Interfacing the ADS8383 to the TMS320C6711 DSP

Lijoy Philipose *Data Acquisition Applications*

ABSTRACT

This application note presents a software and hardware interface of the ADS8383 18-bit 500 kHz analog-to-digital converter to the TMS320C6711 DSP. The hardware platform used to develop this application is ADS8383EVM and TMS320C6711 DSK. The software developed reads 1024 blocks of samples continuously from ADS8383. In an effort to reduce development time, the source code is available on TI's website. Project collateral discussed in this application note can be downloaded from the following URL: http://www.ti.com/lit/zip/SLAA174.

Contents

List of Figures

3 Introduction

The ADS8383 is a single channel, 500KSPS, 18-bit analog-to-digital converter. The converter can easily interface to the TMS320C6711 digital signal processor (C6711 DSP™). For system development the TMS320C6711 DSP starter kit (DSK) and ADS8383EVM was used. The program written uses the EDMA channels and timer to trigger, capture and store blocks of data from the converter. A soft copy of the software for this application note is available for download from TI's website.

4 Hardware

The hardware platform employs the TMS320C6711 DSK and ADS8383EVM and some discrete logic.

4.1 TMS320C6711 DSK

The TMS320C6711 DSP starter kit (DSK) not only provides an introduction to C6000 technology, but is powerful enough to use for fast development of networking, communications, imaging and other applications like data acquisition. See TI's website (www.ti.com) for more information on the C6711 DSK.

4.2 ADS8383EVM

The ADS8383 evaluation module is an easy way to test both the functional and dynamic performance of this 18-bit analog-to-digital converter. The evaluation module includes only those circuits essential to showing the performance of the converter, reference, power, and input circuits. The digital inputs and outputs are buffered. The analog signal can be applied via standard 0.1inch IDC header/socket or via SMA connector. The buffered data bus is available via 0.1inch IDC header/socket. The ADS8383 control input is also made available standard 0.1-inch IDC header/socket. Therefore this EVM can be plugged into any development system for rapid prototyping. For information on this product, search www.ti.com for the ADS8383EVM.

4.3 Hardware Interface

One ADS8383 was mapped into memory space CE2, so the $\overline{CE2}$ signal was used as $\overline{CS}$ for the A/D device. The read and convert start signals were generated by using a 2−4 decoder mapped into CE2. Performing a read operation to CE2 address space generates the $\overline{RD}$ and $\overline{CONVST}$ pulses. In this case a reading from address 0xA0028000 generates a read pulse and reading from 0xA002C000 generates a conversion start pulse. The C6711 has a 32-bit data bus; therefore BYTE was tied low allowing for 18-bit bus operation. The BUSY signal is first inverted and then applied to the external interrupt pin 6. The inversion of the BUSY is necessary because the EDMA interrupt controller recognizes only rising edges of external interrupts. The inversion is not necessary if the CPU is used to service A/D interrupts, because it can be programmed to trigger on a rising or falling edge. Finally, the data bus is mapped LSB-to-LSB. The hardware connections are shown in Figure 1.

C6711 DSP and C6000 DSP are trademarks of Texas Instruments.

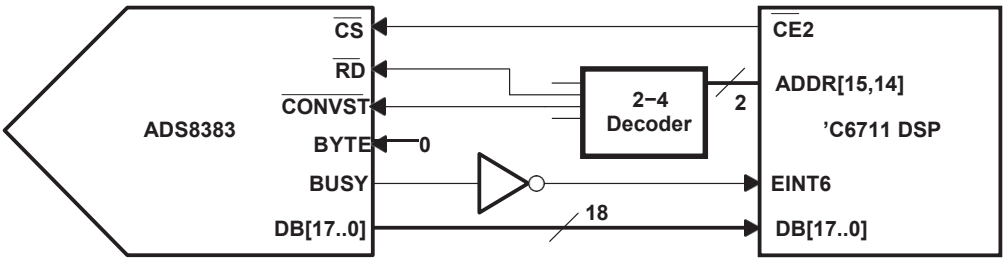

Figure 1. Hardware Connection

5 Software Interface

The program utilizes two EDMA channels to read from 0xA0028000 and 0xA002C000 to generate the conversion start and read signals to the converter. The first EDMA channel is synchronized by Timer0, generating a periodic conversion start signal. The second EDMA channel is triggered by an external interrupt 6 event, it generates the read signal. Once all 1024 samples have been captured and stored in the array *ad_buffer*, the EDMA controller generates an interrupt to the DSP. The interrupt service routine disables all activity with the converter, processes the data and posts a software interrupt. The function associated with that software interrupt copies the data from *ad_buffer*, clears the high 14 bits of the 32-bit word and places the data into an array called *Fbuffer*. The function then posts another software interrupt that enables the EDMA channels and timer0 to capture another 1024 samples. See Figure 33 for software flow and how the individual functions are called.

The software interface is written entirely in C using Code Composer Studio™ (CCS) IDE using CSL and DSP/BIOS. This application report assumes readers have a basic understanding of DSP/BIOS and CSL as found in the help system of CCS. If not, it is recommended they go through the tutorials in Code Composer Studio before proceeding further. The program is written using the configuration tool.

The following sections describe how the configuration tool is used for setting the EDMA, timer, hardware interrupts, software interrupts, and LOG objects.

5.1 EDMA Setup

1. Expand the chip support library (CSL) module in the configuration tool. Right click and insert two edmaCfg objects. Rename those objects to edmaCfgChan2 and edmaCfgChan6 (see Figure 2).
2. EDMA channel 2 is used to trigger a new conversion. Right click on edmaCfgChan2. In the General tab, you may choose to enter a comment saying it is a start conversion channel. In the operation mode tab select the options shown in Figure 3. Enabling Frame Sync causes an EDMA transfer when a Timer0 event occurs. In this case it is a 16-bit read into a variable called temp.

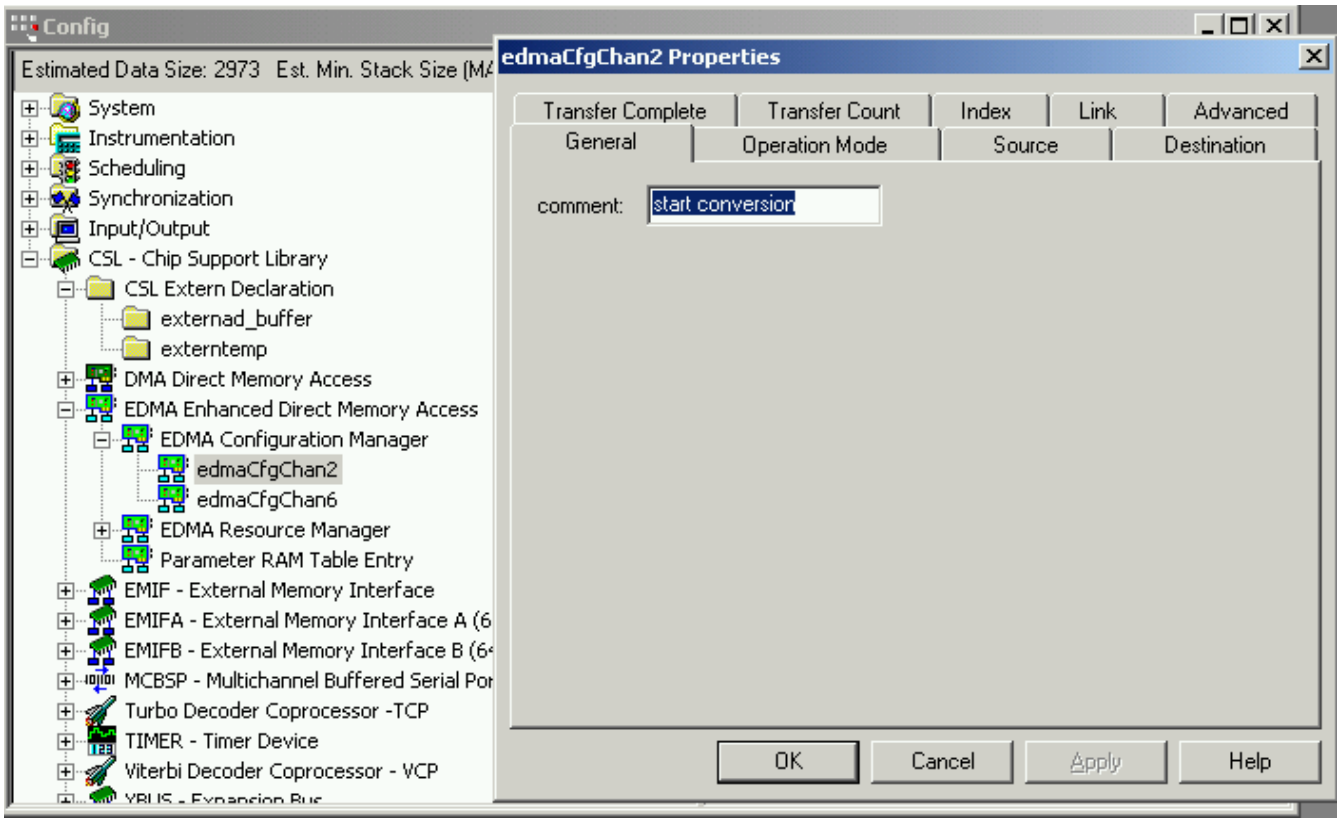

Figure 2. EDMA Chan2 General Tab

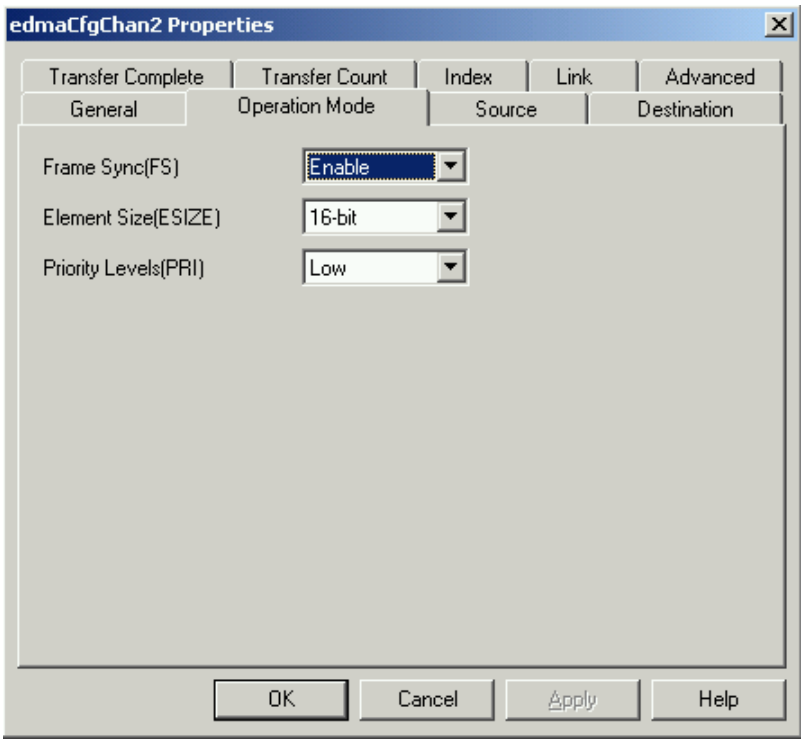

Figure 3. EDMA Chan2 Operation Mode Tab Options

3. In the Source tab, select the options as shown in Figure 4. The 16-bit reads are from address 0xA002C000. Toggling these particular address lines causes the decoder to generate a $\overline{\text{CONVST}}$ pulse.
4. In Destination tab, select the options shown in Figure 5. The 16-bit value read into by the EDMA channel is stored in the temp variable. This object is externally declared. The value on the data bus is discharged, because it is a not read from the converter.

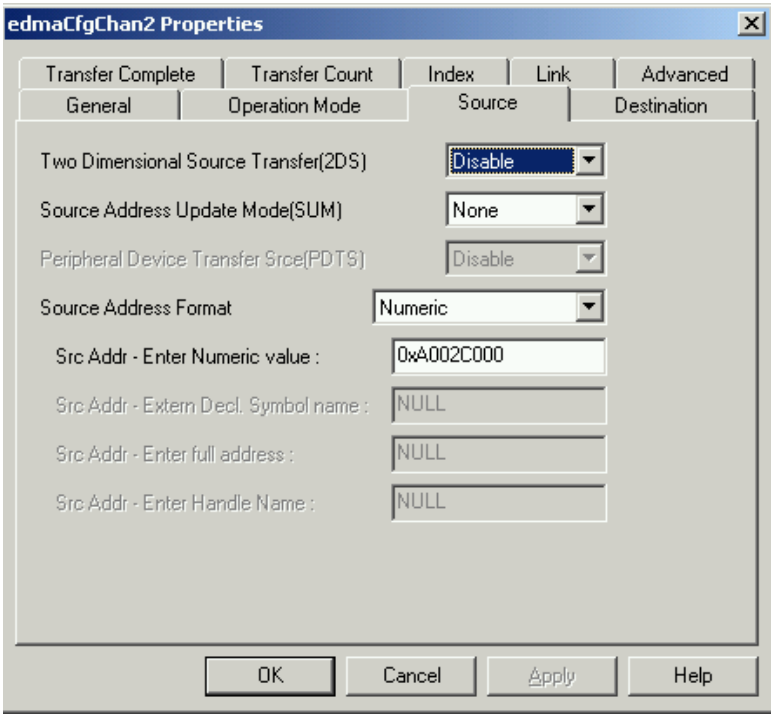

Figure 4. EDMA Chan2 Source Tab Options

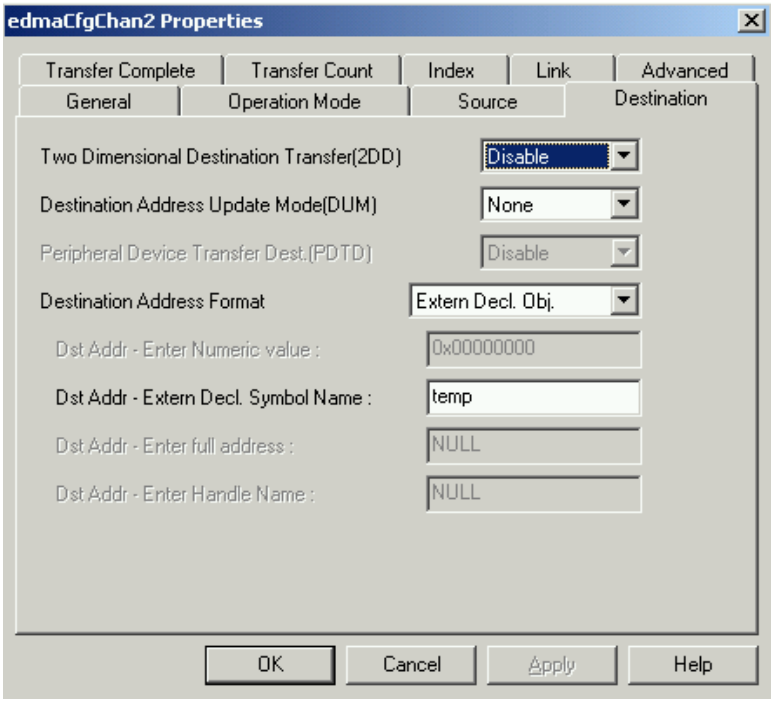

Figure 5. EDMA Chan 2 Destination Tab Options

5. In the Transfer Complete tab, choose options shown in Figure 6. Transfer complete interrupt is disabled.
6. In the Transfer Count tab, choose options shown in Figure 7. Frame count is 0x0100.

