Application Report
SLAA211–September 2004

Interfacing the ADS8402/ADS8412 to TMS320C6713 DSP

Lijoy Philipose *Data Acquisition Applications*

ABSTRACT

This application report presents a solution for interfacing the ADS8402 and ADS8412 16-bit, parallel interface converters to the TMS320C6713 DSP. The hardware solution consists of existing and orderable hardware, specifically the ADS8402EVM, 'C6713 DSK, and 5-6K interface board. The software demonstrates how to use the EDMA controller to efficiently collect data from the data converter. Discussed also are some key points to remember when using this software and modifying it for your application. Project collateral discussed in this application report can be downloaded from the following URL: www.ti.com/lit/zip/SLAA211.

Contents

List of Figures

List of Tables

1 Introduction

The ADS8402 converter is one of the first high-speed and high-resolution converters offered by the Texas Instruments' Burr Brown product line . It is an analog-to-digital converter with 16-bit resolution and a 1.25-MSPS sample rate. The upgrade device, the ADS8412, increases the sample rate to 2 MSPS. This application report presents one hardware and software solution for interfacing and using these two converters with the TMS320C6713 digital signal processor (DSP). The software developed uses the enhanced direct memory access (EDMA) controller, combined with Timer1, to collect 2048 samples. Discussed also are some of the key points to remember when interfacing the converters to host processors and upgrading from the ADS8402 to the ADS8412.

2 Hardware

The hardware solution involves theTMS320C6713 DSK ('C6713 DSK), 5-6K interface board, and the ADS8402EVM evaluation module. The hardware used in this report is available for order from Texas Instruments.

2.1 TMS320C6713 DSK

The TMS320C6713 DSP Starter Kit (DSK) not only provides an introduction to 'C6000 technology, but is powerful enough to use for the fast development of networking, communications, imaging, and other applications like data acquisition. For more information, search for part number TMDSDSK6713 on the TI Web site at http://www.ti.com.

2.2 ADS8402EVM

The ADS8402 evaluation module, or ADS8402EVM, is an easy way to test both the functional and dynamic performance of this 16-bit, analog-to-digital converter. The evaluation module includes only those circuits essential to demonstrate the performance of the converter and the interfacing to a parallel bus. These circuits are the analog input, reference, power, digital buffer circuits, and a simple decode logic. The digital inputs and outputs are buffered to isolate the converter from digital noise common to most shared-bus-type systems. The analog signal can be applied via standard 0.1-inch IDC header/socket(P1) or via SMA connectors (J2 and J4). The buffered data bus is available via 0.1-inch IDC header/socket connectors (P3). The ADS8402 control inputs are also made available via standard 0.1-inch IDC header/socket (J3). The decode logic which generates the convert-start, read, and reset signals are controlled via connector P2. These standard connectors enable the EVM to be plugged into most prototype boards for rapid evaluation. For additional information on this product, search the TI Web site. The ADS8412EVM is similar to the ADS8402EVM; only the A/D chip is replaced.

Table 1 lists the jumper settings used in this application report. The decode logic looks for the three inputs A0, A1, and A2. For this application report, DSP address lines A14, A15, and A16 are mapped to A0, A1, and A2, respectively.

Table 1. Jumper Settings for ADS8402EVM

DESIGNATOR	DESCRIPTION	JUMPER	
		PIN 1–2	PIN 2–3
W1	Selects 5 V for digital supply voltage	Short	
W2	$\overline{RD}$ mapped to 0xA0004000	Short	
W3	Apply inverted busy signal to interrupt pin of processor	Short	
W4	$\overline{RESET}$ mapped to 0xA0014000	Short	
W5	Convert-start signal mapped to 0xA000C000	Short	

2.3 5–6K Interface Evaluation Module

Many data acquisition evaluation modules (EVM) from Texas Instruments have a common set of connectors and signals at those connectors. The 5–6K interface board allows designers to easily connect those EVMs to the C5000 and C6000 family of digital signal processor starter kits.

The 5–6K interface board consists of two serial connectors, two signal conditioning areas, and a parallel interface. The ADS8402EVM plugs into connectors J10 (analog), J17 (data bus), J18 (control bus), and JP5 (power). See TI literature number SLAU104 for more information on the 5–6K interface board, or search the TI Web site for keyword *5–6K interface*.

