Application Report
SLAA212–September 2004

Interfacing the ADS8401/ADS8411 to TMS320C6713 DSP

Lijoy Philipose *Data Acquisition DAC*

ABSTRACT

This application report presents a solution for interfacing the ADS8401 and ADS8411 16-bit, parallel interface converters to the TMS320C6713 DSP. The hardware solution consists of existing hardware, specifically the ADS8411EVM, 'C6713 DSK, and 5-6K interface board. The software demonstrates how to use an EDMA ping-pong buffer and Timer1 peripherals to collect data at 2 MSPS. 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: http://www.ti.com/lit/zip/SLAA212.

Contents

List of Figures

List of Tables

1 Introduction

The ADS8401 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, ADS8411, increases the sample rate to 2 MSPS. This application report presents a 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, in combination with Timer1, to collect 4096 samples. Discussed also are some key points to remember when interfacing the converters to host processors and upgrading your system from ADS8401 to the ADS8411.

2 Hardware

The hardware solution involves the TMS320C6713 DSK (‘C6713 DSK), 5-6K interface board, and the ADS8401EVM 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 ADS8411EVM

The ADS8411 evaluation module, or ADS8411 EVM, 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 interfacing it to a parallel bus. These circuits include the analog input, reference, power, digital buffer circuits, and 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 connector (J2). The buffered data bus is available via 0.1-inch IDC header/socket connectors (P3). The ADS8401 control inputs are also made available via standard 0.1-inch IDC header/socket (J3). The decode logic used to generate 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 ADS8401EVM shares the same PWB as ADS8411EVM, but differs only in the components installed. See the ADS8411EVM user’s guide for more information. Table 1 lists the jumper settings used in this application note. The decode logic looks for three inputs A0, A1, and A2. For this application, DSP address lines A14, A15, and A16 are mapped to A0, A1, and A2, respectively.

Table 1. Jumper Settings for ADS8411EVM

DESIGNATOR	DESCRIPTION	JUMPER	
		PIN 1-2	PIN 2-3
W1	$\overline{RD}$ mapped to 0xA0004000	Short	
W2	$\overline{RESET}$ mapped to 0xA0014000	Short	
W3	Convert-start signal mapped to 0xA000C000	Short	
W4	Apply BUSY to P2 pin19.	Short	N/A

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 ADS8411EVM 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 5-6K interface board. Be sure to short across pins 5 and 6, of connector J13 on the 5-6K interface board. It routes the BUSY signal from the ADS8401EVM to INTC on the 5-6K board and finally to external interrupt number six (INT6) signal to 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 ADS8411EVM address lines A0 to A3	OPEN	N/A

Table 2. Jumper Settings for 5-6K Interface Board (continued)

DESIGNATOR	DESCRIPTION	JUMPER	
		PIN 1-2	PIN 2-3
W2	Select +5VD from DSK		SHORT
W3	Select +3.3VD from DSK		SHORT
W4	Apply $\overline{CL\_WR}$	SHORT	
W/5	Apply $\overline{DC\_ARE}$		SHORT
W/6	Apply $\overline{DC\_IS}$		SHORT
W/7[(1)]	Interrupt signal is applied directly to J13.	SHORT	

(1) Revision B of 5-6K board.

2.4 Hardware Connections

One ADS8411 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 ADS8411EVM. Address lines A16, A15, and A14 are used to generate read, convert-start, and reset pulses 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 applied to the external interrupt pin 6 of the DSP. $\overline{CE2}$ signal of the DSP is tied to $\overline{CS}$ of the converter. If other devices were on the parallel bus, another scheme would need to be devised because $\overline{CE2}$ would go low anytime a memory access to that space was active. If power consumption is not a concern in your application, $\overline{CS}$ can be tied low permanently, because it is the only device on the bus. The user only needs to provide read, reset, and convert-start signals.

