Application Report
SLAA210A–October 2004–Revised August 2007

Getting Started With the Data Converter Plug-In

Lars Lotzenburger
E-HPA Marketing and Systems Engineering

ABSTRACT

The Data Converter Plug-In (DCP) is a software plug-in shipped with the Code Composer Studio™ (CCS) for the TMS320C2800™, TMS320C5000™, TMS320C6000™, and TMS320C6400™ digital signal processor (DSP) families. It generates C source code drivers based on user inputs for Texas Instruments (TI) data converters (ADC, DAC, and CODEC) connected to a TI DSP. This application report introduces the tool and its use. A step-by-step walkthrough demonstrates how to configure a converter using the THS1206 parallel ADC as an example. The integration of the generated driver files into new or existing projects, and the steps to set up a CCS project, are also covered. These practices are easily adapted to other converters included in the DCP.

Contents

1 Introduction 2
2 Generation of Driver Files With the Data Converter Plug-In 3
3 Embedding the Driver in the Application 14
4 References 18

List of Figures

1 CCS Project Creation 4
2 DCP Start Screen 5
3 Data Converter Selection 6
4 Add a Converter to the DCP 7
5 DSP Tab 8
6 GUI For the THS1206 10
7 Files Property Page 11
8 Flow of an API Function Call 12
9 Seed Selection Window 14
10 TCF File Main Window 15
11 Interrupt Properties Dialog 16
12 Application Layer Example 17

1 Introduction

Data converters provide the interface between the analog world and the digital world of the digital signal processor. In previous years, most converter ICs offered only simple conversion and had only straightforward digital interfaces. This limitation has changed recently for various reasons, with more functionality moving into the converter. Some converters have special features suited to particular applications, such as high-precision differential inputs, while some have enhanced digital interfaces. Others execute common pre/post-processing tasks to decrease the CPU load (for example, volume control on audio converters).

Different applications require different processor interfaces. Many different interfaces are on the market, ranging from simple interfaces with minimum pin counts such as the Inter-Integrated Circuit™ (I^{2}C™) interface, to parallel interfaces mapped into the processor memory space to maximize data throughput.

These factors complicate the design process when working with state-of-the art converters. Frequently, design engineers must read through lengthy data sheets simply to get the selected converter operating. Initially, the interface between the converter and the processor must be clarified, because different approaches are sometimes possible. Following this clarification, the behavior of more complex converters must be defined through the configuration registers.

At this stage in the design process, the data converter support software (DCP) can be a useful tool. It generates driver source code in response to inputs from the user. All interface and configuration settings are made through an easy-to-use graphical user interface (GUI). The drivers have been developed and tested on the data converter evaluation modules (EVMs) that are also available from Texas Instruments. See the DCP page on the TI web site at www.ti.com for a complete list of TI data converters supported by the DCP.

2 Generation of Driver Files With the Data Converter Plug-In

2.1 Set Up and Open Code Composer Studio (CCS)

Texas Instruments offers two versions of Code Composer Studio:

1. **CCS DSK version:** The Development Starter Kit (DSK) comes with a fully functional version of CCS. The only limitation is that this CCS version must be used with the DSK. Therefore, no particular setup program is shipped with the DSK version.
2. **CCS full version:** The setup program for this version allows the user to select different configuration schemes.
   - Hardware: In this mode, software developed in the CCS environment runs on the target hardware (DSP board) connected to the host computer. The application is debugged on real hardware. The interface between the host computer and the DSP hardware can be the parallel port, an emulator (XDS5x0), or the universal serial bus (USB).
   - Simulator: Applications can be developed with no target hardware connected to the computer. CCS simulates the DSP and runs the application on it.

   For more details regarding the setup program, see the *Code Composer Studio User's Guide* (SPRU328), available for download at www.ti.com.

Action: Start the setup program, and choose the appropriate driver (if not using the DSK version of CCS).

When you close the setup program, you can start the Code Composer Studio Integrated Development Environment (CCS IDE).

Action: Launch CCS using the icon on your desktop.

The DCP is shipped with the CCS package. However, it is recommended to download the latest version of the DCP through the update advisor utility in CCS or directly from the DCP web site.

Action: Select *CCS menu* → *Help* → *Update Advisor* → *Check For Updates*.

