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

Interfacing the ADSP-2191 to an AD7476 via the SPI port.

Last modified: 8/08/01

Contributed By: H.Desai. European DSP Applications

The purpose of this note is to describe how to set-up an SPI interface between the ASDP-2191 and an A/D converter (AD7476) in both hardware and software. It will show how easy it is to connect an external SPI compatible peripheral device (such as A/D converters etc.) to the DSP via the SPI port.

The hardware for this system was tested using ADSP-2191-EZ-KIT LITE, the included asm and C code was built using the VisualDSP++ 2.0® tools.

Introduction

The ADSP-2191 has two independent serial peripheral interface ports (SPI0/1). In this example SPI0 is set up as a master.

The SPI is a full duplex, 4 wire, synchronous interface, which supports both master and slave modes and multi-master environments. The interface consists of two data pins (MOSI & MISO), one device select pin $\overline{\text{(SPISS}}$/PFx) and a clock pin (SCK). The programmable flag pins on the ADSP-2191 can be configured to behave like a device select signal. Using the flag pins the Master ADSP-2191 can select up to seven slave devices on each SPI port.

The SPI interface is essentially a shift register that serially transmits and receives data bits, one bit a time at SCK rate, to and from other SPI compatible devices. During an SPI transfer, data is simultaneously transmitted and received. A serial clock line synchronizes the shifting and sampling of the information on the two serial data lines.

Hardware Interface

In this application an AD7476 is connected to the SPI0 port.

The AD7476 is a 12bit ADC with a SPI interface. The part operates from a single power supply up to 5.25V and features throughput rates up to 1MSPS.

The AD7476 has 3 pins that connect to the DSP, the SCLK (which is an input from the DSP), the $\overline{\text{CS}}$ (also an input from the DSP) and the SDATA (an output to the DSP). The AD7476 is a 12bit converter but the conversion result is output in a 16bit word with four leading zeros followed by the MSB of the 12-bit result.

A voltage reference (ref198) is used as the power supply to the AD7476. The reference is used to decouple noise from the power supply and give more precise results. The output from the reference is a steady voltage (4.096V).

The configuration of connecting the reference device to the ADC is stated in the data sheets.

The input to the reference device is taken from the breadboard area of the ADSP2191, but can be something else i.e. signal generator.

The input to the ADC is taken from a POT.

The ADC output data line is connected to the DSP data input line (MISO) via a voltage devider, this is due to the output voltage from the ADC (4.096V) being higher then the input voltage of the DSP (3.3V).

The SPI pins on the DSP and the ADC are all originally at tri state hence pull-up resisters (10K) are used at the pins.

Software Description

Once all the hardware is connected it is time to generate the system software to control the SPI port. For this to work we need to have working software for the Master (DSP).

The assembly code has been broken into 4 files.

SPI_MAIN.asm –

This is the main routine used to call all the other source files in the project.

PRIORITY.asm –

This file is used to set up the interrupt.

The first thing that is always required when initializing an SPI interface is to set the OPMODE bit (bit 0) in the System Configuration Register (SYSCR).

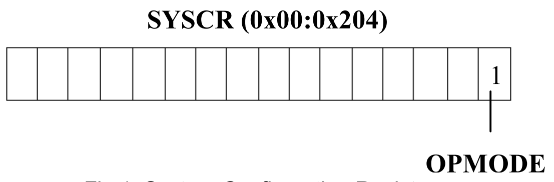


Fig 1: System Configuration Register

The state of the OPMODE pin during hardware reset determines if SPORT2 or the SPI ports are active, as the SPI ports are multiplexed with SPORT2. Setting this bit enables SPI0/1 and disables SPORT2.

Next, is to prioritize the peripheral interrupts. This is optional; the default priority at reset of each of the peripheral interrupts can be used.

For this example the peripheral interrupt is being assigned a different priority. The priority of each interrupt is set in the Interrupt Priority Registers (IPR0 – 3). In this example only the SPI0 interrupt is being used, and the peripheral interrupt is assigned priority 1, the second highest priority. This makes it interrupt 5 in the vector table (remember that the first four interrupts (0-3) are the core interrupts with the highest priority and all other interrupts are assigned after, starting at position 4 in the vector table).

IPR1 is set to 0xBB1B and IPR0/2/3 are set to 0xBBBB. The one indicates SPI0 set to priority 1 and the B's indicate all the other interrupts set to 11 (lowest priority).