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

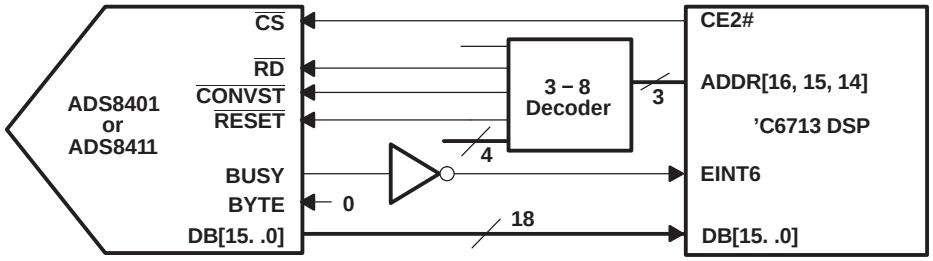

Figure 1. Hardware Connection

3 Software Interface

The software interface 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 study the application notes 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 a 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 be stored in the variable *temp*. Likewise, a read from address 0xA0004000 generates a read pulse to the converter. The program is written to collect 4096 continuous samples. The first 2048 sample data read are stored by the EDMA channel into array pingBuf[]. The second set of 2048 samples are stored in array pongBuf[]. After each 2048 samples, the EDMA controller interrupts the DSP. The DSP services the second hardware interrupt by disabling Timer1 and the EDMA channels.

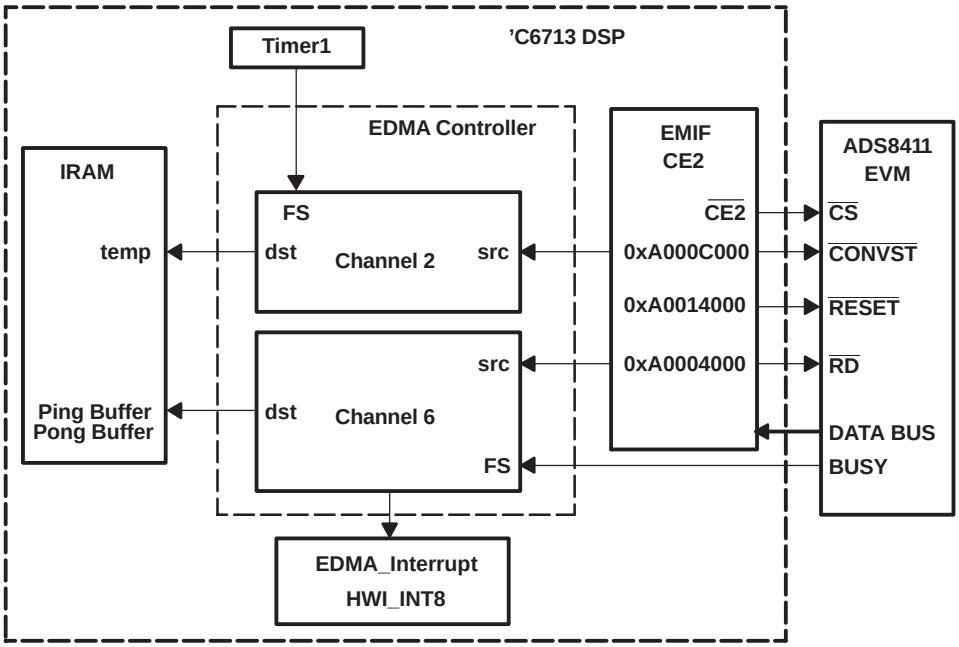

Figure 2. Data Transfer Block Diagram Using EDMA

It takes the DSP a fixed time to accomplish this task. So long as the timer is enabled, it continues to trigger new conversions, and the EDMA continues to start new conversions and 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 desired number of samples. To see how this can be accomplished, open the file config.cdb in Code Composer Studio and expand as shown in Figure 3. EDMA configuration 6 (edmaCfgChan6) tells the EDMA controller to write 2048 words to pingBuf[]. EDMA configuration 6A (edmaCfgChan6a) forces the EDMA controller to write 2048 words to pongBuf[]. EDMA configuration 0 (edmaCfg0) forces the EDMA controller to write data read from the converter to variable *temp*. The EDMA continues to write the samples to *temp* until the DSP disables Timer1 and the respective EDMA channels.