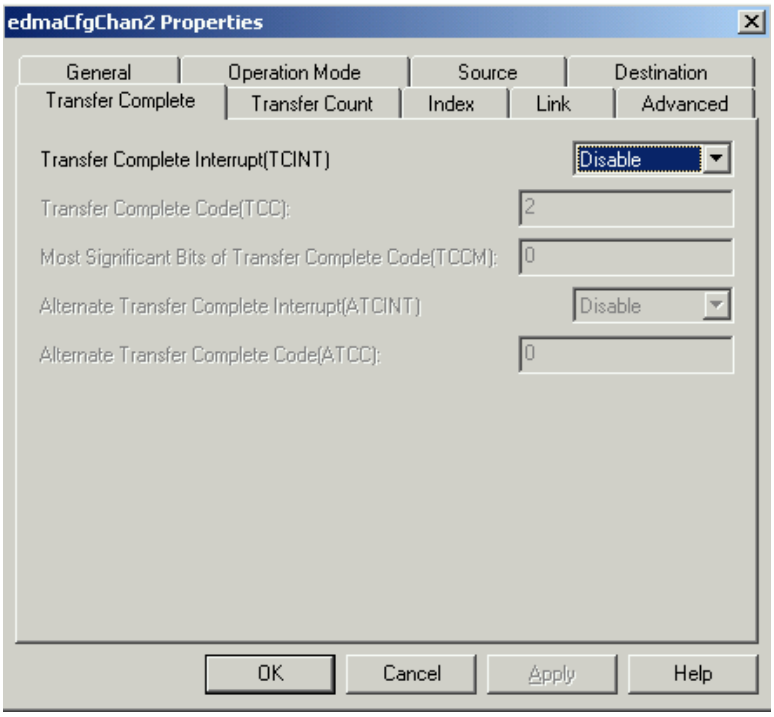

Figure 6. EDMA Chan2 Transfer Complete Tab Options

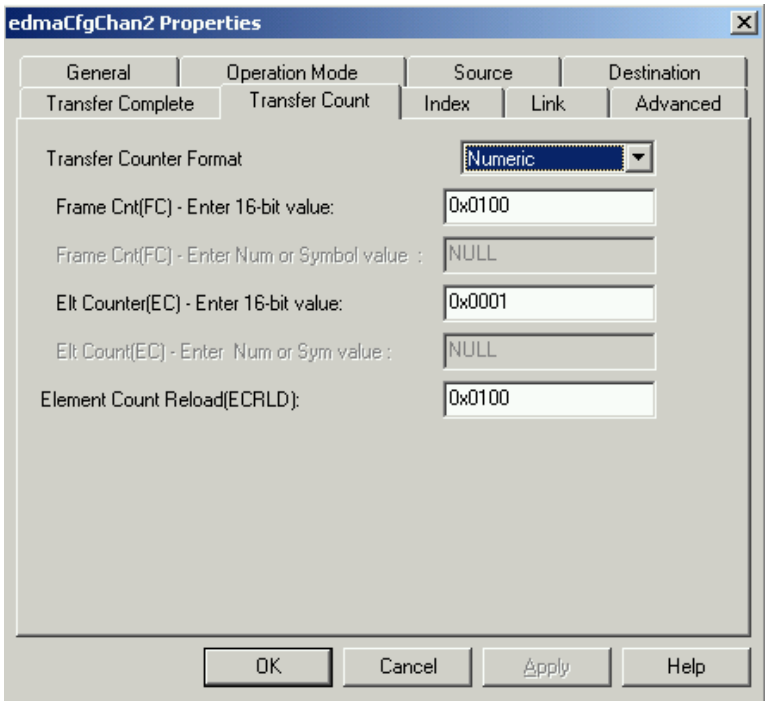

Figure 7. EDMA Chan2 Transfer Count Tab Options

7. In the Index and Link tabs, choose options shown in Figure 8 and Figure 9.

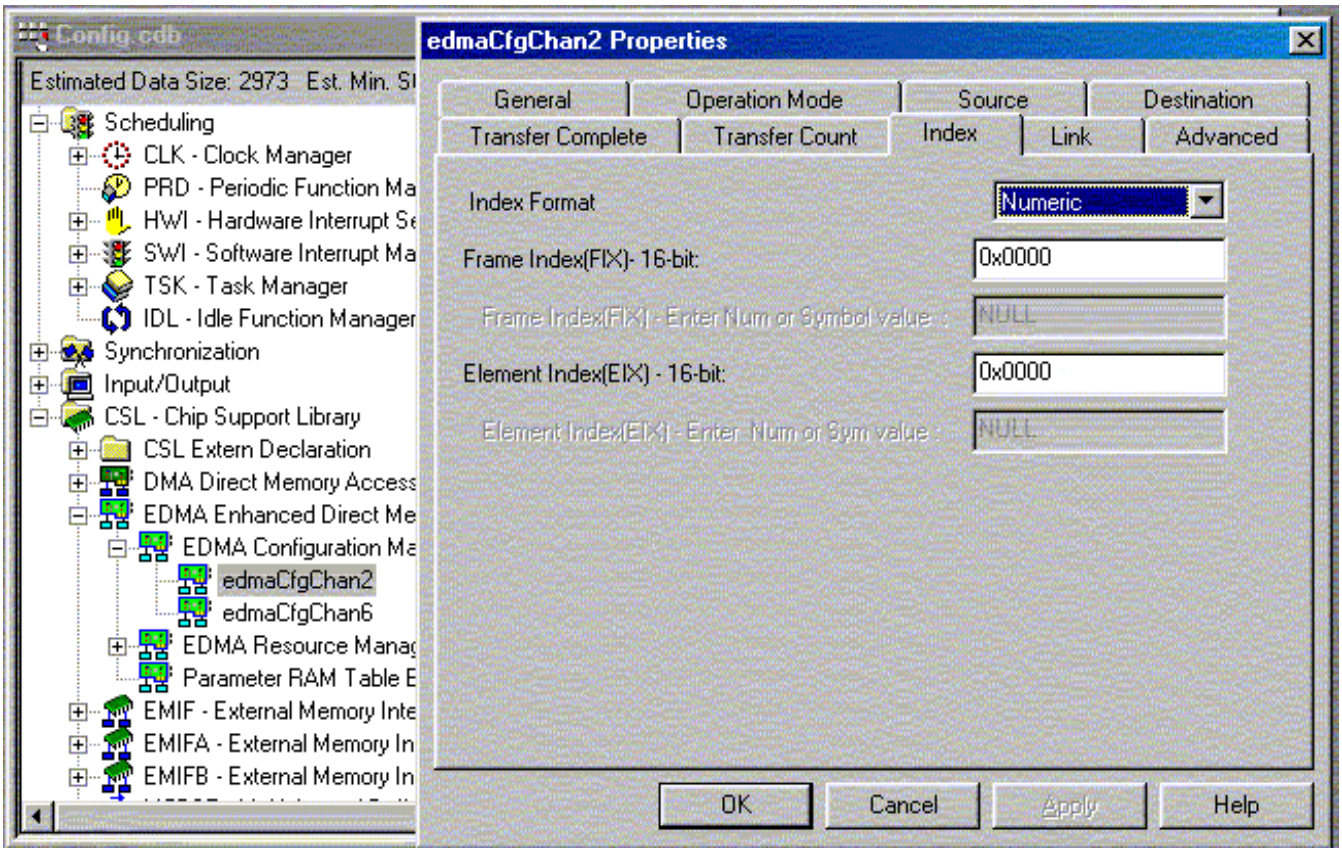

Figure 8. EDMA Chan2 Transfer Index Tab Options

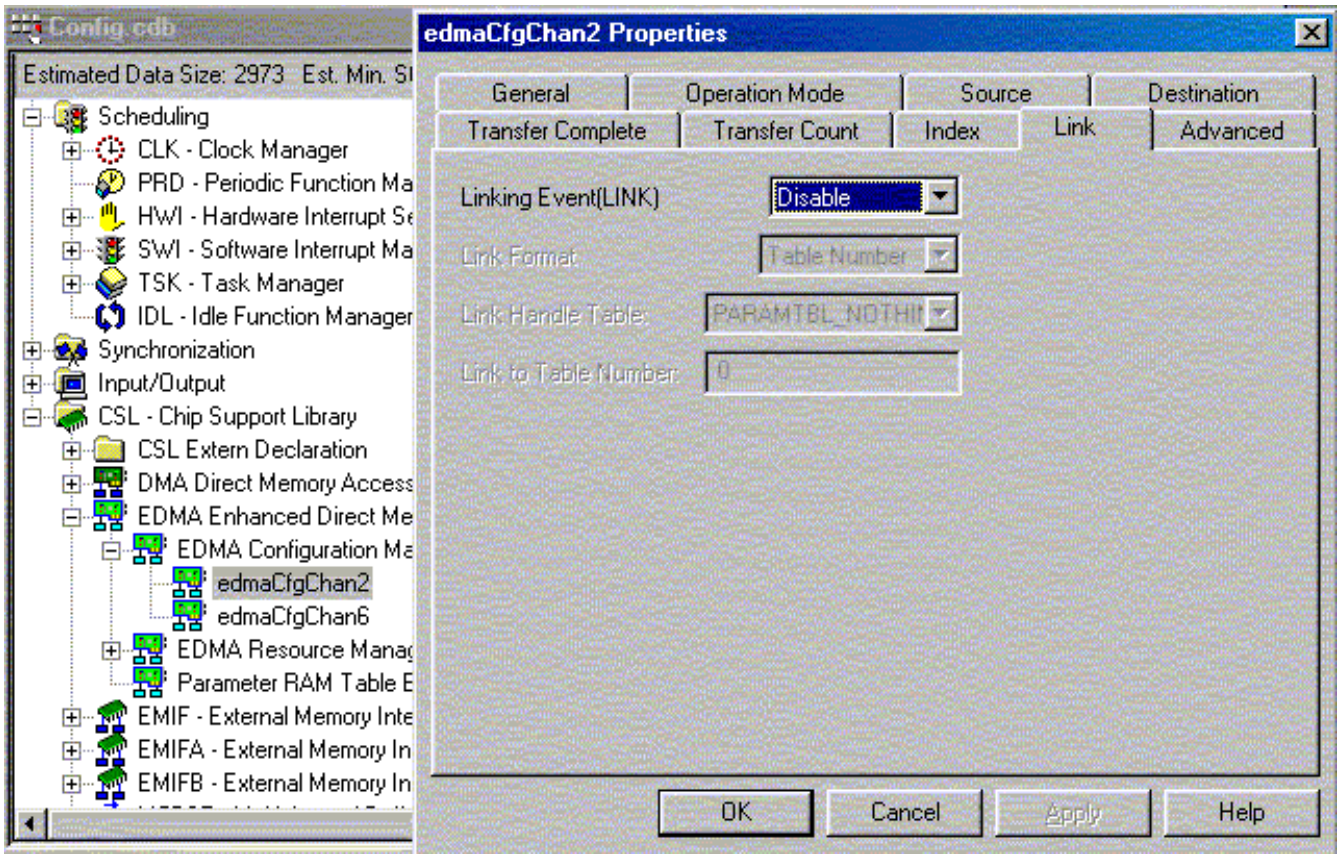

Figure 9. EDMA Chan2 Link Tab

8. The Advanced tab should look like Figure 10. $\overline{\text{CONVST}}$ pulse is mapped in address 0xA002C000. The 16-bit data is stored in an externally declared object *temp*.
9. To map this configuration object to the resource, select and expand the EDMA resource manager. Right click on EDMA_channel1_TINTO and select properties. In the general tab, select those options shown in Figure 11. Create an EDMA channel handle called *hEdmaTint0* and enable pre-initialization of EDMA channel 1 registers. In the main program, the user needs to enable the channel. The initialize is done before entering into the main() function.