Finally the SPI0 interrupt is unmasked in the IMASK register.

SPI_CONFIG.asm –

This source file sets up the SPI port.

Once the interrupt is enabled, the SPI port (hardware) needs to be configured.

The first thing to configure is the slave select pin by writing to the SPI0 Flag Register (SPIFLG0). In this register the individual SPI slave-select lines are enabled, when the SPI is enabled as a master. Bit 1 enables programmable flag 2 (PF2) as an output. Since the slave select line is active-low, the corresponding value of the pin (bit 9) is kept high until it is time to transmit/receive.

The handling of this register is determined by the value of clock phase bit (CPHA) in SPI control register.

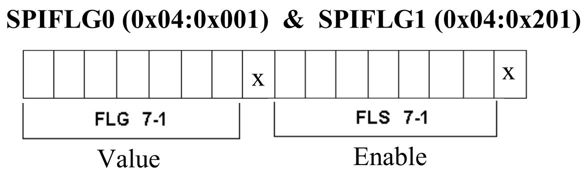


Fig 2: SPIx Flag Register

If CPHA = 1 the value of the flag bit is set by software control (FLG bits manually toggled by user). If CPHA = 0 the value is set by the SPI hardware (FLS bits in SPIFLGx toggled automatically).

Next the baud rate, the rate at which the SPI transfer will operate is set in the SPI Baud Rate Register (SPIBAUD0) for a master device.

The baud rate is determined by the following equation

$$\text{SCK} = \frac{\text{HCLK}}{(2 * \text{SPIBAUD})}$$

Writing a 0 or 1 to this register disables the serial clock.

Now finally the SPI Control Register (SPICTL0) is set to configure the SPI system. This register is used to enable the SPI interface, select the device as a master or slave, and determine the data transfer format and word size.

SPICTL0 (0x04:0x000)

0	1	0	1	1	0	0	1	0	0	0	0	1	0	0	

Fig 3: SPI Control Register

Bit 0-1 – Transfer initiate mode (TIMOD) instructs the DSP when to have the interrupt occurring and when to begin the transfer.

Bit 2-3 – instruct what to do when receive/transmit buffers are empty/full.

Bit 4 – used when DSP is configured as a slave device.

Bit 5 – needed when using multiple slaves

Bit 6-7 – reserved

Bit 8 – sets the transmission word size (SIZE)

Bit 9 – sets the data format (LSBF)

Bit 10 – Clock phase (CPHA)

Bit 11 – clock polarity (CPOL)

Bit 12 – Configures DSP as master or slave (MSTR)

Bit 13 – Enable open-drain (WOM) is needed in the event that you do not wish to have data-lines connected in a multi-master mode or multi-slave environment. As this is a single master-slave design, this feature is not enabled.

Bit 14 – this bit enables the SPI module (SPE).

Bit 15 – reserved

A note to remember is that certain components of this control register in both the master and slave devices need to be configured identically to one another.

The SPI clock phase (CPHA), polarity (CPOL), data format (LSBF) and word length (SIZE) must be the same in both master and slave devices, else the interface will not function properly.

From the data sheet of AD7476 and the timing diagram shown in fig 4, it can be seen that the ADC starts transmitting on the first falling clock edge and SCK high is the idle state, hence CPHA = 0 and CPOL = 1.

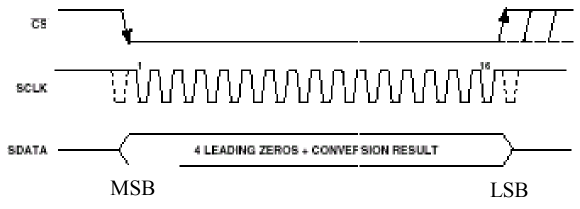


Fig 4: Timing diagram of AD7476.

Also the data format and word size information can be found in the data sheet of the AD7476.

As mentioned earlier the AD7476 outputs a 16-bit word with four leading zeros followed by the MSB of the 12-bit result.

Once all the registers are set it is time to start the transfer.

An output buffer has been set up to store the results into memory.

Finally the transfer is kicked-off with a dummy read of the SPI0 Receive data buffer (RDBR0) and the results written into the received_data buffer in memory.

For this example we do not need to write to the SPI0 flag register to enable and disable the slave on PF2 as CPHA = 0, which means that the FLG value is automatically set by the hardware.

ISR.asm –

This file contains the interrupt handler.