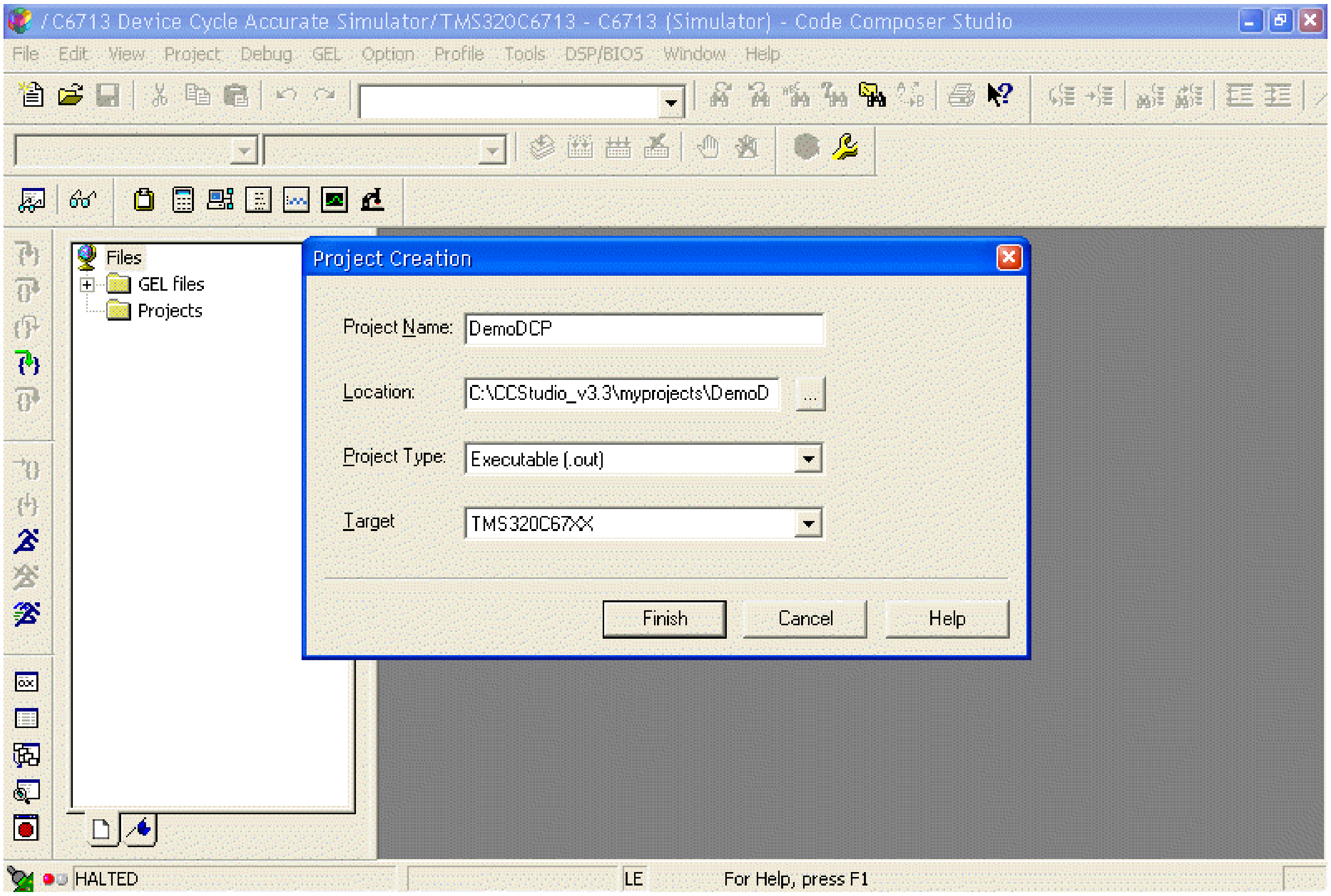
2.2 Create a New Project in CCS

The DCP creates a set of files and adds them to a CCS project. Therefore, it is important to create a new project or to open an existing project before running the DCP.

Action:

- Select *CCS menu* → *Project* → *New*… to create a new project, as shown in Figure 1; or
- Select *CCS menu* → *Project* → *Open*… to open an existing project

Go to Section 2.3 if you are working on an existing project.



Project Name: DemoDCP

Location: C:\CCStudio_v3.3\myprojects\DemoD

Project Type: Executable (.out)

Target TMS320C67XX

Figure 1. CCS Project Creation

Action: In the field labeled *Project Name*, type the name of your project; for example, *DemoDCP*.