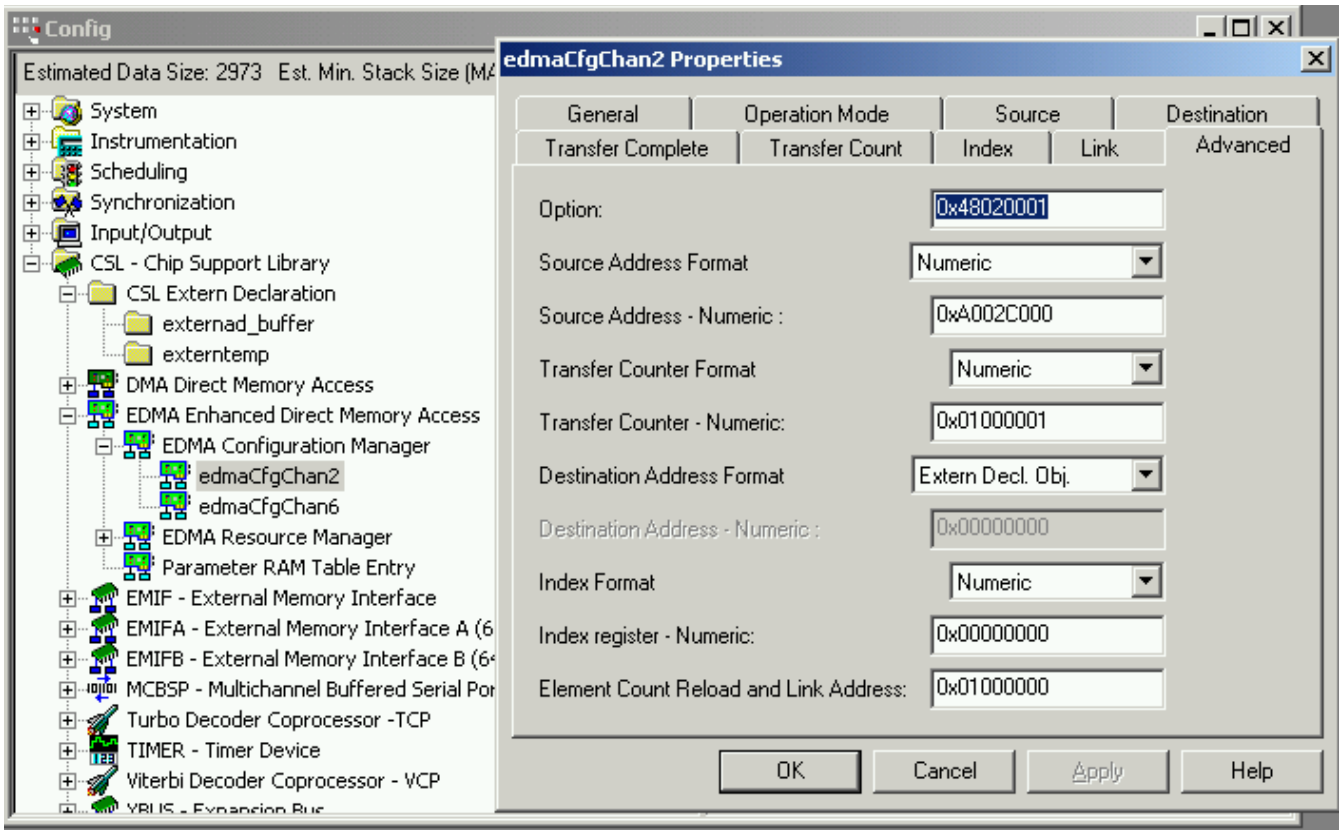

Figure 10. EDMA Chan2 Advanced Tab Options

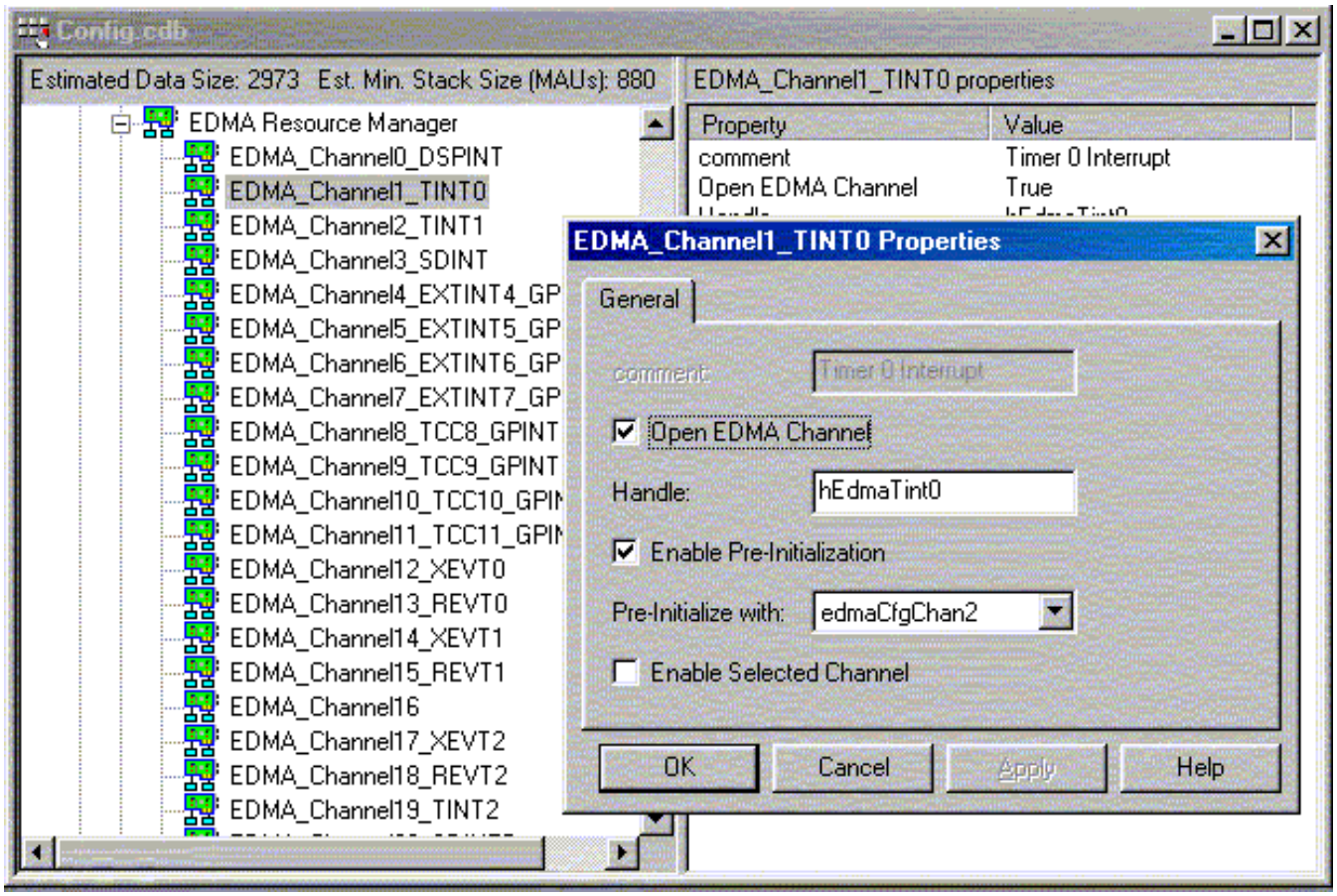

Figure 11. EDMA Chan6 General Tab Options

10. Right click on edmaCfgChan6 object and select properties. In the General tab, the user may choose to write that this object is used to read data from the converter.
11. In the Operation Mode tab, select options shown in Figure 13. Enable Frame sync, this triggers a read operation whenever an interrupt occurs signaling the end of a conversion. In this application, the BUSY pulse is inverted before being applied to the External interrupt 6 pin. This inversion is necessary before the EDMA recognizes only rising edges and falling edges. Remember that BUSY goes high when conversion begins, and returns low when conversion is complete.

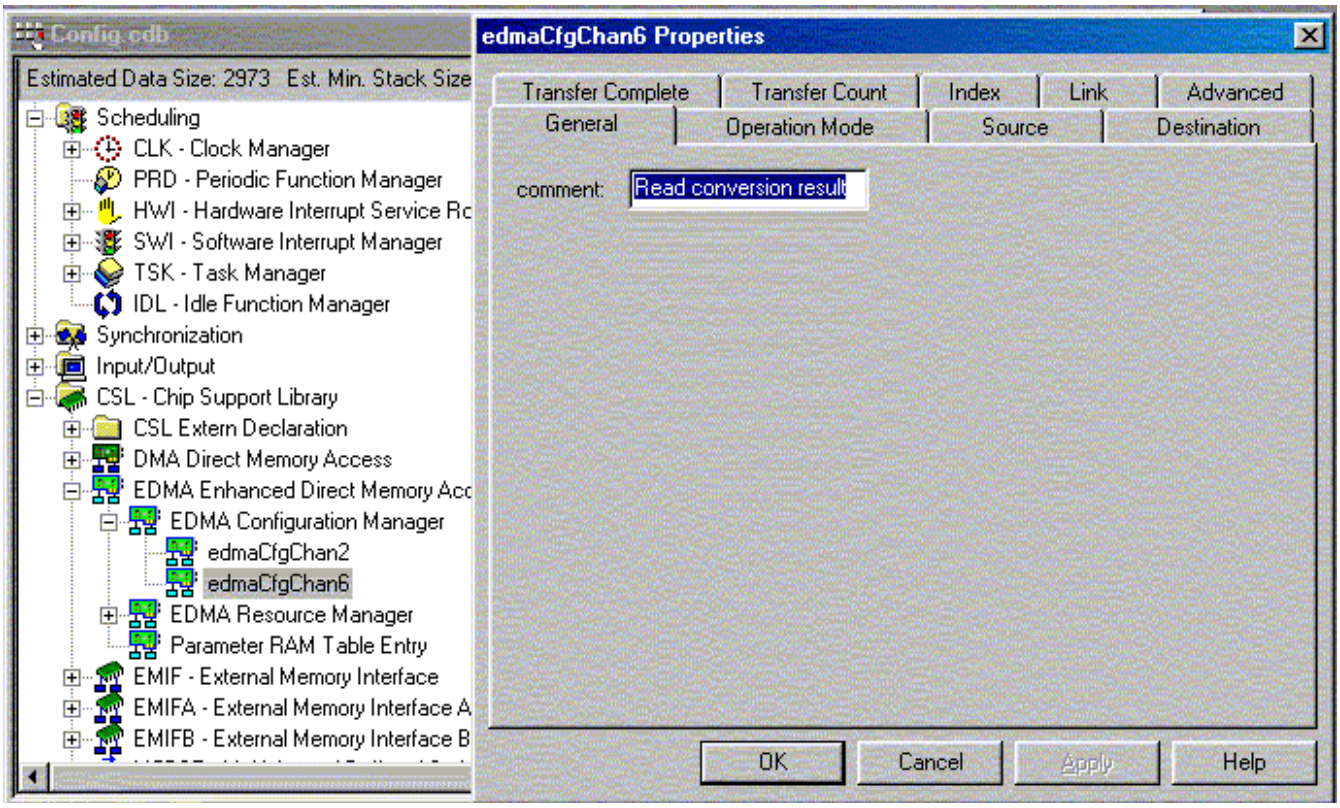

Figure 12. EDMA Chan6 General Tab

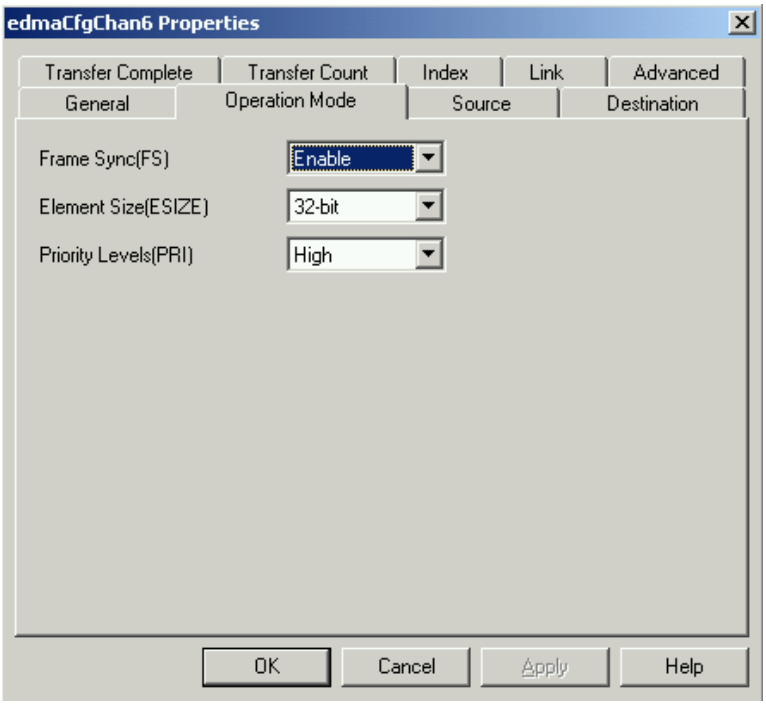

Figure 13. EDMA Chan6 Operation Mode Tab Options

12. In the Source tab, set the options shown in Figure 14. A read pulse is generated by the decoder when accessing 0xA0028000. So the source address is 0xA0028000. Disable two-dimensional source transfer and source address update.
13. In the Destination tab, select the options shown in Figure 15. Disable the two dimensional Source transfer and select increment destination address. The data read from the converter is stored in an externally declared array called *ad_buffer*. This is a 32-bit read and the converter is mapped DSP LSB to A/D LSB, therefore the top 14-bits of the 32-bit word should be masked out.