Table 2 lists the jumper settings for the 5–6K interface board. Be sure to short across pin 5 and pin 6 of connector J13 on the 5–6K interface board. It routes the inverted BUSY from the ADS8402EVM to INTC on the 5–6K board and finally to the external interrupt number six (INT6) signal of the DSP.

Table 2. Jumper Settings for 5–6K Interface Board

DESIGNATOR	DESCRIPTION	JUMPER	
		PIN 1–2	PIN 2–3
W1	Apply DSP address lines A14 to A17 to ADS8402EVM address lines A0 to A3	OPEN	N/A
W2	Select +5VD from DSK		SHORT
W3	Select +3.3VD from DSK		SHORT
W4	Apply $\overline{\text{CL\_WR}}$	SHORT	
W5	Apply $\overline{\text{DC\_ARE}}$		SHORT
W6	Apply $\overline{\text{DC\_IS}}$		SHORT
W7 (1)	Interrupt signal is applied directly to J13.	SHORT	

(1) Only on board revision B.

2.4 Hardware Connections

One ADS8402 is mapped into memory space CE2 of 'C6713 DSP. The read, reset, and convert-start signals are generated by using a 3-to-8 decoder on the ADS8402EVM. A read operation from addresses 0xA0004000, 0xA000C000, and 0xA0014000 generates read, convert-start, and reset pulses, respectively, to the analog-to-digital converter. The 'C6713 has a 32-bit data bus; therefore, the BYTE line is tied LOW, enabling 16-bit bus operation. The BUSY signal is inverted and then applied to the external interrupt pin 6 of the DSP. The inversion of the BUSY signal was necessary because the EDMA interrupt controller only triggers off rising edges of external interrupts. The scheme allows for read operations to occur only during sampling mode and the first data in ad_buffer to be from a valid conversion cycle. The inversion of the BUSY signal would not be necessary if the CPU were used to service A/D interrupts, because it can be programmed to trigger on rising or falling edge. The next section explains why reading only during the sampling period limits the maximum sample rate.

The $\overline{\text{CE2}}$ signal of the DSP is tied to the $\overline{\text{CS}}$ signal of the converter. If other devices were on the parallel bus, another scheme would be necessary because $\overline{\text{CE2}}$ would go low anytime a memory access to this space was active. If power consumption is not a major concern in your application, the $\overline{\text{CS}}$ pin can be tied low permanently. The user needs only to provide read, reset, and convert-start signals.

Finally, the data bus is mapped LSB to LSB. The hardware connections are shown in Figure 1.

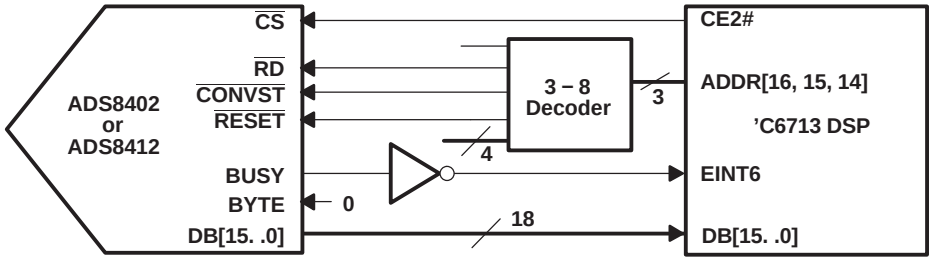

Figure 1. Hardware Connection

3 Software Interface

The software's objective is to collect a block of samples as quickly as possible, while freeing up the processor to perform other tasks. The processor should be alerted only when the samples are ready for processing. The most efficient way to do this is to use a couple EDMA channels, a timer, and interrupts. For this discussion, it is assumed that the reader is familiar with the DSP and its peripherals. If not, the reader should read the application reports and data sheets listed in the references section of this document.

The EDMA controller, along with a timer, is a highly efficient method of gathering data from an analog-to-digital converter (see Figure 2). The timer can be set to the desired frequency and used to trigger the EDMA channel to read or write from a memory location. This reading or writing causes the respective address lines to toggle. The decoder inputs tied to the address lines are used to create the read, reset, and convert-start pulses. The two channels used in this application are EDMA channels 2 and 6. A convert-start pulse is generated by a read from 0xA000C000 in CE2 space. The data read by the EDMA channel is stored in a variable named temp. Likewise, a read from address 0xA0004000 generates a read pulse to the converter. The sample data read is stored by the EDMA channel into the array ad_buffer[ ]. When all the samples have been collected and stored, the EDMA controller interrupts the DSP. The DSP services the hardware interrupt by disabling Timer1 and the EDMA channels.