This is the main interrupt service routine. It uses the secondary set of registers to read the data from RDBR0 and store the data in the received_data buffer in memory.

C code

This example only contains one file, main.c, which contains four functions. The source code in the functions performs the same task as the assembler codes.

The memory mapped registers are accessed using io_space_read and io_space_write commands. The non-memory mapped registers are accessed using sysreg_read and sysreg_write commands. These commands allow easy access to the io-pages that contain the required registers to set up the system and are contained in the sysreg.h header file.

Results

From the diagrams below it can be seen that data is transmitted when CS goes low. The first four bits are all zeros (the four leading zeros sent by the ADC) and then the resulting data is sent. The clock starts toggling at the middle of the first data bit.

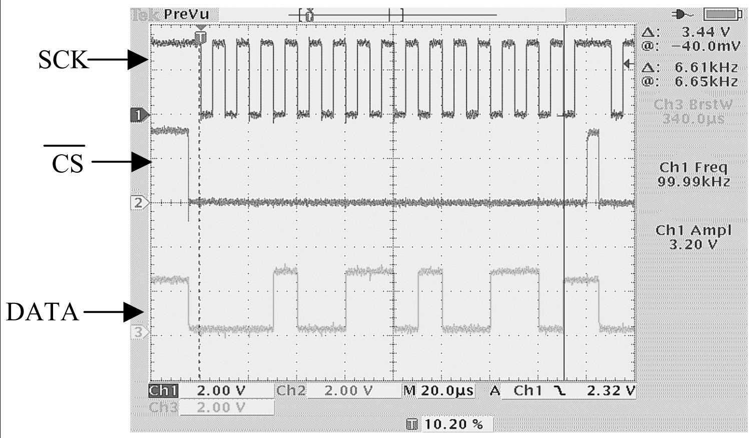


Fig 5: Results of SPI system

Conclusion

This note should have given you an idea on how to connect SPI compatible peripheral devices to the ADSP-2191 via the SPI port and also how to very simply configure the port (write a small piece of code to allow communication).

Finally please refer to additional documents to help getting a better understanding of this note.

References

- ADSP 2191 Hardware Reference Manual
- 2191 EZ-KIT LITE (hardware)
- ADSP-2191 EZ-KIT LITE Schematic
- Data sheet of AD7476
- Data sheet of REF198
- Data sheet of ADSP-2191
- www.analog.com

Appendix A

Listed below are the source codes used to illustrate the SPI
2191. (Please note that these included code modules were b
tools for the 219x processor family and the ADSP-2191 EZ

Assembler code

MAIN.asm

```
/*************************************************************************************************

File Name:          SPI interface to AD7476

Date Created:       06/01               H.Desai

PURPOSE:            SPI interface routine for connecting the AD7476 to the ADSP-2191.

*************************************************************************************************/

#include <def2191.h>

//------------------------------------------------------------------------------------------------------------------------------------------
//                                        GLOBAL & EXTERNAL DECLARATIONS
//------------------------------------------------------------------------------------------------------------------------------------------

.GLOBAL Start;
.EXTERN SPI1_Interrupt_Priority;
.EXTERN Initialization_of_SPI1;

//------------------------------------------------------------------------------------------------------------------------------------------
//                                        Program memory code
//------------------------------------------------------------------------------------------------------------------------------------------

.SECTION /pm program;
Start:
        call SPI0_Interrupt_Priority;
        call Initialization_of_SPI0;

wait_forever:
        jump wait_forever;

/*************************************************************************************************/
```