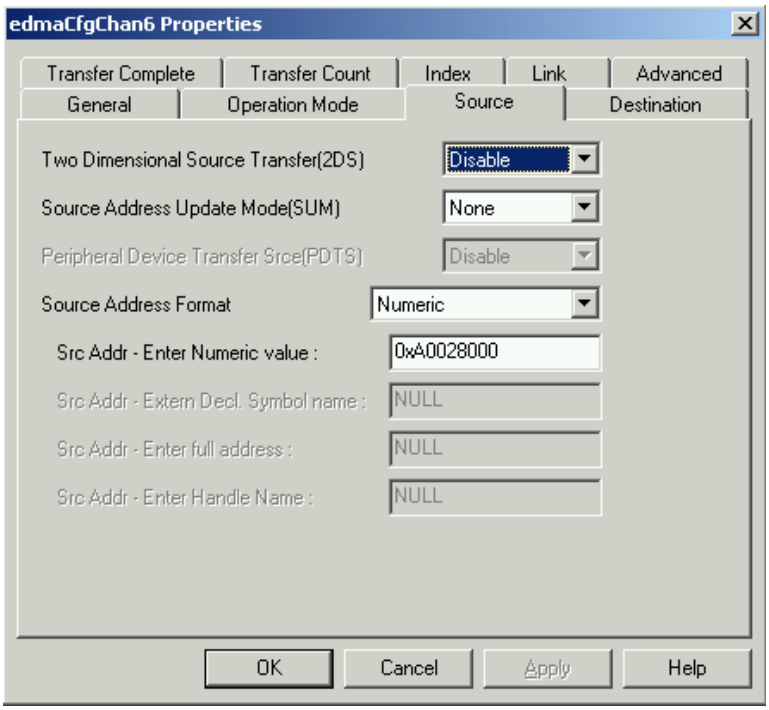

Figure 14. EDMA Chan6 Source Tab Options

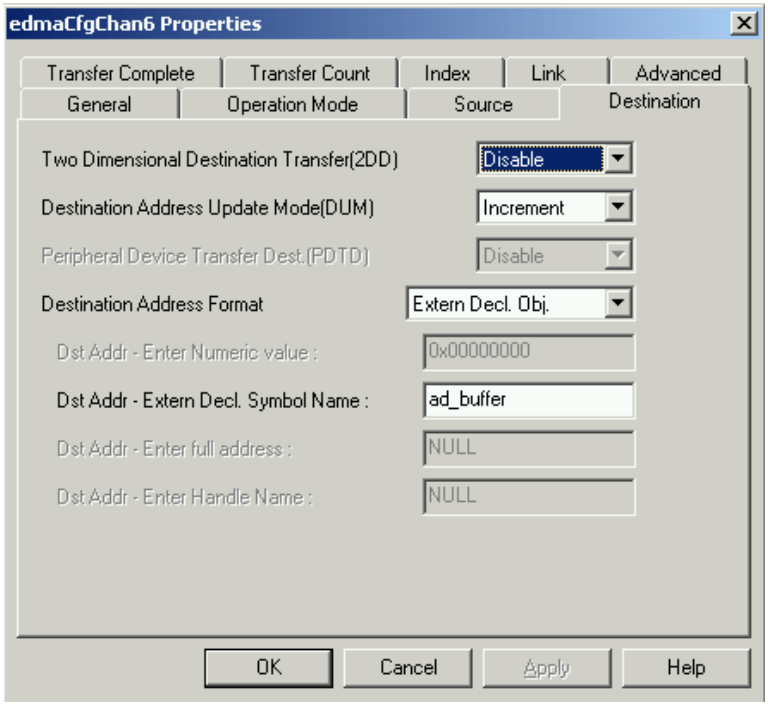

Figure 15. EDMA Chan6 Destination Tab Options

14. In the Transfer Complete tab select the options as shown in Figure 16. Enable Transfer Complete Interrupt to interrupt the DSP. The controller interrupts the DSP, and transfers complete code 6 when all 0x0400 samples have been read and stored.
15. In the Transfer Count tab, select options shown in Figure 17. Frame Count is set to 0x0400, indicating a total transfer of 1024 samples.

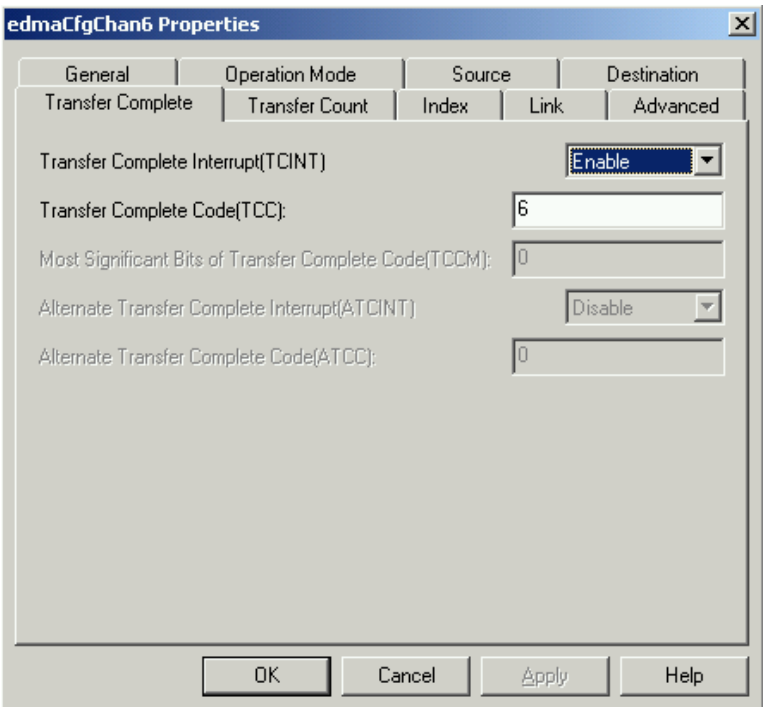

Figure 16. EDMA Chan6 Transfer Complete Tab Options

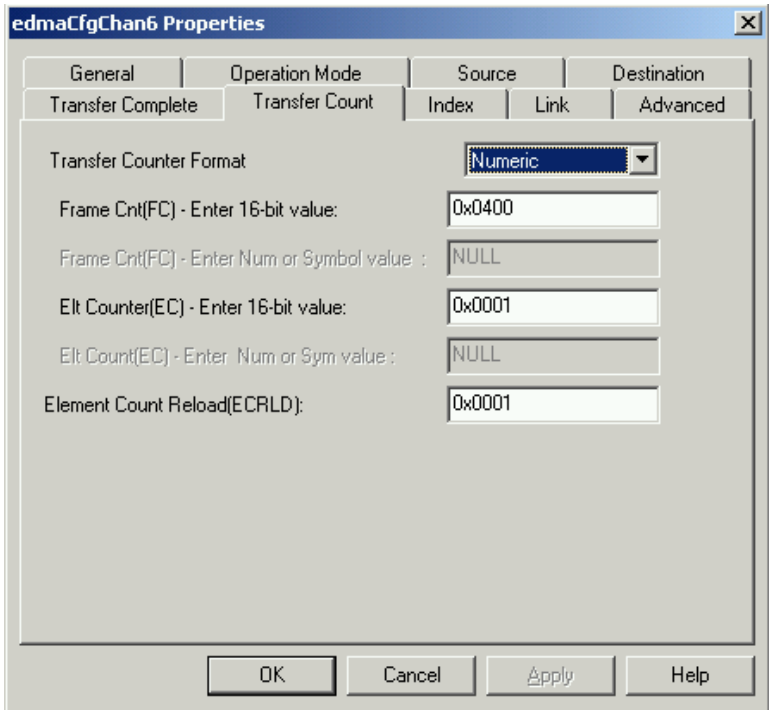

Figure 17. EDMA Chan6 Transfer Count Tab Options

16. In the Index and Link Tabs, select options shown in Figure 18 and Figure 19.

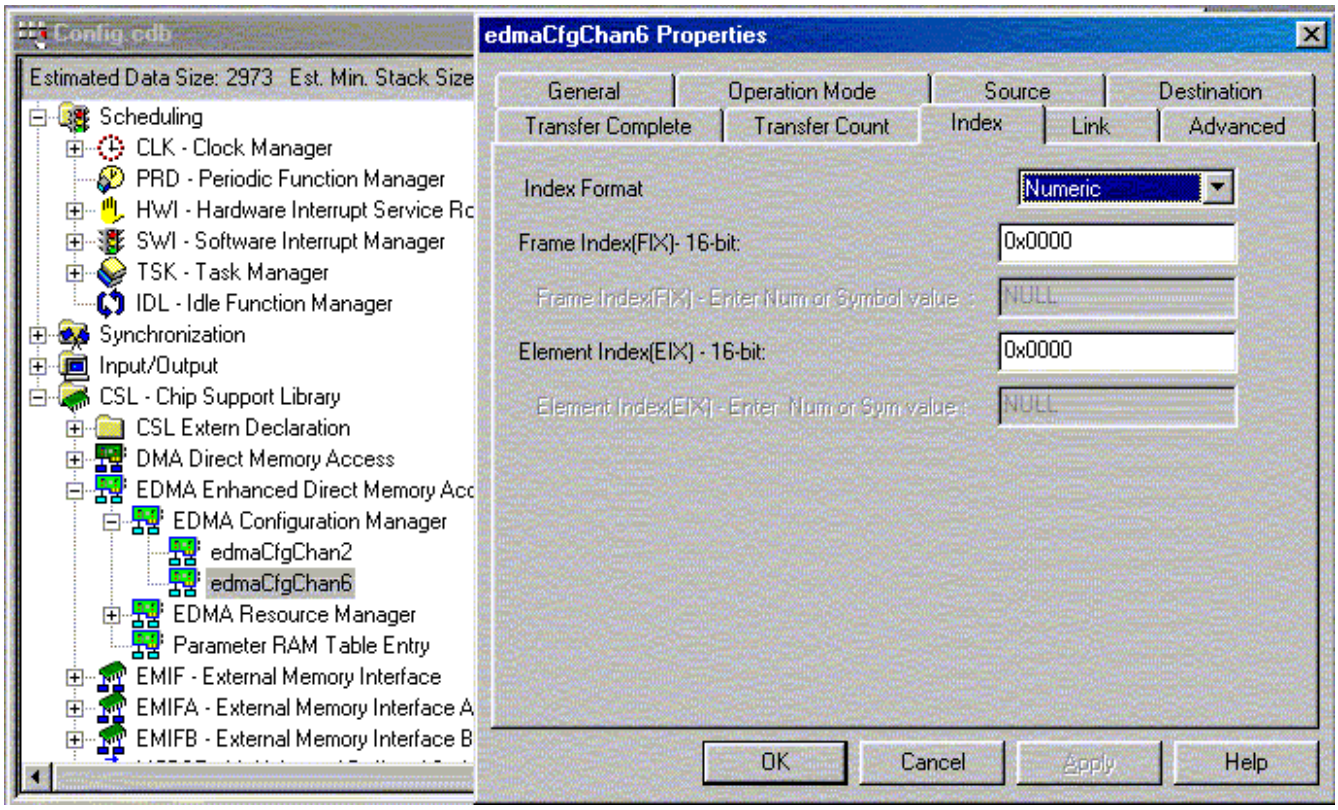

Figure 18. EDMA Chan6 Index Tab Options

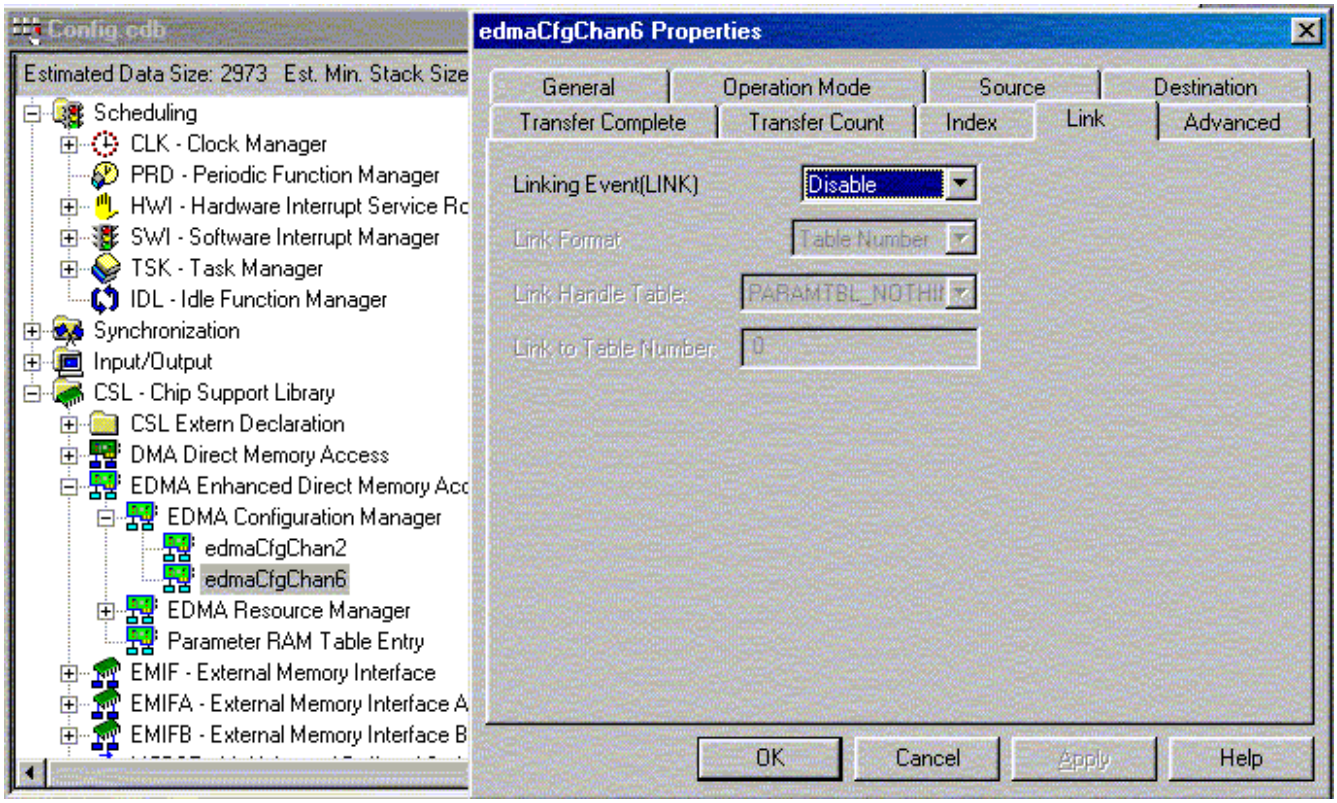

Figure 19. EDMA Chan6 Link Tab Options

17. In the Advanced tab, set the options shown in Figure 20. Read 1024 samples from the A/D converter for which the Read signal is mapped to 0xA0028000. When all 1024 samples have been stored, interrupt the DSP.
18. The configuration object now needs to be mapped into the appropriate resource. In the EDMA Resource Manager, right click on EDMA_Channel6_EXTINT6_GPINT6 and select properties. In the General tab, select the options shown in Figure 21. Create hEdmaCha6 handle and preinitialize EDMA channel 6 registers with edmaCfgChan6. Channel initialization completes before entering the main() function. In the main function the channel needs to be enabled.