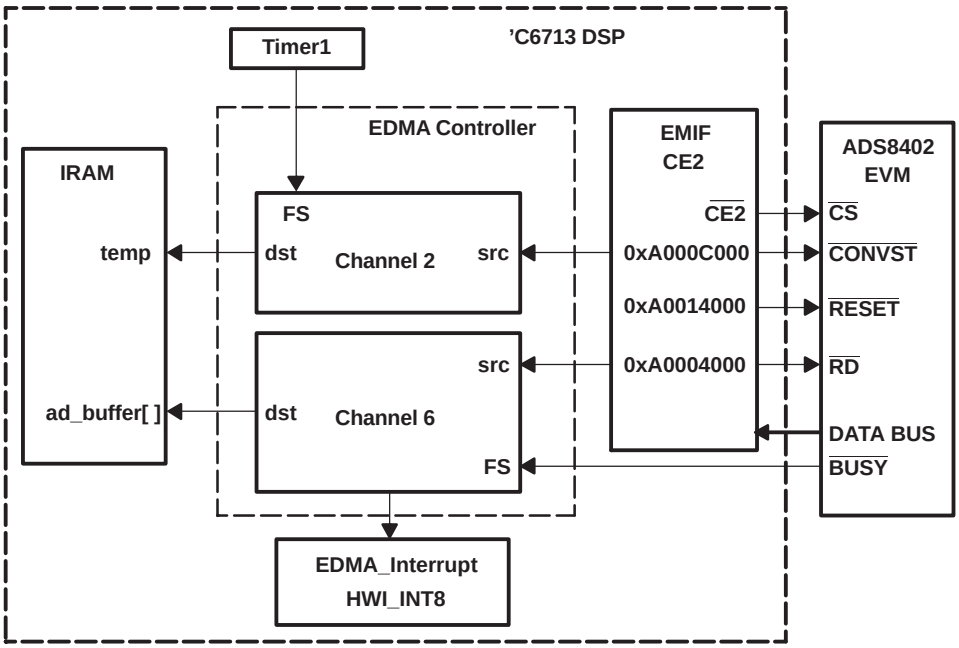

Figure 2. Data Transfer Block Diagram Using EDMA

It takes the DSP a fixed time to accomplish this task. As long as the timer is enabled, it continues to trigger new conversions, and the EDMA controller continues to start new conversions and to read data from the converter. At higher sampling rates, it takes the CPU a number of conversion cycles before it disables the timer and EDMA channels. The EDMA controller reads data from the converter and stores it at the destination address. For this reason, it is recommended that the EDMA channel be linked to transfer data to another location after it has completed collecting the samples. To see how this can be accomplished, open the file *config.cdb* in Code Composer Studio and expand as shown in Figure 3.