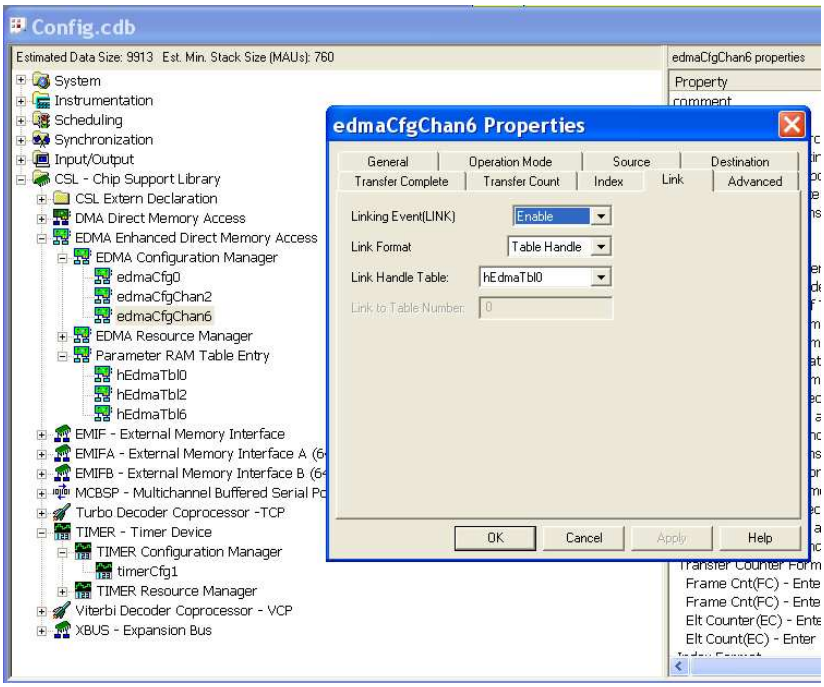

Figure 3. Screenshot of DSP/BIOS Configuration File

3.1 ADS8401 AND ADS8411

The ADS8401 and ADS8411 are successive approximation register (SAR)-type converters. They work in two modes: the sample mode and the hold mode. In the sample 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 single-ended input range of 4.096 V, half of an LSB is 62.5 µV. In the hold mode, the sample-and-hold capacitor is disconnected from the input pin. In this mode, the converter is trying 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 ADS8401 and ADS8411 is pin compatible and similar from a software programmer's point of view. A few things like power-on initialization reset modes and a few timing requirements are different. The ADS8401 requires one reset pulse at power on, whereas the ADS8411 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 RESET pulse for the ADS8401 must be low for at least 20 ns, whereas for the ADS8411, it must be low for at least 25 ns. The ADS8401 can be reset only using the dedicated reset signal. The ADS8411 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,
3. Falling edge of $\overline{\text{CS}}$ while BUSY is HIGH.

The hardware used for development was the ADS8411EVM; therefore, you see the reset instruction in function adc_init() commented out. Figure 3 is a screenshot with the reset pulse and power-on cycles.

The data sheet defines a *quiet zone* before and after the falling edge of convert-start pulse. The falling edge of the convert-start signal takes the converter from the sampling mode to the 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 ADS8401 is 100 ns before and 40 ns after the falling edge of $\overline{\text{CONVST}}$ signal. The minimum sample time for the ADS8401 and ADS8411 is 150 ns and 100 ns, respectively. The quiet zone for the ADS8411 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 meet this requirement. This requirement must be considered when designing the interface to the host processor.