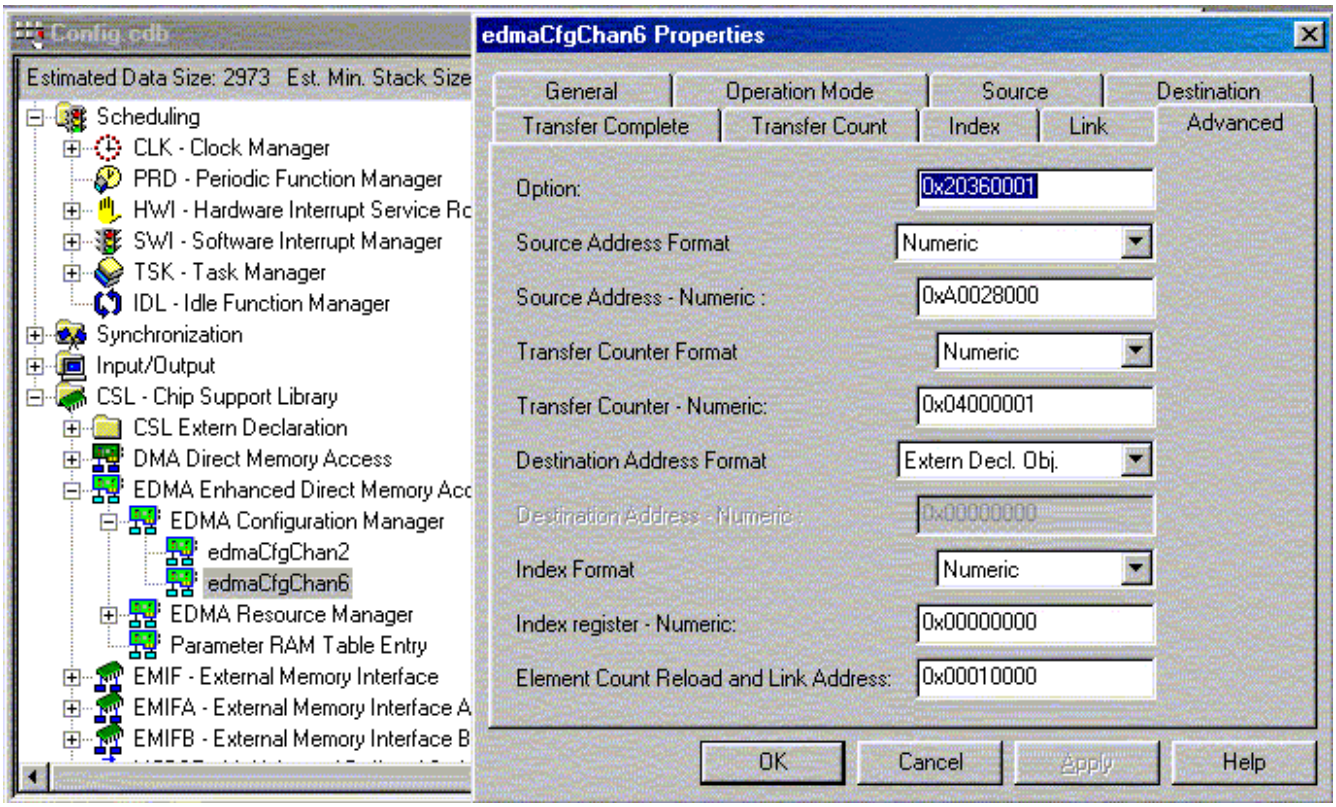

Figure 20. EDMA Chan6 Advanced Tab Options

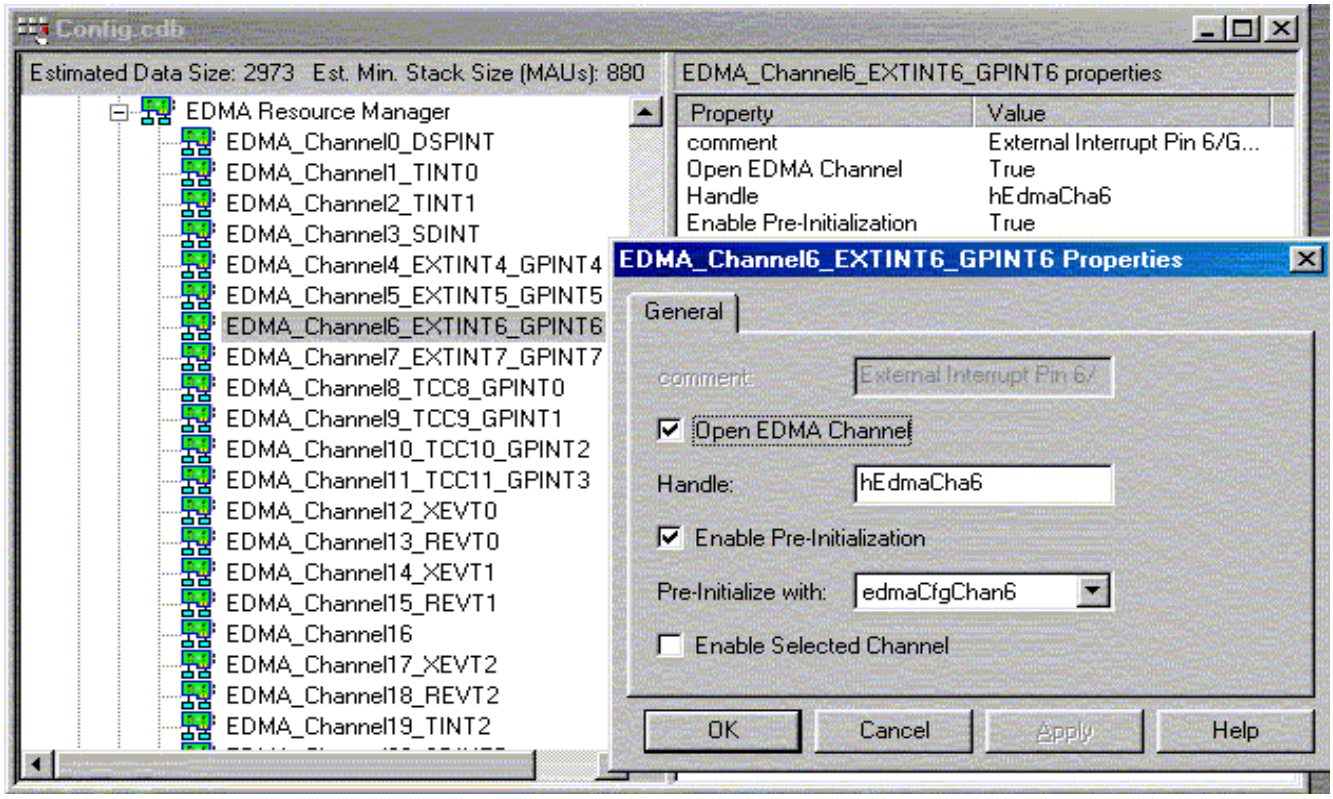

Figure 21. EDMA Chan6 EXTINT6 Properties Tab

5.2 Timer Setup

The Timer setup can also be done via the configuration tool. Expand the CSL−Chip Support Library section, and expand Timer–Timer Device module. Right click on Timer Configuration Manager and select Insert timerCfg. Rename timerCfg to timerCfg0 to map it into Timer0.

1. Right click on timerCfg0 and go to the Clock Control tab. Select options shown in Figure 22. Input Clock source is CPU clock divided by 4. Select Clock mode.
2. In the Pin Control tab, select options as shown in Figure 23. Function of TOUT is Timer Output and TOUT is uninverted.

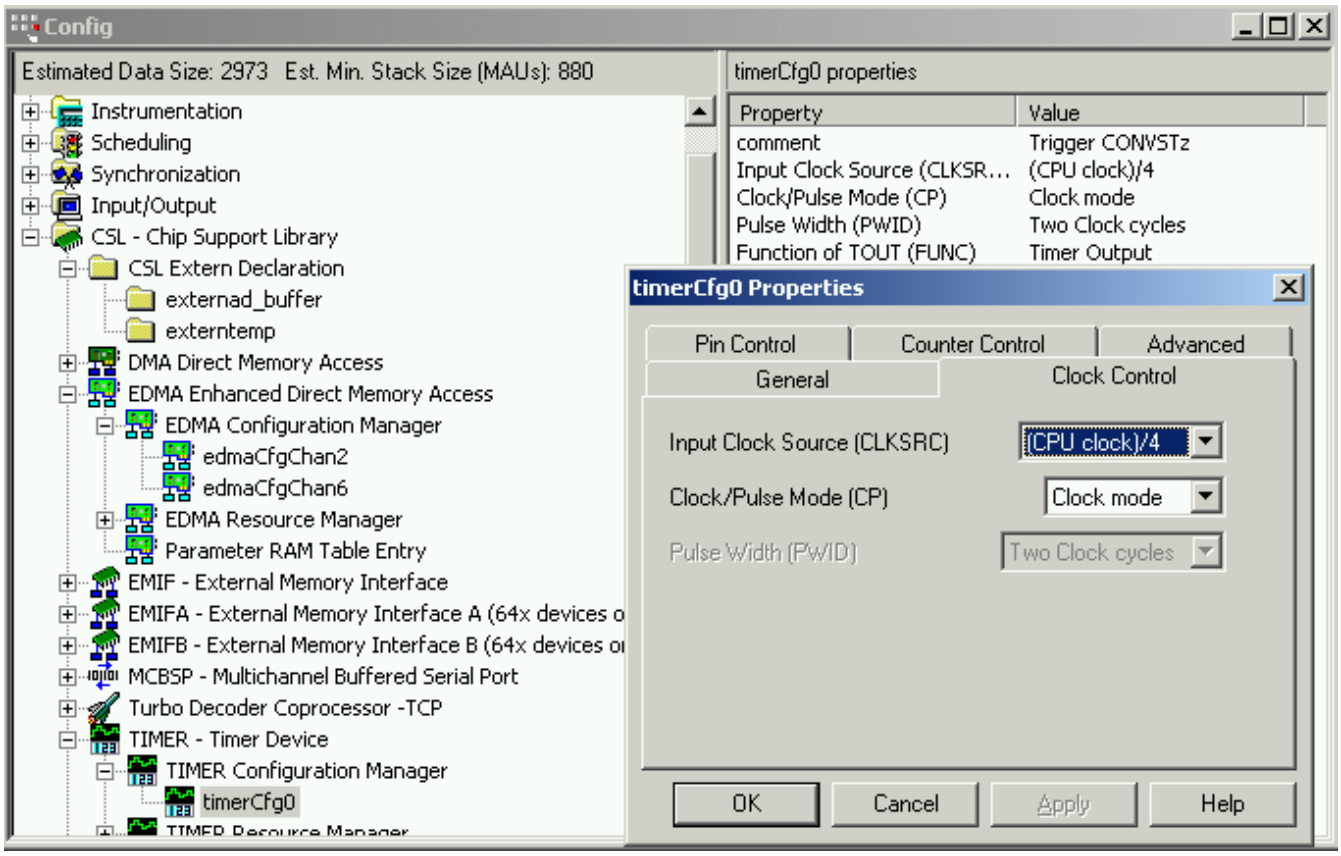

Figure 22. Timer0 Cfg. Clock Control Tab Options

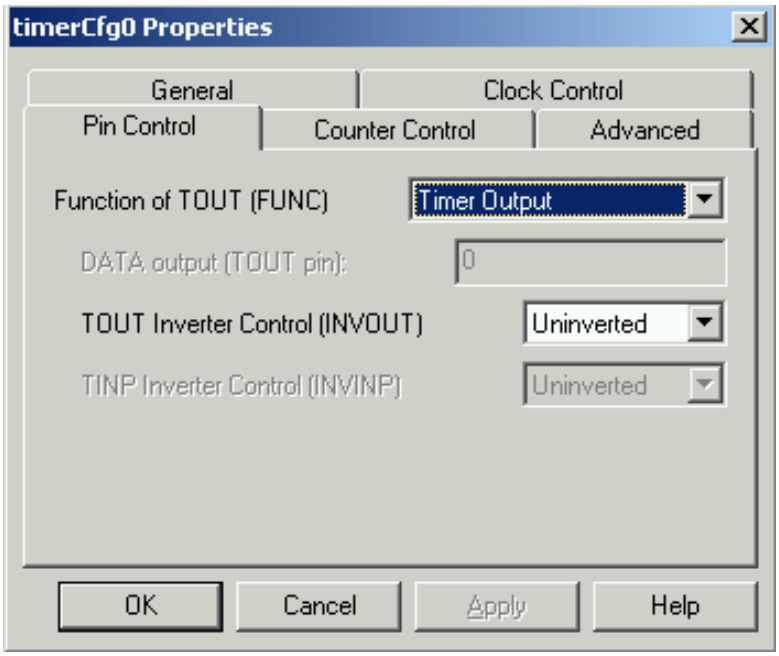

Figure 23. Timer0 Cfg. Pin Control Tab Options

3. In the Counter Control tab, type the value shown in Figure 24. Type 0x26 into the Period Value field. The frequency of the timer is set by the formula below:

$$\text{Frequency} = \frac{[f(\text{clock source})]}{(2 \times \text{timer period register})}$$

In this application note, the C6711 DSP is clocked at 150MHz. In the previous tab, clock divide-by-4 option was chosen as the source for timer0. Therefore the sampling frequency is set for

$$\text{Frequency} = \frac{(150e6/4)}{38} = 493 \text{ kHz.}$$

The Advanced tab should read as shown in Figure 25.