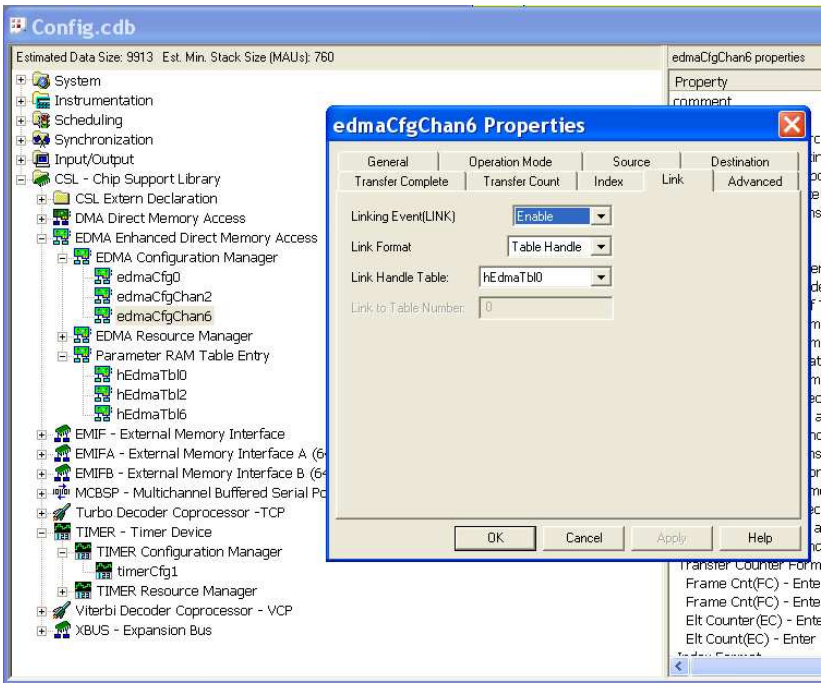

Figure 3. Screenshot of DSP/BIOS Configuration File

3.1 ADS8402 and ADS8412

The ADS8402 and ADS8412 are successive approximation register (SAR)-type converters. They operate in two modes: the sampling mode and the hold mode. In the sampling mode, the converter sample-and-hold capacitor is connected to the input pins of the connector. During this time, the analog input buffer is trying to settle the signal to half an LSB. Assuming a 4.096-V reference and a differential input range of 8.192 V, half of an LSB is 62.5 µV. In hold mode, the sample-and-hold capacitor is disconnected from the input pin. The converter tries to resolve the charge stored on the sample-and-hold capacitor. At the end of this process, a digital representation of the charge is ready to be read.

The ADS8402 and ADS8412 are pin compatible and similar from a software programmer's point of view. A few things like power-on initialization reset modes and timing requirements are different. The ADS8402 requires one reset pulse at power on, whereas the ADS8412 does not. For both converters, the first three conversion cycles are invalid and can be discarded. The $\overline{\text{RESET}}$ is an asynchronous input signal on both converters. The $\overline{\text{RESET}}$ pulse for the ADS8402 must be low for at least 20 ns, whereas for the ADS8412, it must be low for at least 25 ns. The ADS8402 can only be reset by using the dedicated reset signal. The ADS8412 can be reset in three ways: 1) falling edge of $\overline{\text{RESET}}$ signal, 2) falling edge $\overline{\text{CONVST}}$ signals while BUSY is HIGH and $\overline{\text{CS}}$ is LOW, and 3) falling edge of $\overline{\text{CS}}$ while BUSY is HIGH. The hardware used for development was the ADS8402EVM; therefore, in the main instruction, you see an instruction to issue a $\overline{\text{RESET}}$ pulse at power on. Figure 4 is a screenshot of the actual reset and power-on cycles.

The data sheet defines a *quiet zone* before and after the falling edge of the convert-start pulse. The falling edge of convert start takes the converter from sampling mode to hold mode. At times, the analog input voltage to resolve is stored in the sample-and-hold (S/H) capacitor. Any noise or ground bounce caused by high-speed digital signals, or any other signals, can be coupled into the ground reference of the device and cause an error in the voltage stored on the sample-and-hold capacitor. This error can show itself as an offset error. To minimize this effect, the data sheet defines a *quiet zone*. The quiet zone for the ADS8402 is 100 ns before and 40 ns after the falling edge of $\overline{\text{CONVST}}$ signal. The minimum sample time for the ADS8402 and ADS8412 is 150 ns and 100 ns, respectively. The quiet zone for the ADS8412 is 50 ns before and 40 ns after the falling edge of $\overline{\text{CONVST}}$ pulse. To run either converter at full speed, for best results, the host processor must be able to read the data within 50 ns after BUSY goes low. This requirement must be considered when designing the interface to the host processor.

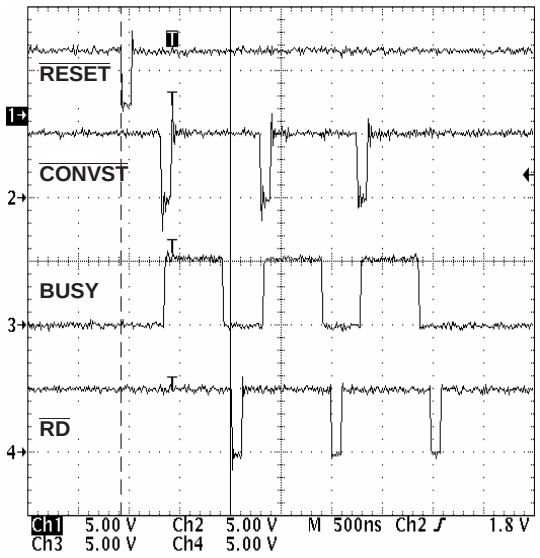

Figure 4. Scope Screenshot Showing Power-On Initialization Cycles

3.2 'C6713 DSP

The TMS320C6713 DSP and its respective peripherals (i.e., Timer1, EMIF, and EDMA) need to be set up to work with the converter. It is assumed that the reader has a working understanding of the host processor; therefore, the DSP setup is not covered in detail in this application report. For more information, see the downloadable code and the references listed at the end of this application report.

