm(MRCCS0) = r0;               /* no companding on receive */
        dm(MTCS0) = r0;                /* transmit on no channels */

        /*Clear RDIV or TDIV because they will be externally generated*/
        r0 = 0x0;
        dm(TDIV0)=r0;
        dm(RDIV0)=r0;

        /*configure the DMA*/
        r0=dest;
        dm(II0) = r0;   /*internal memory location */
        r0=1;
        dm(IM0) = r0;  /* stride is 1 */
        r0=16;
        dm(C0) = r0;   /*move 16 words*/

        r0 = 0x000001F0;
        dm(STCTL0) = r0;

        /*Set up receive comparision resgisters*/
        r0=0x33333333;    /*compare with this word*/
        dm(KEYWD0)=r0;
        r0=0x0;
        dm(KEYMASK0)=r0;  /*Compare all bits*/

        r0=0x79481F0;          /*Save if data matches KEYWDx*/
        dm(SRCTL0) = r0;     /*enable receive comparison registers, enable multi-channel mode*/

wait:   idle;               /* deluxe thumb-twiddling loop */
        jump wait;
```