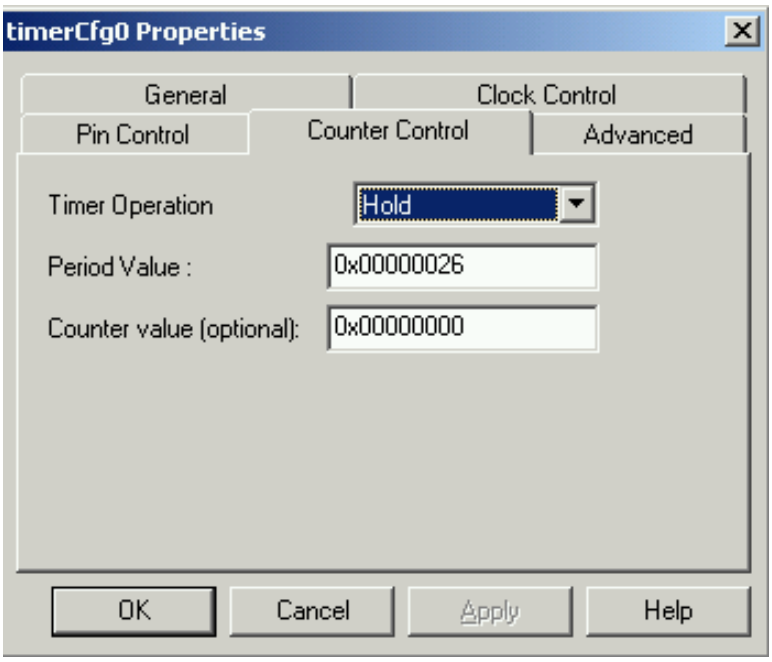

Figure 24. Timer0 Cfg. Counter Control Tab Options

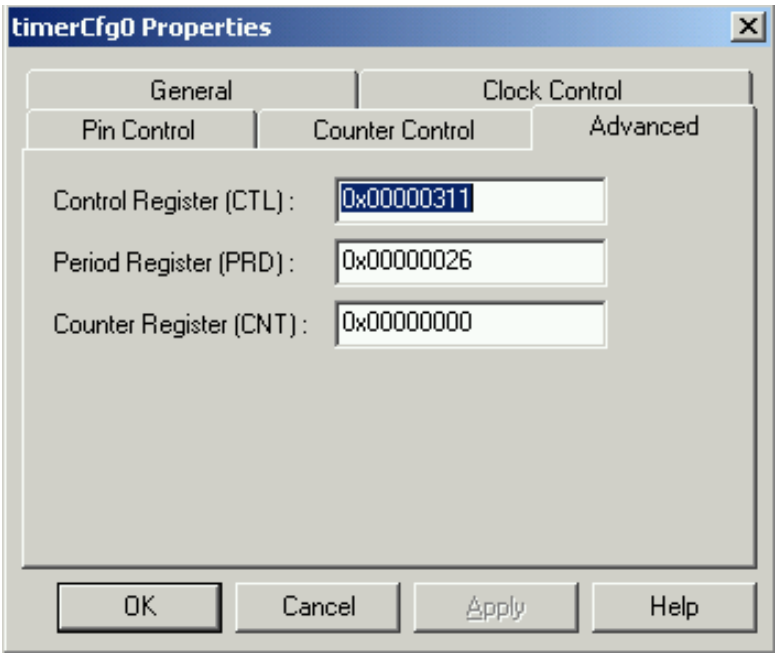

Figure 25. Timer0 Cfg. Advanced Tab Options

4. Now that the configuration object has been set, it must be mapped into the appropriate device. Expand the Timer Resource Manager and right click on Timer_Device0. Select options as shown in Figure 26. Enable timer0 registers to be preinitialized and open a hTimer0 handle.

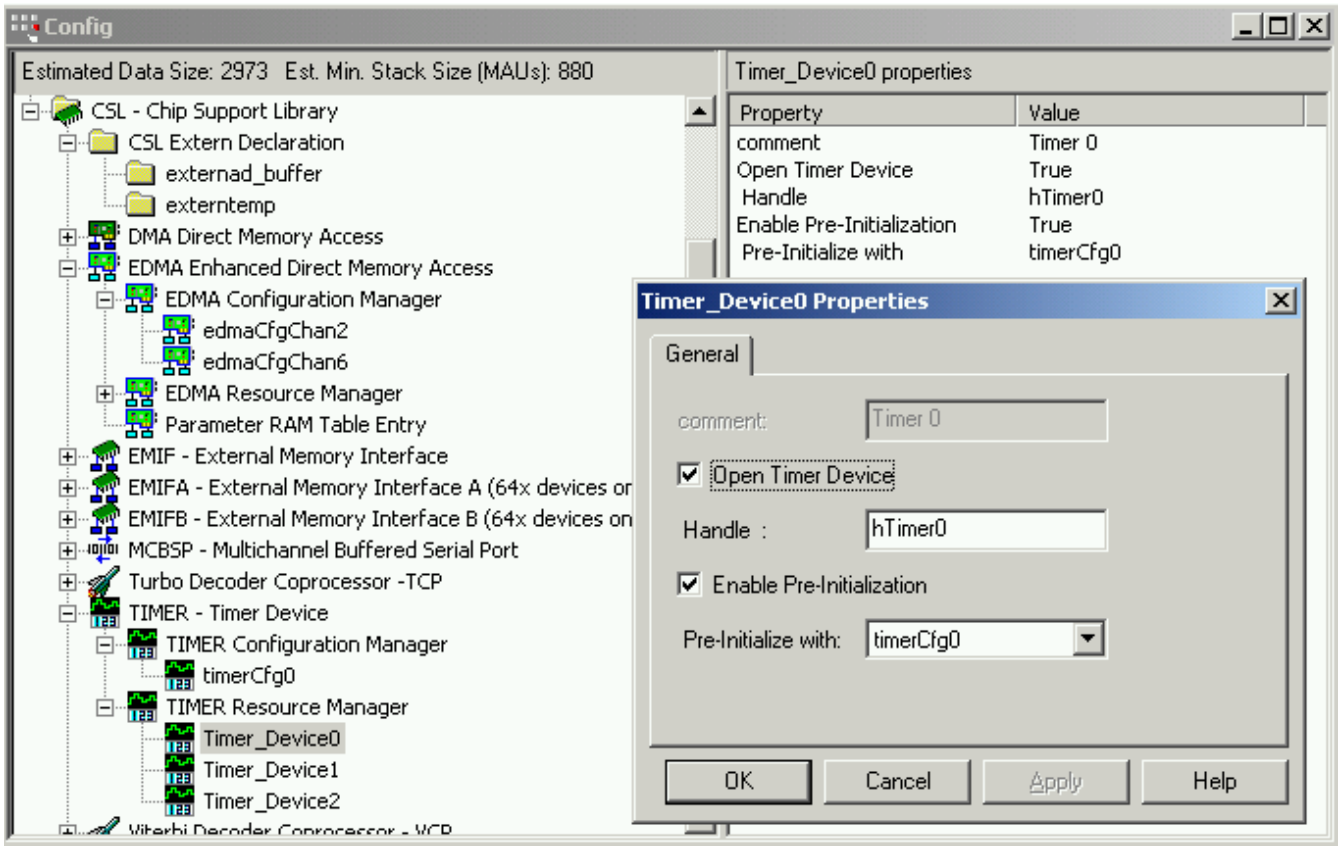

Figure 26. TimerCfg0 Mapped in Timer0 Device Tab Options

5.3 HWI Setup

The interrupt routines can be mapped into the appropriate location in the interrupt vector table using the configuration tool.

In this application note only the EDMA controller interrupt is enabled. Under Scheduling, expand the Hardware Interrupt tab (HWI). Right click on HWI_INT8 and select properties. In the General tab, select the options shown in Figure 27. Interrupt source is EDMA_controller. Interrupt Routine is hwiDMA_isr function. In the Dispatcher tab, select options shown in Figure 28. Use Dispatcher, and enable self interrupt mask.