The settings for the various peripherals can be found and modified in the config.cdb file, as shown in Figure 3. The seed file for the DSP/BIOS configuration file was the dskC6713.cdb file

Immediately following this paragraph are the registers settings for EDMA channel 2, which is used to trigger a conversion cycle. The source is the address which generates the convert-start pulse through the decoder on the ADS8402EVM. The destination for the data transfer is a variable named temp. This variable is used to store data that can be discarded. The EDMA controller should transfer NUMSAMPLES, or 2048 sample data points. Each transfer frame is synchronized to a Timer1 event. As a result, the frequency of each transfer or convert-start pulse is the frequency of Timer1.

```
/* Config Structures */
EDMA_Config edmaCfgChan2 = {
 0x48020001, /* Options */
 0xA000C000, /* Source Address - Numberic */
 0x00000000, /* Transfer Counter - Numberic */
 (Uint32) &temp, /* Destination Address - Extern Decl. Obj */
 0x00000000, /* Index register - Numberic */
 0x00010000 /* Element Count Reload and Link Address */
```

The registers settings for EDMA channel 6 follow. Recall that this channel is used to read data from the converter. The source is the address which generates the read pulse using the decoder on the ADS8402EVM. The destination for the data transfer is the array named ad_buffer. The array is BLOCK_SZ long. The EDMA controller should transfer NUMSAMPLES, in this case 2048 sample data. Each EDMA transfer is synchronized to the external interrupt 6; it is the inverted BUSY signal from the ADS8402EVM.

```
EDMA_Config edmaCfgChan6 = {
 0x28360003, /* Option */
 0xA0004000, /* Source Address - Numeric */
 0x00000000, /* Transfer Counter - Numeric */
 (Uint32) ad_buffer, /* Destination Address - Extern Decl. Obj */
 0x00000000, /* Index register - Numeric */
 0x00010000 /* Element Count Reload and Link Address */
};
```

At a higher sampling rate, the DSP cannot disable the timer and the EDMA channels immediately after collecting 2048 samples. So long as the timer and the EDMA channels are enabled, they continue to trigger and store conversion data. The data is stored at the respective EDMA channel 6 destination register. Therefore, it is possible that the last point in the data array will be overwritten many times before the EDMA channel is disabled. To avoid corruption of sampled data, the EDMA channel is linked to configuration edmaCfg0. After the *read* EDMA channel captures 2048 samples, the EDMA controller stores data in the variable temp.

```
EDMA_Config edmaCfg0 = {
 0x28160001, /* Option */
 0xA0004000, /* Source Address - Numeric */
 0x00000000, /* Transfer Counter - Numeric */
 (Uint32) &temp, /* Destination Address - Extern Decl. Obj */
 0x00000000, /* Index register - Numeric */
 0x00010000 /* Element Count Reload and Link Address */
};
```

The register setting for the Timer1 which sets the sampling rate follows. The Timer1 input clock source is the CPU CLOCK divided by 4.

Timer1 input clock = 225E6/4 = 56.25 MHz

The formula for setting the sampling frequency (F_s) is F_s = 56.25e6 /(2 × Period Value) or Period Value = 56.25e6/(2 × Fs)

```
TIMER_Config timerCfg1 = {
 0x00000311, /* Control Register (CTL) */
 0x0000001B, /* Period Register (PRD) */
 0x00000000 /* Counter Register (CNT) */
};
```

The Timer1 output is set for clock mode. The ADS8402 requires a quiet zone of 100 ns before and 50 ns after CONVST start falling edge. The highest sample rate with this particular interface solution is about 1.0417 MSPS. It takes the EDMA controller about 130 ns and BUSY rising edge to generate a read pulse (see Figure 4). For this reason, the Period Value register is set to 27. To change the timer frequency, open the file config.cdb in Code Composer Studio and expand as shown in Figure 3. Right click timerCfg1, select properties, and select counter control tab.