PRIORITY.asm

```
/*************************************************************************************************

Interrupt Priority Configuration

*************************************************************************************************/

#include <def2191.h>

//-----------------------------------------------------------------------------------------------------------------------------------------------
//                                              GLOBAL & EXTERNAL DECLARATIONS
//-----------------------------------------------------------------------------------------------------------------------------------------------

.Global SPI0_Interrupt_Priority;


//-----------------------------------------------------------------------------------------------------------------------------------------------
//                                              INTERRUPT PRIORITY CONFIGURATION
//-----------------------------------------------------------------------------------------------------------------------------------------------

.section/code program;
SPI0_Interrupt_Priority:
            IOPG = 0;
            ar=io(SYSCR);                               /* Map Interrupt Vector Table to Page 0*/
            ar = setbit 4 of ar;
            ar = setbit 0 of ar;                        /* Turn on SPI */
            io(SYSCR)=ar;

            DIS int;                                    /* Disable all interrupts */
            IRPTL = 0x0;                                /* Clear all interrupts */
            ICNTL = 0x0;                                /* Interrupt nesting disable */
            IMASK = 0;                                  /* Mask all interrupts */

            IOPG = Interrupt_Controller_Page;           /* Set Interrupt Priorities */
            ar = 0xBB1B;                                /* Set SPI0 to priority of 1 */
            io(IPR1) = ar;                              /* Set all other peripherals to lowest priority  */
            ar = 0xBBBB;
            io(IPR0) = ar;
            io(IPR2) = ar;
            io(IPR3) = ar;

            AY0=IMASK;
            AY1=0x0020;                                 /* Unmask SPI0 Interrupt */
            AR = AY0 or AY1;
            IMASK=AR;

            RTS;

/*************************************************************************************************/
```

SPI_CONFIG.asm

```
/*******************************************************************************************

SPI Port Configuration

*******************************************************************************************/

#include <def2191.h>

//--------------------------------------------------------------------------------------------------------------------------------
//                                             GLOBAL & EXTERNAL DECLARATIONS
//--------------------------------------------------------------------------------------------------------------------------------

.GLOBAL Initialization_of_SPI0;

//--------------------------------------------------------------------------------------------------------------------------------
//                                     DM DATA
//--------------------------------------------------------------------------------------------------------------------------------

.SECTION /dm dmdata;
.VAR        recieved_data[16];

//--------------------------------------------------------------------------------------------------------------------------------
//                                             SPI0 REGISTER INTIALIZATION

//--------------------------------------------------------------------------------------------------------------------------------

.SECTION /pm program;
Initialization_of_SPI0:
            IOPG = SPI0_Controller_Page;

            AR = 0xFF02;                            /* Enable Slave On PF2 */
            IO(SPIFLG0) = AR;

            AR = 0x50;                              /* Set SPI0 Baud rate, => SCLK0 ~= 50Khz*/
            IO(SPIBAUD0) = AR;

            AR = 0x5908;                            /* Set SPI0 Configuration Reigster */
            IO(SPICTL0) = AR;                       /* Enable SPI0 as MASTER */


            I0 = recieved_data;                     /* Set data buffer */
            M1 = 1;
            L0 = LENGTH(recieved_data);
            ax0 = I0;
            REG(B0) = ax0;

            ENA INT;

            IOPG = SPI0_Controller_Page;
            AR = IO(RDBR0);                         /* Dummy read from SPI0 */

            RTS;

/*******************************************************************************************/
```

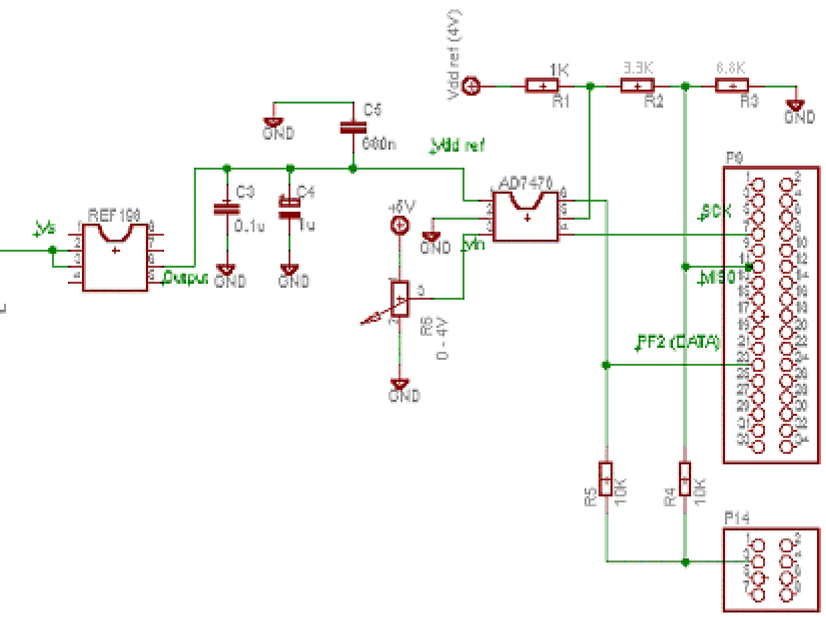
ISR.asm

```
/***********************************************************************************************

SPI Interrupt handler

***********************************************************************************************/

#include <def2191.h>

//-----------------------------------------------------------------------------------------------------------------------------------------
//                                          EXTERNAL DECLARATIONS
//-----------------------------------------------------------------------------------------------------------------------------------------

.EXTERN Start;

//-----------------------------------------------------------------------------------------------------------------------------------------
//                                          DM DATA
//-----------------------------------------------------------------------------------------------------------------------------------------

.SECTION /dm dmdata;
.VAR        counter = 0;
.VAR        save_io_page;

//-----------------------------------------------------------------------------------------------------------------------------------------
//                                                    PM Reset interrupt vector code
//-----------------------------------------------------------------------------------------------------------------------------------------

.section/pm IVreset;
            jump Start;
            nop; nop; nop;




//-----------------------------------------------------------------------------------------------------------------------------------------
//                                                    SPI0 ISR
//-----------------------------------------------------------------------------------------------------------------------------------------

.section/pm IVint5;
            ENA SR;
            AR = IOPG;
            dm(save_io_page) = AR;

            IOPG = SPI0_Controller_Page;

            AR = dm(counter);                              /* Interrupt counter  for debugging purposes*/
            AR = AR + 1;
            DM(counter) = AR;

            AR = IO(RDBR0);                                /* Read from SPI0 Receive Buffer Register */
            DM(I0 += M1) = AR;



            AR = dm(save_io_page);
            IOPG = AR;
            DIS SR;
```