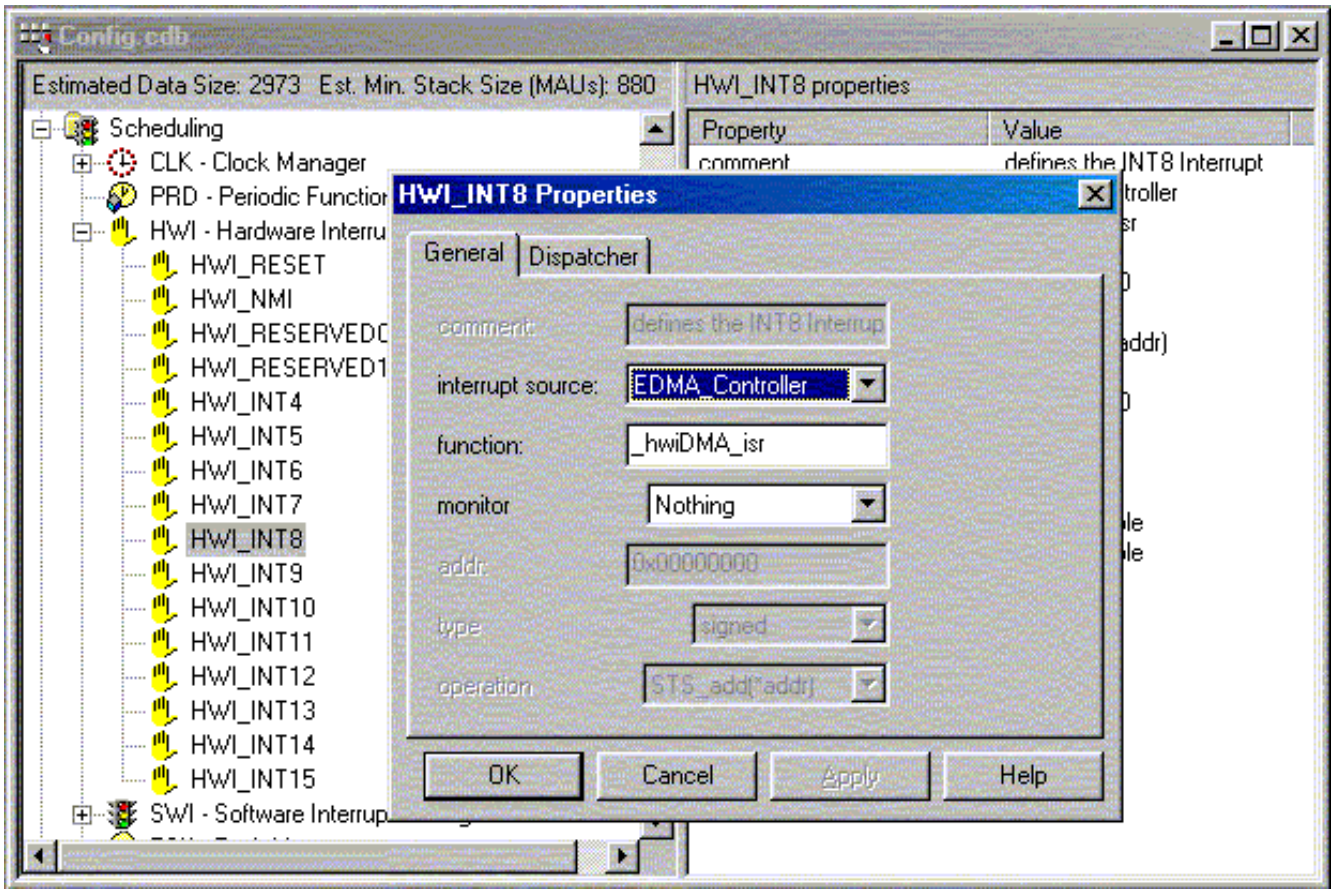

Figure 27. HWI_INT8 General Tab Options

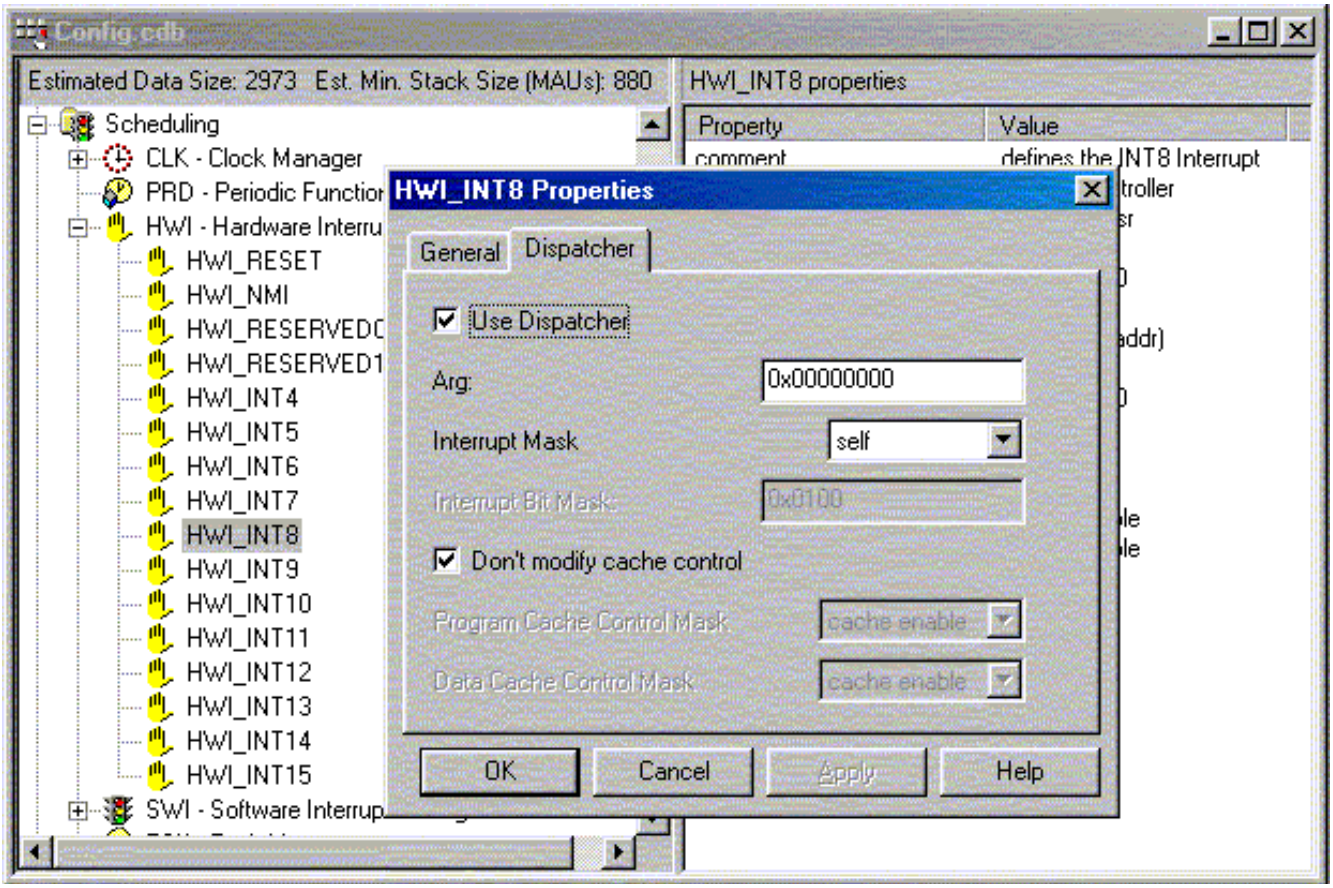

Figure 28. HWI_INT8 Dispatcher Tab Options

5.4 SWI Setup

Software Interrupts are mapped under the Schedule tab in the configuration tool. Right click on SWI−software interrupt manager and select insert SWI twice.

1. This creates two SWI objects. Rename the objects to swiEnablePreph and swiFormat18 bit.
2. Right click on swiEnablePreph and set the fields shown in Figure 29. EnablePreph software interrupt enables the timers, and re-initializes and enables EDMA channels.
3. Right click on swiFormat18bit and set fields as shown in Figure 30. Format18bit software interrupt copies the data from ad_buffer to F_Buffer. While copying data into F_Buffer, the data is formatted such that the top 14 bits of the 32-bit word are cleared.

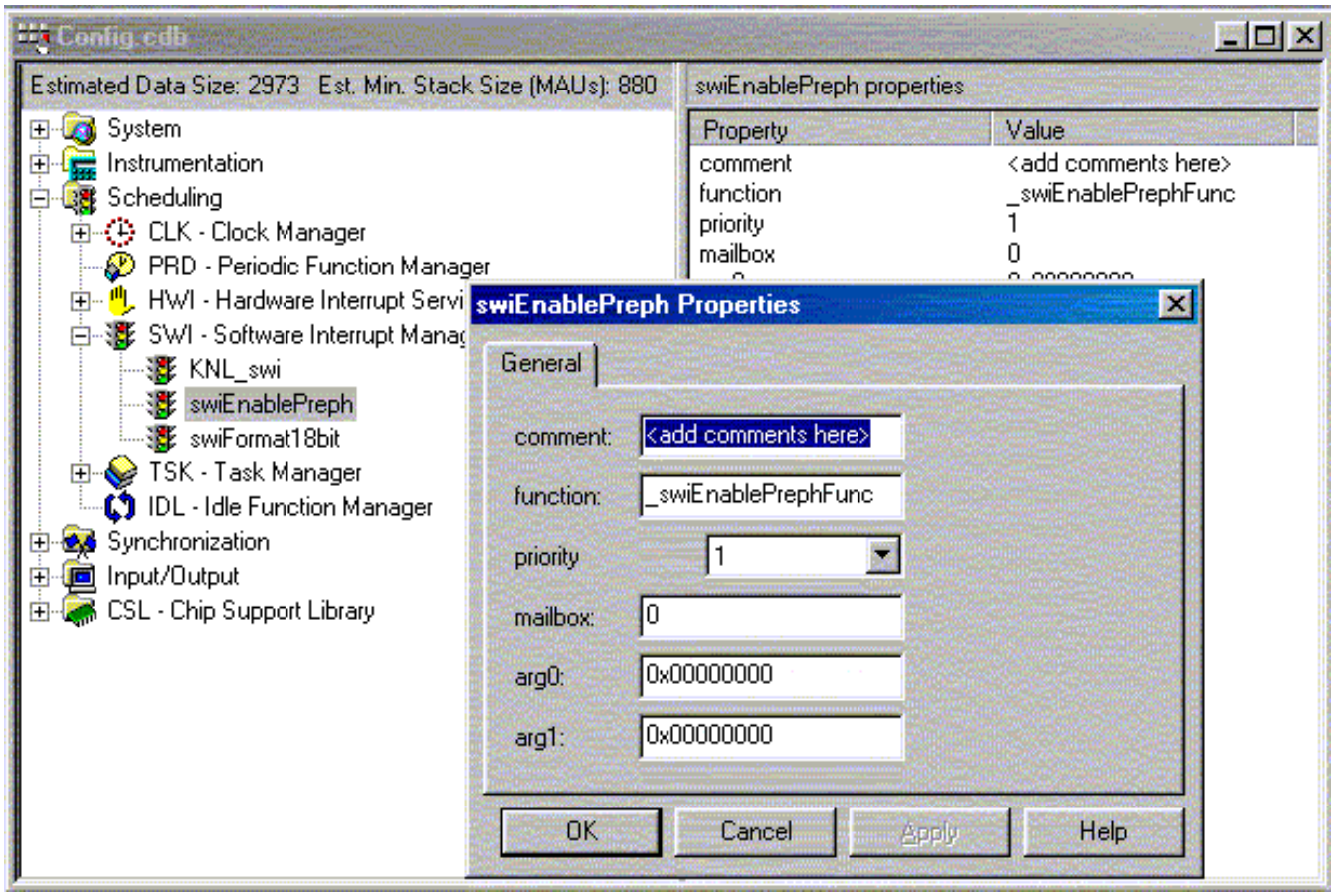

Figure 29. swiEnablePreph Properties

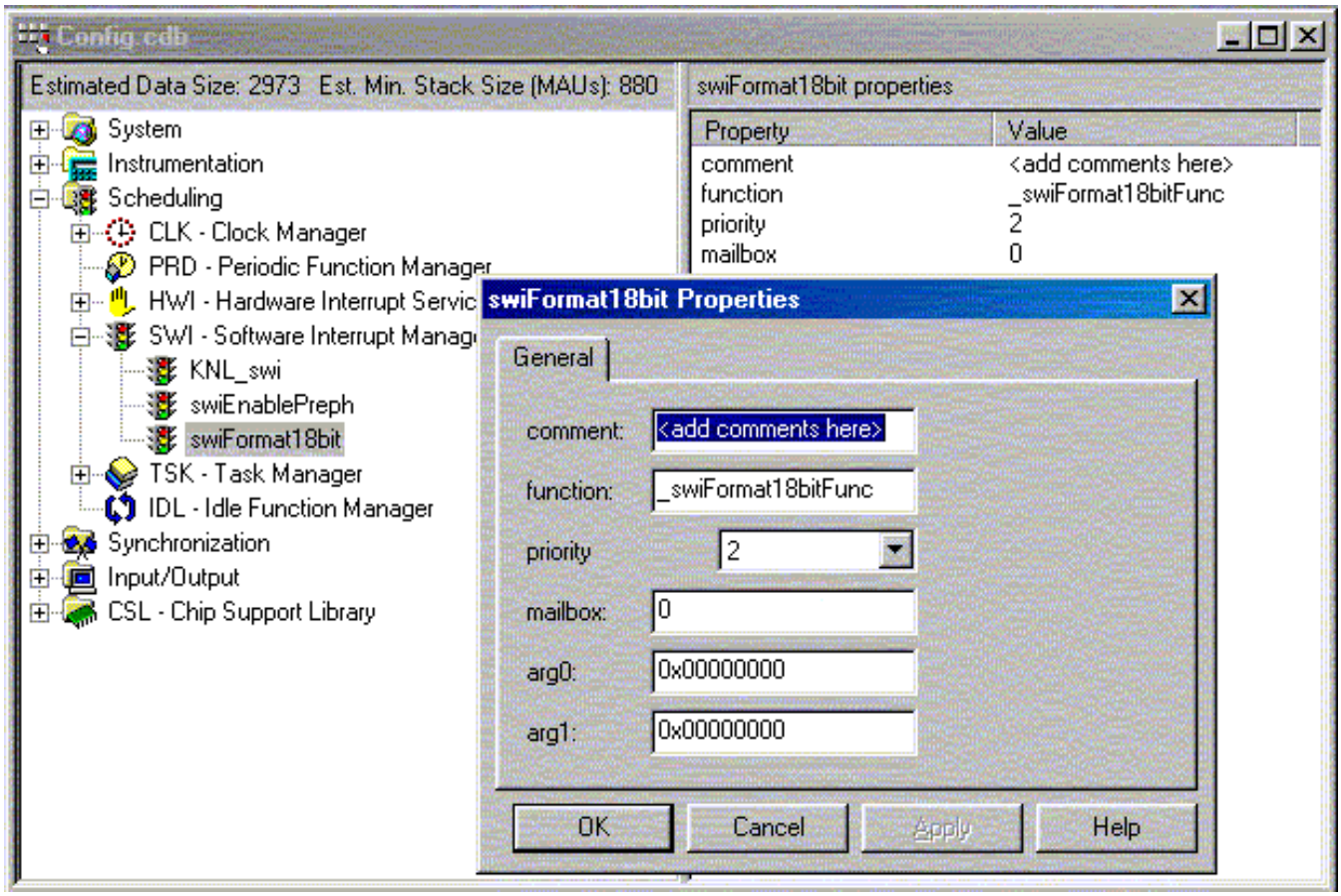

Figure 30. swiFormat18bit Properties

5.5 LOG Setup

The LOG object is a under the Instrumentation tab of the configuration tool. It behaves as a *printf* function in C. Unlike *printf*, this function updates the user screen when the CPU is in an idle state. It is useful in debugging software.

1. Right click on LOG−Event Log Message and select insert LOG as shown in Figure 31. In the general tab, set the options shown in Figure 32. Create an internal circular buffer of 512 words long.