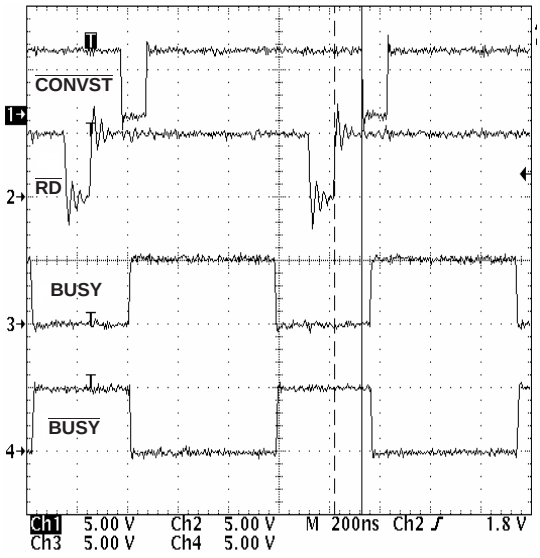

Figure 5. Screenshot Showing General Timing

4 Conclusion

This application report presents one solution for interfacing the ADS8402 and ADS8412 converters to the ‘C6713 DSP. The ADS8402EVM plugs onto the 5-6K interface board, which in turn, plugs onto the ‘C6713 DSK. All the hardware used for this application report can be ordered directly from Texas Instruments, making this solution a relatively inexpensive and low-stress hardware design. In this application, it was not possible to operate the converters at full speed because of the quiet-zone requirement of the converter and the desire to read only the sampling time. It is possible to design a system so that the read operation occurs after the quiet zone. The *quite-zone* requirement restricts activity before and after the convert-start signal goes low. Figure 4 shows that it takes the DSP about 130 ns to generate a read pulse after receiving a rising edge on the interrupt pin. One solution then is to present the BUSY signal to interrupt pin 6, instead of inverted BUSY. This forces the EDMA controller to read sample data during the conversion cycle, thus allowing much faster sustained sample rates. See the application report entitled *Interfacing the ADS8411 to the TMS320C6713 DSP* (SLAA212) to discover how this can be accomplished. In that application note, the converter is run at 2 MHz and a ping-pong buffering scheme is implemented using the EDMA controller. Generally, during conversion time, the IC should be kept free of switch currents that could disrupt the ground reference plane of the converter.

5 References

1. *TMS320C621x/TMS320C671x EDMA Architecture* (SPRA996)
2. *TMS320C6000 Enhanced DMA: Example Applications* (SPRA636)
3. *TMS320C6000 DSP Enhanced Direct Memory Access (EDMA) Controller Reference Guide* (SPRU234)
4. *Interfacing the ADS8411 to the TMS3206713 DSP* (SLAA212)
5. ADS8402 data sheet (SLAS154)
6. ADS8412 data sheet (SLAS384)
7. TMS320C6713 data sheet (SPRS186)