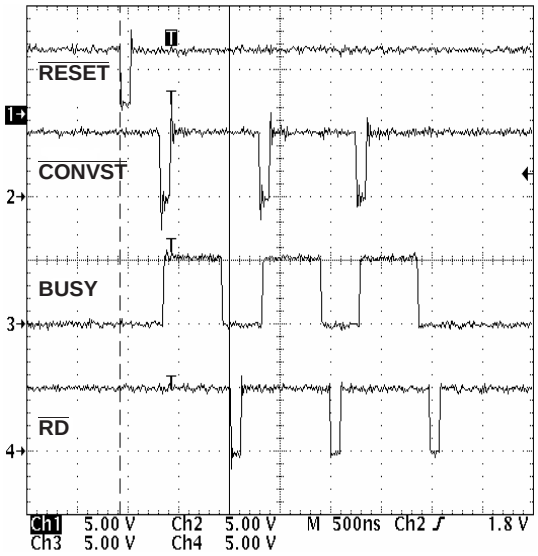

Figure 4. Scope Screenshot Showing Power-on Initialization Cycles

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 the reader has a working understanding of the host processor; therefore, the DSP setup is not covered in much detail in this application report. For more information, see the downloadable code and references listed in Section 5.

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.

The following are 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 ADS8411EVM. The destination for the data transfer is a variable named temp. That variable is used to store data that can be discarded. The EDMA should transfer NUMSAMPLES, or 2048 sample data points. (NUMSAMPLES is defined in the " dc_conf.h " file.) 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 the Timer1.

```
EDMA_Config edmaCfgChan2 = {
0x48020001, /* Option */
0xA000C000, /* 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 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 ADS8411EVM. An array called *pingBuf* is the destination for the first block of 2048 data points. The destination for the second block of 2048 data is the array *pongBuf*. Both arrays are BLOCK_SZ (2048) words long. The EDMA channel 6 transfers NUMSAMPLES(2048) and then loads the second configuration, which sets the destination address to the pongBuf array. After 4096 words are transferred from the A/D converter, it loads the third configuration which sets the destination address to the variable

temp. Each EDMA transfer is synchronized to the external interrupt 6. In this case, it is the BUSY signal from the ADS8411EVM. EDMA transfers are synchronized to a rising edge of the frame sync signal. The frame sync signal is the rising edge of the BUSY. The rising edge of BUSY indicates a start of a conversion and not the end. Therefore, the first read operation and the first point in the pingBuf[] array should be discarded.

```
EDMA_Config edmaCfgChan6 = {
0x28360003, /* Option */
0xA0004000, /* Source Address - Numeric */
0x00000000, /* Transfer Counter - Numeric */
(Uint32) pingBuf, /* Destination Address - Extern Decl. Obj */
0x00010001, /* Index register - Numeric */
0x00010000 /* Element Count Reload and Link Address */
};
EDMA_Config edmaCfgChan6A = {
0x48360003, /* Option */
0xA0004000, /* Source Address - Numeric */
0x00000000, /* Transfer Counter - Numeric */
(Uint32) pongBuf, /* Destination Address - Extern Decl. Obj */
0x00010001, /* Index register - Numeric */
0x00010000 /* Element Count Reload and Link Address */
};
```

At higher sampling rates, 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 address specified in the 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 4096 samples, the EDMA begins storing sample data to the variable temp. By using the linking feature of the EDMA controller, it was easy to implement a ping-pong buffer and to capture a block number of data samples at 2 MSPS.