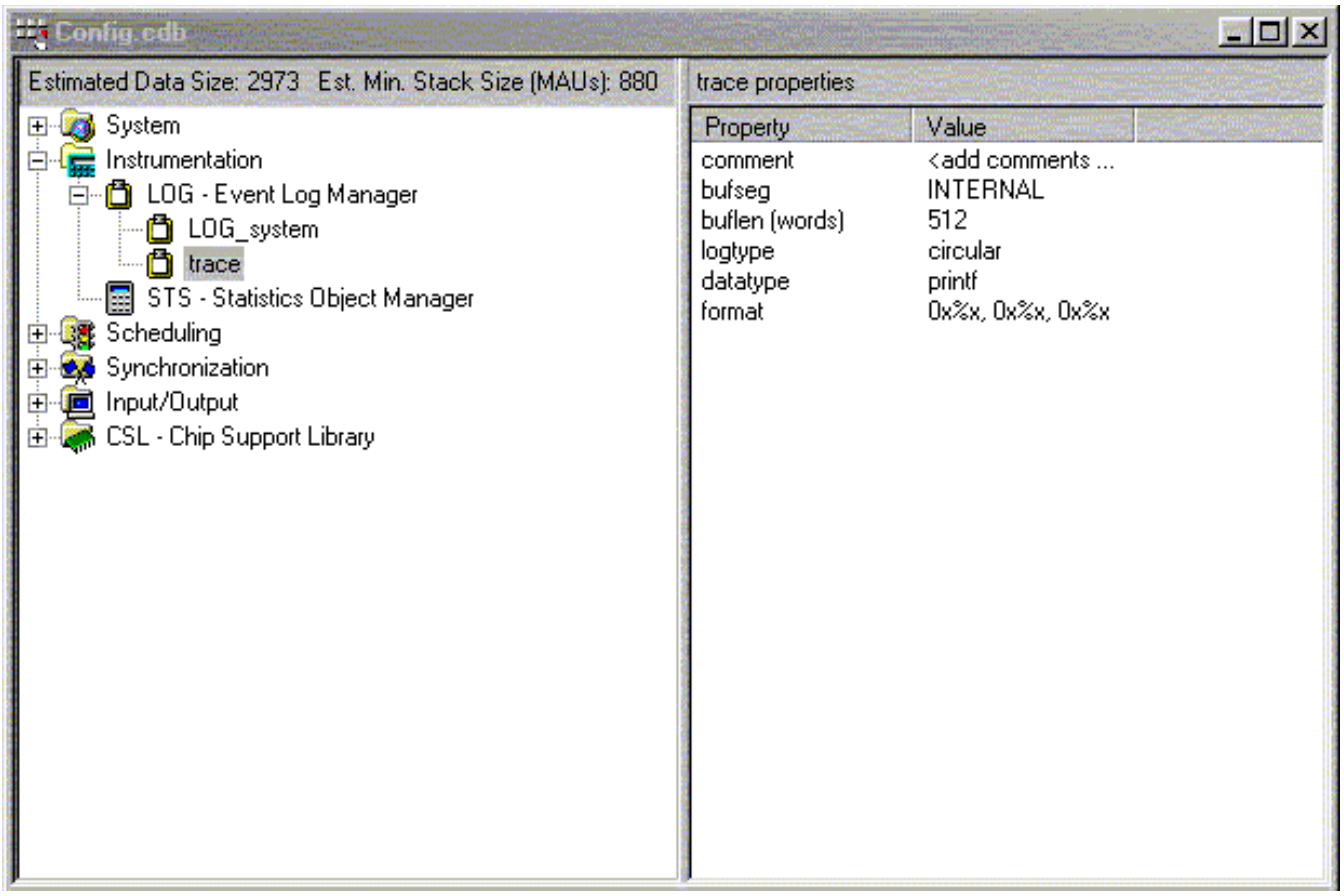

Figure 31. LOG Screen

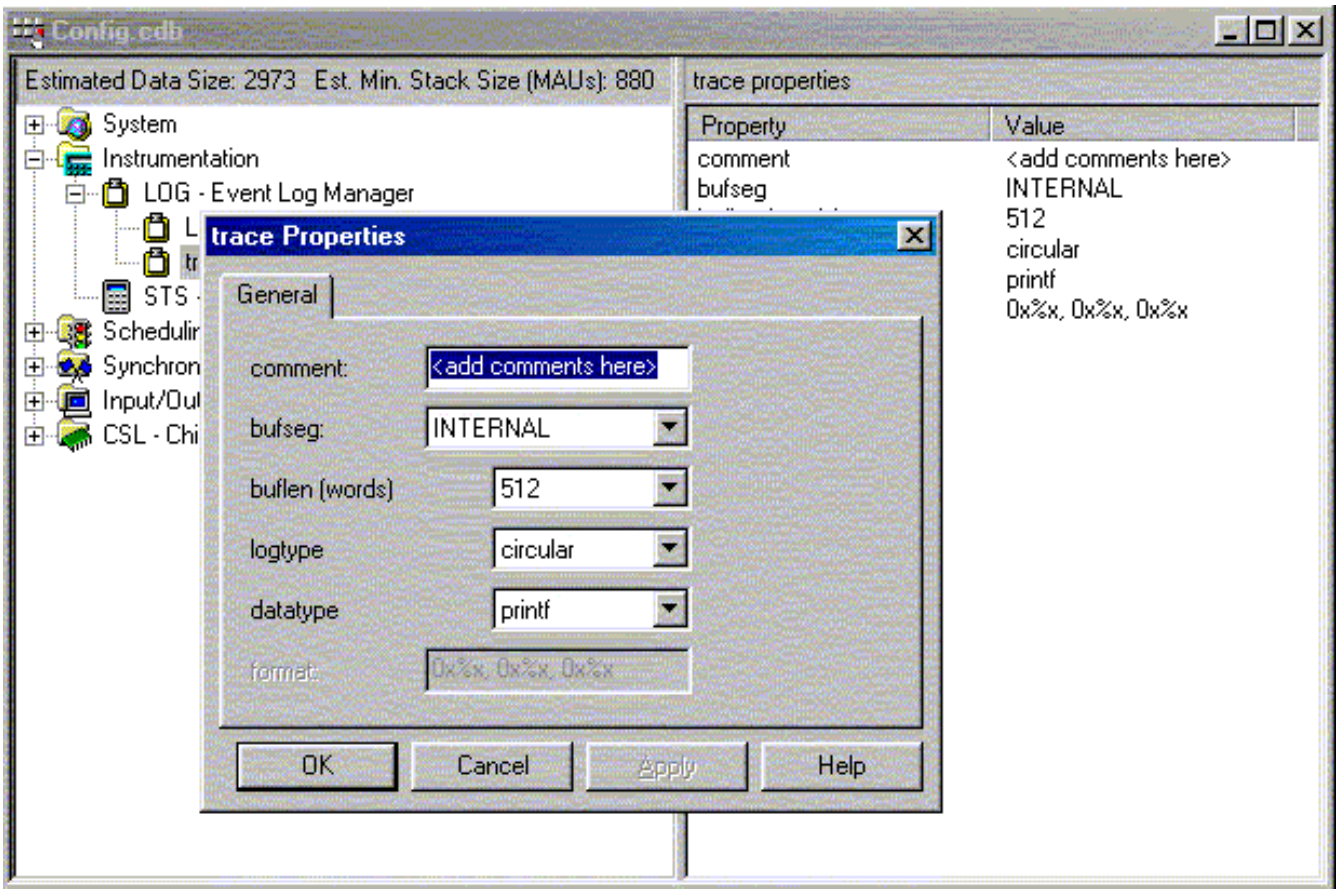

Figure 32. Trace Properties Tab Options

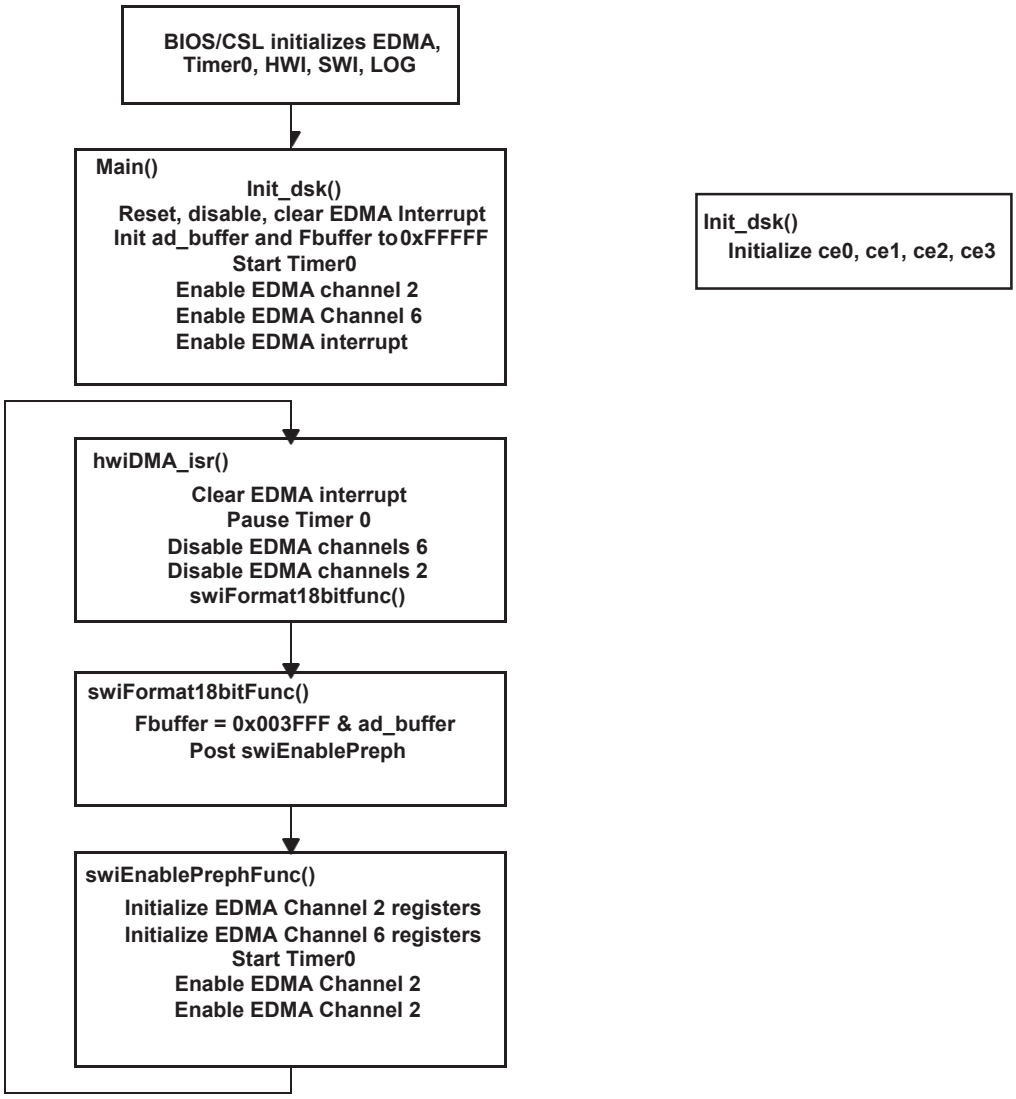

Figure 33. Functions and Flow

Appendix A MAIN.C

```
/*****************************************************************/
/* File: main.c                                                  */
/*    Description: Program for interfacing ADS8383               */
/*    to an C6711 DSK.  Program use the Timer0 to trigger a      */
/* convst# pulse, at about 493kHz 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 timer0 and post software       */
/* interrupt with will mask off the top 14 bits of 32 bit word.  */
/*  The data is stored in Fbuffer.  The program then repeats.    */
/*                                                               */
/* AD converter address: CE2 memory space                        */
/*                 0xA0028000 (Read#)                            */
/*                 0xA002C000 (Convst#)                          */
/* Hardware Connections:                                         */
/*    CS# => CE2#                                                */
/* RD# => Generated from 2-4 decoder mapped to 0xA0028000        */
/* CONVST# => Generated from 2-4 decoder mapped at 0xA002C000    */
/* BUSY => Inverted then wired to EXTERNAL INT6                  */
/* AUTHOR   : DAP Application Group, L. Philipose, Dallas        */
/*    CREATED 2003(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 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);

/* 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);
#pragma DATA_ALIGN(Fbuffer,BLOCK_SZ);
unsigned int ad_buffer[BLOCK_SZ];    /* raw data from AD, written to by EDMA  */
unsigned int Fbuffer[BLOCK_SZ];     /* masked buffer of AD   */
unsigned int temp, n=0;

void main(void)
{
   int i;

    /* initialize the EMIF*/
    init_dsk();
   IRQ_reset(IRQ_EVT_EDMAINT);              /*Reset EDMA interrupt     */
   EDMA_intDisable(TCCINTNUM);              /*Disable EDMA interrupt   */
   EDMA_intClear(TCCINTNUM);                /*Clear EDMA interrupt     */

   EDMA_intEnable(TCCINTNUM);               /*Enable EDMA interrupt    */

   for (i=0; i<BLOCK_SZ; i++) {             /*Initialize data buffers  */
      Fbuffer[i]=0xFFFFF;
        ad_buffer[i]=0xFFFFF;
    }

/*Configuration of Timer0, EDMA channels 2 and 6 where during before entering
  main.c  */
/*Only need to enable them in main. */
   TIMER_start(hTimer0);                    /*Start  A/D  CONVST#  trigger  */
   EDMA_enableChannel(hEdmaTint0);          /*Enable EDMA channel 2 -CONVST# */
   EDMA_enableChannel(hEdmaCha6);           /*Enable EMDA channel 6 -RD#  */

   /* Enable the EDMA controller interrupt   */
   IRQ_enable(IRQ_EVT_EDMAINT);

    LOG_printf(&trace, "GO! \n");           /*Go sample at 490 kHz*/
}

/************************************************************/
/* end main.c                                               */
/************************************************************/
```

Appendix B Functions.C

```
/*****************************************************************/
/* File: functions.c                                             */
/*    Description: Functions for interfacing ADS8383 to C6711    */
/*  init_dsk, swiFormat18bitFunc, hwiDMA_isr, swiEnablePrephFunc */
/* AD converter address: CE2 memory space                        */
/*                 0xA0028000 (Read#)                            */
/*                 0xA002C000 (Convst#)                          */
/* Hardware Connections:                                         */
/*    CS# => CE2#                                                */
/* RD# => Generated from 2-4 decoder mapped to 0xA0028000        */
/* CONVST# => Generated from 2-4 decoder mapped at 0xA002C000    */
/* BUSY => Inverted then wired to EXTERNAL INT6                  */
/* AUTHOR   : DAP Application Group, L. Philipose, Dallas        */
/*    CREATED 2003(C) BY TEXAS INSTRUMENTS INCORPORATED.         */
/* VERSION: 1.0                                                  */
/*****************************************************************/

/*  Include Header File  */
#include ”Configcfg.h”
#include ”dc_conf.h”
#include <csl_legacy.h>

extern unsigned int ad_buffer[BLOCK_SZ];      /* Raw buffer for AD data     */
extern unsigned int Fbuffer[BLOCK_SZ],cnt=0;  /* Buffer for masked 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 ADS8383 EVM.       */
    /* 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() */

/********************************************************/
/*swiFormat18bitFunc:                                   */
/*    Software Interrupt Function resets the upper 14 bits  */
/*    of the 32-bit word and stores 18-bit A/D samples      */
/* in Fbuffer                                  */
/********************************************************/

void swiFormat18bitFunc()
{
```

```
      int i=0;

      for (i=0; i<BLOCK_SZ; i++) {
         Fbuffer[i] = 0x0003FFFF & ad_buffer[i];
   }
    SWI_post(&swiEnablePreph);

}
/*********************************************************/
/*hwiDMA_isr():                                          */
/*  Hardware Interrupt Function disables EDMA channels and  */
/*  Timer0, Then post software interrupt.                */
/*********************************************************/
void hwiDMA_isr()
{  cnt++;
//    LOG_printf(&trace, "hwiDMA_isr \n");
   /* Clear the pending interrupt from the EDMA interrupt pending register */
    EDMA_intClear(TCCINTNUM);
    TIMER_pause(hTimer0);
    EDMA_disableChannel(hEdmaCha6);
    EDMA_disableChannel(hEdmaTint0);
   swiFormat18bitFunc();
}

/*********************************************************/
/*swiEnablePrephFunc:                                    */
/*    Software Interrupt Function configures EDMAC2 and     */
/*    EDMAC6, enables respective EDMA channels and Timer0   */
/*********************************************************/
void swiEnablePrephFunc()
{
   EDMA_config(hEdmaTint0, &edmaCfgChan2);
    EDMA_config(hEdmaCha6, &edmaCfgChan6);
   TIMER_start(hTimer0);
   EDMA_enableChannel(hEdmaTint0);
    EDMA_enableChannel(hEdmaCha6);
}

/*************************************************************/
/* End Functions.c                                           */
/*************************************************************/
```

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

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

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