MAIN.C

```
/*********************************************************************/
 /* File: main.c */
 /* Description: Program for interfacing ADS8402 */
 /* to a 'C6713 DSK. Program uses the Timer1 to trigger a */
 /* convst# pulse, at about 1.04MHz using EDMA channel 2. */
 /* Inverted BUSY signal is used to trigger EDMA channel 6 to */
 /* read from A/D and store data into ad_buffer[]. Once */
 /* BLOCK_SZ of data is captured, the EDMA will interrupt CPU. */
 /* CPU will then halt EDMA channels and timer1. */ /* */
/* AD converter address: CE2 memory space */
 /* 0xA0004000 (RD#) */
 /* 0xA000C000 (CONVST#) */
 /* 0xA0014000(RESET#) */
 /* Hardware Connections: */
 /* CS# => CE2# */
 /* RD# => Generated from 3-8 decoder mapped to 0xA0004000 */
 /* CONVST# => Generated from 3-8 decoder mapped at 0xA000C000 */
 /* RESET# => Generated from 3-8 decoder mapped at 0xA0014000 */
 /* BUSY => Inverted then wired to EXTERNAL INT6 */
 /* AUTHOR : DAP Application Group, L. Philipose, Dallas */
/* CREATED 2004(C) BY TEXAS INSTRUMENTS INCORPORATED. */
 /* VERSION: 1.0 */
 /*********************************************************************/
Include Header File */
#include "Configcfg.h"
#include "dc_conf.h"
/* include files for DSP/BIOS */
#include <std.h>
#include <swi.h>
#include <log.h>
#include <rtdx.h> /* RTDX */
/* include files for chip support library */
#include <csl.h>
#include <csl_legacy.h>
#include <csl_irq.h>
#include <csl_timer.h>
#include <csl_edma.h>
/*Function Prototypes*/
void init_dsk (void);
void init_adc();
/* Create the buffers. we want to align the buffers to be cache friendly */
/* by aligning them on an L2 cache line boundary. */
#pragma DATA_ALIGN (ad_buffer,BLOCK_SZ);
signed short ad_buffer [BLOCK_SZ]; /*data from AD, written to by EDMA */
signed short temp, n=0;
void main (void)
{
 int i;
 /* intialize the EMIF */
 init_dsk();
 init_adc();
 for (i=0; i<=BLOCK_SZ; i++){ /*Initialize data buffers */
 ad_buffer[i]=0x0000;
}
```

```
/* Enable the EDMA controller interrupt */
 IRQ_reset (IRQ_EVT_EDMAINT); /*Reset EDMA interrupt */
 IRQ_reset (IRQ_EVT_EDMAINT);
 EDMA_intDisable(TCCINTNUM6); /*Disable EMDA interrupt */
 EDMA_intClear(TCCINTNUM6); /*Clear EDMA interrupt */
 EDMA_intEnable(TCCINTNUM6); /*Enable EDMA interrupt */
/*Configure EDMA Channels, clear and enable them */
 EDMA_config(hEdmaCha6, &edmaCfgChan6);
 EDMA_config(hEdmaCha2, &edmaCfgChan2);
 EDMA_clearChannel(hEdmaCha2);
 EDMA_clear(hEdmaCha6);
 EDMA_enableChannel (hEdmaCha2); /*Enable EMDA channel 2 -CONVST# */
 EDMA_enableChannel(hEdmaCha6); /*Enable EDMA channel 6 -RD# */
/*Configuratjon of Timer1 were completed by DSP/BIOS */
/*before entering main.c */
/*Only need to enable them here. */
 TIMER_start (hTimer1); /*Start A/ CONVST# trigger */
 /*Timer Pervalue = (225e6/4)/fs*/
 /*Go sample at 1.04MSPS*/
]
```