```
EDMA_Config edmaCfg0 = {
0x48160003, /* Option */
0xA0004000, /* Source Address - Numeric */
0x00000000, /* Transfer Counter - Numeric */
(Uint32) &temp, /* Destination Address - Extern Decl. Obj */
0x00010001, /* Index register - Numeric */
0x00010000 /* Element Count Reload and Link Address */
};
```

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

Timer1 Input Clock = 225E6/4 = 56.25MHz

The formula for setting the sampling frequency (Fs) is

F_s = 56.25e6 /(2*Period Value)

Or

Period Value = 56.25e6/(2*F_s)

```
TIMER_Config timerCfg1 = {
0x00000311, /* Control Register (CTL) */
0x0000000F, /* Period Register (PRD) */
0x00000000 /* Counter Register (CNT) */
};
```

The Timer1 output is set for clock mode and period value is 15. The ADS8411 requires a quiet zone of 50 ns before and 40 ns after CONVST start falling edge. It takes the EDMA controller about 130 ns after the BUSY rising edge to generate a read pulse (see Figure 5). For this reason, it is possible to use BUSY to trigger conversions and the read at full speed. The disadvantage to this solution is that the first conversion result is invalid because the EDMA controller reads the converter before the first conversion cycle is complete. Generally, during conversion time, the IC should be kept free of switch currents that could disrupt the ground reference plane of the converter.

To change the timer frequency, open file config.cdb in Code Composer Studio and expand as shown in Figure 3. Right-click timerCfg1, select properties, and select counter control tab.

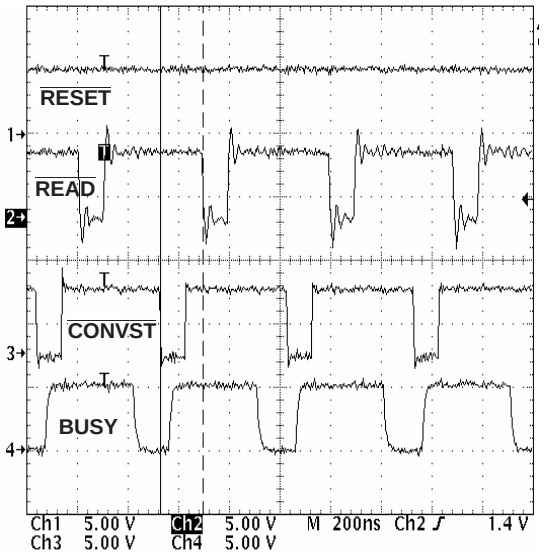

Figure 5. Screenshot Showing General Timing

4 Conclusion

This application report presents a solution for interfacing the ADS8401 and ADS8411 converters to the ‘C6713 DSP. The ADS8401EVM 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 from Texas Instruments. The software solution involves using the DSP/BIOS and configuration tool (i.e., config.cdb file) to visually set up the EDMA controller, timer, and interrupts. The EDMA controller and Timer1 were used to trigger conversion cycles at 2 MHz. The ping-pong buffering scheme was employed using the linking feature of the EDMA controller. In this application, the EDMA halts after collecting 4096 samples. The user may change the program to continuously ping pong between the two buffers by changing the link handle in edmaCfgChan6A to hEdmaTbl6, instead of hEdmaTbl0.

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. ADS8401 data sheet (SLAS376)
5. ADS8411 data sheet (SLAS369)
6. TMS320C6713 data sheet (SPRS186)
7. ADS8401/ADS8411EVM User's Guide (SLAU114)
8. 5-6K Interface Board EVM User's Guide (SLAU104)

MAIN.C

A.1 MAIN.C

```
/*****************************************************************/
/* File: main.c */
/* Description: Program for interfacing ADS8411 */
/* to a 'C6713 DSK. Program uses the Timer1 to trigger EDMA a */
/* transfer which causes a convst# pulse, at about 2 MHz. */
/* The BUSY signal is used to trigger EDMA channel 6 to */
/* read from the A/D and store data into pingBuf[] and pongBuf[] data*/
/* arrays. */
/* Once BLOCK_SZ of data is captured, the EDMA will interrupt CPU. */
/* CPU will then halt EDMA channels and timer1 after the second input. */
The program collects 4096 samples. EDMA triggers of the rising edge*/
/* BUSY. The first point in the pingBuf should be discarded, */
/* because it is a read before the first conversion cycle is complete. */
/* 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 => 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"