Verify the location where your project will be created. It is usually *c:\CCStudio_v3.3\myprojects* where ***c:\CCStudio_v3.3*** is the CCS installation directory.

Set *Project Type* to *Executable* and *Target* to your DSP family (in this example, TMS320C67XX).

Action: Click the *Finish* button to create the project.

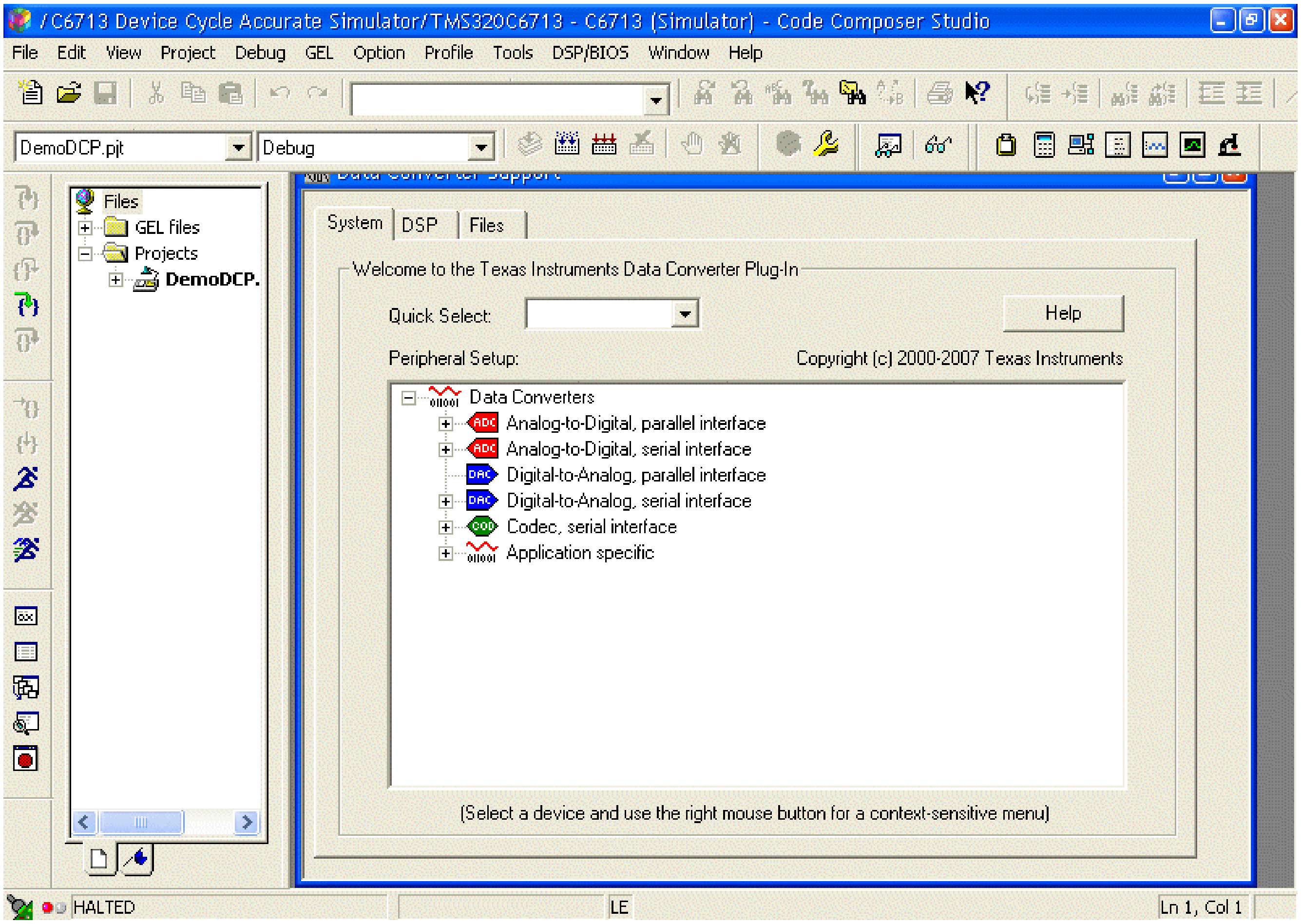
CCS automatically creates a subdirectory and places the project file in it. The newly-created project appears in the project pane (at the left side of the CCS window).

2.3 Open DCP

With the project open, you are ready to start the DCP.