C code

```
/************************************************************************************************
Title:      SPI - C-interface - ADSP-2191 Evaluation Board

Date  :     06/01

Information- Connection of an AC7476 to the ADSP-2191 via SPI.

************************************************************************************************/

#include<signal.h>
#include<sysreg.h>

//------------------------------------------------------------------------------------------------------------------------------------------
//                                                   GLOBAL & EXTERNAL DECLARATIONS
//------------------------------------------------------------------------------------------------------------------------------------------

void SPI0_Interrupt_Priority(void);
void Initialization_of_SPI0(void);
void SPI_recieve();

//------------------------------------------------------------------------------------------------------------------------------------------
//                                          DM DATA
//------------------------------------------------------------------------------------------------------------------------------------------

int received_data[16];
int i;

//------------------------------------------------------------------------------------------------------------------------------------------
//                                                   Program memory code
//------------------------------------------------------------------------------------------------------------------------------------------

void main(void)
{
            SPI0_Interrupt_Priority();                          /* call SPI0_Interrupt_Priority */

            Initialization_of_SPI0();                           /* call Initialization_of_SPI0 */

            sysreg_write(sysreg_IOPG , 0x4);                    /* Select IO page */
            received_data[i] = io_space_read(0x4);              /* Dummy read from SPI0 */

            interrupt(SIG_INT5,SPI_recieve);                    /* SPI0 interrupt masks and calls interrupt routine */

            for (;;)
            {
                        asm("idle;");
            }

}

/***********************************************************************************************/

void SPI0_Interrupt_Priority(void)
{
            sysreg_write(sysreg_IOPG , 0x1);                    /* Select IO page */
```

```
/*********************************************************************************************/

void Initialization_of_SPI0(void)
{
        sysreg_write(sysreg_IOPG , 0x0);              /* Select IO page */

        io_space_write(0x204, 0x11);                  /* Map Interrupt Vector Table to Page 0*/
                                                      /* Turn on SPI */

        sysreg_write(sysreg_IOPG , 0x4);              /* Select IO page */

        io_space_write(0x1, 0xff02);                  /* Enable Slave On PF2 */
        io_space_write(0x5, 0x50);                    /* Set SPI0 Baud rate, => SCLK0 ~= 50Khz*/
        io_space_write(0x0, 0x5908);                  /* Set SPI0 Configuration Reigster */
                                                      /* Enable SPI0 as MASTER */
}

/*********************************************************************************************/

void SPI_recieve()
{
static int count=0;

        sysreg_write(sysreg_IOPG , 0x4);              /* Select IO page */

        count++;                                      /* Interrupt counter */

        received_data[i] = io_space_read(0x4);        /* Read from SPI0 Receive Buffer Register */
        i++;

        if (i >= 80)
                i=0;
}

/*********************************************************************************************/
```



TITLE: SPI interface

Document Number:

REV:

Date: 8/14/2001 04:57:46p

Sheet: 1/1