/* 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(pingBuf,BLOCK_SZ);
unsigned short pingBuf[BLOCK_SZ]; /*ping buffer*/
#pragma DATA_ALIGN(pongBuf,BLOCK_SZ);
unsigned short pongBuf[BLOCK_SZ]; /*pong buffer*/
unsigned short temp, n=0;
void main(void)
{

int i;
/* initialize the EMIF and A/D*/
init_dsk();
init_adc();
/*Initialize data buffers */
for (i=0; i<=BLOCK_SZ; i++) {
 pingBuf[i]=0x0000;
 pongBuf[i]=0x0000;
}
```

```
/*****************************************************************/
/* Enable the EDMA controller interrupt */
 IRQ_reset(IRQ_EVT_EDMAINT); /*Reset EDMA interrupt */
 IRQ_enable(IRQ_EVT_EDMAINT);
 EDMA_intDisable(TCCINTNUM6); /*Disable EDMA 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_clearChannel(hEdmaCha6);
 EDMA_enableChannel(hEdmaCha2); /*Enable EMDA channel 2 -CONVST#*/
 EDMA_enableChannel(hEdmaCha6); /*Enable EDMA channel 6 -RD# */
/*Timer1 was initialized by DSP/BIOS before entering main.c */
/*Only need to enable it here. */
 TIMER_start(hTimer1); /*Start A/D CONVST# trigger */
 /*Timer Period Value = (225e6/4)/fs*/
 /*Go sample at 2MSPS*/
}
```

Functions.C

B.1 Functions.C

```
/*****************************************************************/
/* File: functions.c */
/* Description: Functions for interfacing ADS8411 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 => 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>
/*Function Prototypes*/
void swiEnablePrephFunc();
extern unsigned short temp, ad_buffer[BLOCK_SZ]; /* Raw buffer for AD data */
 int pingpong=0; /*=1 pingbuffer full*/
/*****************************************************************/
/* 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 ADS8411. */
```

```
 /* 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 = *ADS8411_RESETZ; /*Reset ADC*/
 for (i=0; i<=1; i++){};
 temp = *ADS8411_CONVSTZ; /*Discard the first three conversions*/
 for (i=0; i<=10; i++){};
 temp = *ADS8411_RD;
 temp = *ADS8411_CONVSTZ;
 for (i=0; i<=10; i++){};
 temp = *ADS8411_RD;
 temp = *ADS8411_CONVSTZ;
 for (i=0; i<=10; i++){};
 temp = *ADS8411_RD;
 }
/*************************************************************/
 /*hwiDMA_isr(): */
 /* Hardware Interrupt Function disables EDMA channels and */
 /* Timer1, Then post software interrupt. */
 /*************************************************************/
 void hwiDMA_isr()
 {
if (pingpong==1){
 EDMA_disableChannel(hEdmaCha2); /*Disable EMDA channel 2-CONVST#*/
 EDMA_disableChannel(hEdmaCha6); /*Disable EMDA channel 2 -RD#*/
 TIMER_pause(hTimer1);
 }
 pingpong =!pingpong;
 IRQ_reset(IRQ_EVT_EDMAINT); /*Reset EDMA interrupt */
 IRQ_enable(IRQ_EVT_EDMAINT);
EDMA_intDisable(TCCINTNUM6); /*Disable EDMA interrupt */
 EDMA_intClear(TCCINTNUM6); /*Clear EDMA interrupt */
 EDMA_intEnable(TCCINTNUM6); /*Enable EDMA interrupt */
/*Uncomment next line if continuous capture of BLOCK_SZ */
 /*of data */
 // SWI_post(&swiEnablePreph);
}
/*************************************************************/
 /*swiEnablePrephFunc: */
 /* Software Interrupt Function configures EDMAC2 and */
/* EDMAC6, enables respective EDMA channels and Timer1 */
 /*************************************************************/
 void swiEnablePrephFunc()
 {
 // temp = *ADS8411_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 */
}
```

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