Functions.C

```
/*********************************************************************/
 /* File: functions.c */
 /* Description: Functions for interfacing ADS8402 to C6713 */
 /* init_dsk(), init_adc(), hwiDMA_isr(), swiEnablePrephFunc() */
 /* AD converter address: CE2 memory space */
 /* 0xA0004000 (RD#) */
 /* 0xA000C000 (CONVST#) */
 /* 0xA0014000 (RESET#) */
 /* Hardware Connections: */
 /* CS# => CE2# */
 /* RD# => Generated from 3-8 decoder mapped to 0xA0004000 */
 /* CONVST# => Generated from 3-8 decoder mapped at 0xA000C000 */
 /* RESET# => Generated from 3-8 decoder mapped at 0xA0014000 */
 /* BUSY => Inverted then wired to EXTERNAL INT6 */
 /* AUTHOR : DAP Application Group, L. Philipose, Dallas */
 /* CREATED 2004(C) BY TEXAS INSTRUMENTS INCORPORATED. */
 /* VERSION: 1.0 */
 /*********************************************************************/
Include Header File */
#include "Configcfg.h"
#include "dc_conf.h"
#include <csl_legacy.h>#in
/*Function Prototypes*/
void swiEnablePrephFunc();
extern signed short temp, ad_buffer[BLOCK_SZ];/* Raw buffer for AD data */
/*********************************************************************/
/* init_dsk() */
/* This initializes the EMIF */
/*********************************************************************/
(void init_dsk(void)
 {
 UINT32 gblctl,ce0ctl,ce1ctl,ce2ctl,ce3ctl,sdctl,sdtim,sdext;
 /* intialization of the EMIF */
 /* RBTR8,SSCRT,CLK2EN,CLK1EN,SSCEN,SDCEN,NOHOLD */
 gblctl = EMIF_MK_GBLCTL( 0, 0, 1, 0, 0, 0, 0);
 /* RDHLD,MTYPE,RDSTRB,TA,RDSETUP,WRHLD,WRSTRB,WRSETUP */
 ce0ctl = EMIF_MK_CECTL( 0, 3, 0, 0, 0, 0, 0, 0);
 /* RDHLD,MTYPE,RDSTRB,TA,RDSETUP,WRHLD,WRSTRB,WRSETUP */
 ce1ctl = EMIF_MK_CECTL( 0, 2, 0, 0, 0, 0, 0, 0);
 /* RDHLD,MTYPE,RDSTRB,TA,RDSETUP,WRHLD,WRSTRB,WRSETUP */
 ce3ctl = EMIF_MK_CECTL( 0, 2, 0, 0, 0, 0, 0, 0);
 /* TRC,TRP,TRCD,INIT,RFEN,SDWID,SDCSZ,SDRSZ,SDBSZ */
 sdctl = EMIF_MK_SDCTL( 7, 1, 1, 1, 1, 0, 1, 0, 0);
 /* PERIOD,XRFR */
 sdtim = EMIF_MK_SDTIM( 1562, 0);
 sdext = EMIF_SDEXT_NA;
 /* make CE2 control register value */
 /* This is the CE space used by the ADS8402. */
 /* Use the timing values from dc_conf.h: */
 ce2ctl = EMIF_MK_CECTL(
 EMIF_CECTL_RDHLD_OF (RDHLD), /* read hold */
 EMIF_CECTL_MTYPE_ASYNC32,
 EMIF_CECTL_RDSTRB_OF (RDSTRB), /* read strobe */
 EMIF_CECTL_TA_NA,
```

```
 EMIF_CECTL_RDSETUP_OF (RDSETUP), /* read setup */
 EMIF_CECTL_WRHLD_OF (WRHLD), /* write hold */
 EMIF_CECTL_WRSTRB_OF (WRSTRB), /* write strobe */
 EMIF_CECTL_WRSETUP_OF (WRSETUP) /* write setup */
 };
 /* configure the EMIF */
 EMIF_ConfigB(gblctl,ce0ctl,ce1ctl,ce2ctl,
 ce3ctl,sdctl,sdtim,sdext);
 return;
} /* end init_dsk() */
/************************************************************************/
/* init_adc() */
/* This initializes the ADC */
/************************************************************************/
void init_adc()
{ int i;
 /*Initialize A/D */
 temp = *ADS8412_RESETZ; /*Reset ADC*/
 for (i=0; i<=10; i++) {};
 temp = *ADS8412_CONVSTZ; /*Discard the first three conversions*/
 for (i=0; i<=1; i++) {};
 temp = *ADS8412_RD;
 temp = *ADS8412_CONVSTZ;
 for (i=0; i<=10; i++) {};
 temp = *ADS8412_RD;
 temp = *ADS8412_CONVSTZ;
 for (i=0; i<=10; i++) {};
 temp = *ADS8412_RD;
}
/************************************************************************/
/* hwiDMA_isr(): */
/* Hardware Interrup Function disables EDMA channels and */
/* Timer0, Then post software interrupt. */
/************************************************************************/
void hwiDMA_isr()
{ int i=0;
 TIMER_pause(hTimer1);
 EDMA_disableChannel(hEdmaCha2); /*Disable EMDA channel 2-CONVST#*/
 EDMA_disableChannel(hEdmaCha6); /*Disable EMDA channel 2 -RD#*/
 IRQ_reset(IRQ_EVT_EDMAINT); /*Reset EDMA interrupt */
/*Uncomment next line if continuous capture of BLOCK_SZ */
/*of data is desired*/
// SWI_post (&swiEnablePreph);
}
/************************************************************************/
/*swiEnablePrephFunc: */
/* Software Interrupt Function configures EDMAC2 and */
/* EDMAC6, enables respective EDMA channels and Timer1 */
/************************************************************************/
void swiEnablePrephFunc ()
{
 temp = *ADS8412_RESETZ; /*Reset ADC*/
 EDMA_clearChannel(hEdmaCha6);
 EDMA_clearChannel(hEdmaCha2);
 IRQ_enable(IRQ_EVT_EDMAINT);
 EDMA_intDisable(TCCINTNUM6); /*Disable EDMA interrupt */
 EDMA_intClear(TCCINTNUM6); /*Clear EDMA interrupt */
 EDMA_intEnable(TCCINTNUM6); /*Enable EDMA interrupt */
 EDMA_config(hEdmaCha6, &edmaCfgChan6);
 EDMA_config(hEdmaCha2, &edmaCfgChan2);
 EDMA_enableChannel (hEdmaCha2); /*Enable EMDA channel 2 -CONVST#*/
 EDMA_enableChannel(hEdmaCha6); /*Enable EDMA channel 6 -RD# */
 TIMER_start(hTimer1); /*Start A/D CONVST# trigger */
}
/************************************************************************/
/* End Functions.c */
```

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

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265