Action: Select *CCS menu* → *Tools* → *Data Converter Support* to launch the DCP.

The DCP opens with the *System Tab* selected, as shown in Figure 2.



Welcome to the Texas Instruments Data Converter Plug-In

Quick Select:

Peripheral Setup:

Copyright (c) 2000-2007 Texas Instruments

- Data Converters
  - Analog-to-Digital, parallel interface
  - Analog-to-Digital, serial interface
  - Digital-to-Analog, parallel interface
  - Digital-to-Analog, serial interface
  - Codec, serial interface
  - Application specific

(Select a device and use the right mouse button for a context-sensitive menu)

Figure 2. DCP Start Screen

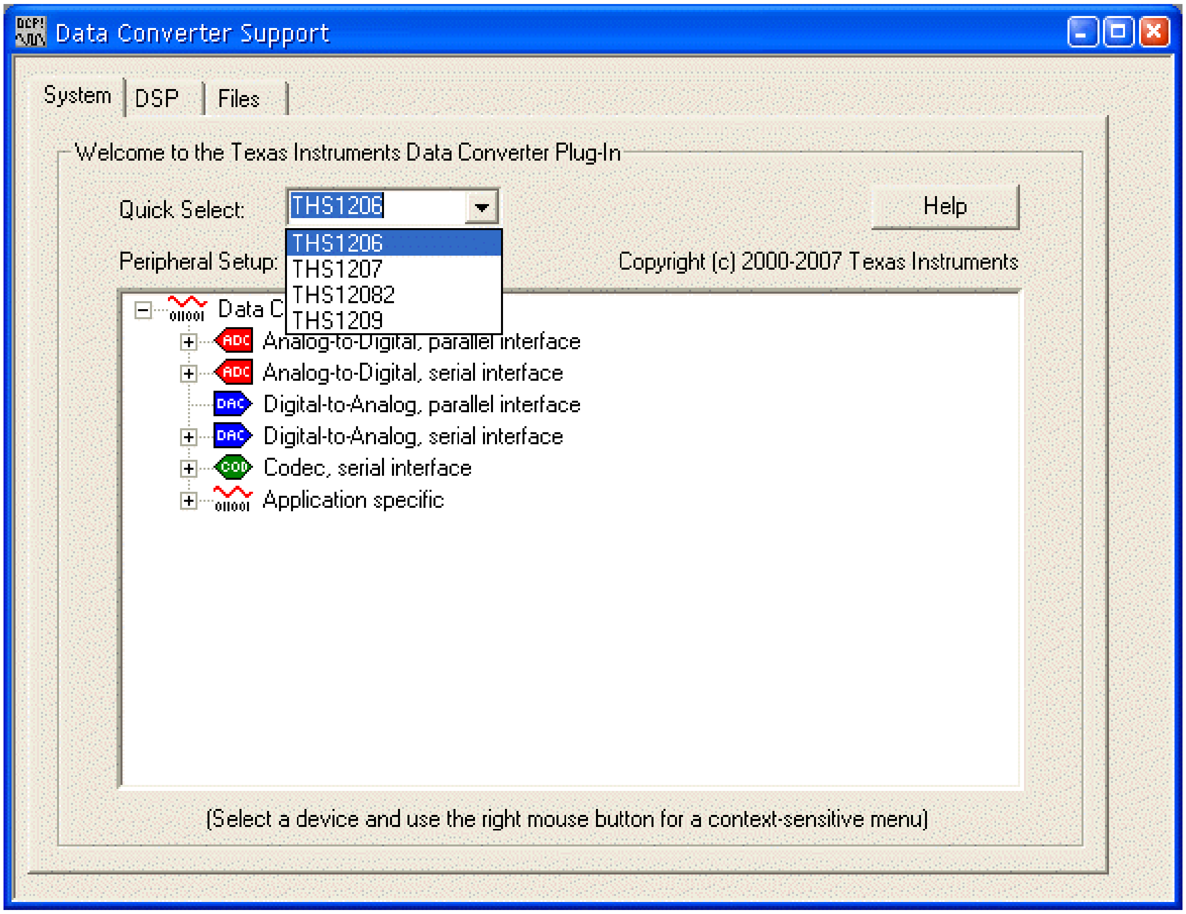
2.4 Add a Converter to the DCP

The next step is to add the desired converter to the DCP. This step can be accomplished in two ways: either navigate the converter tree to find the desired converter based on the interface and width, or use the *Quick Select* feature (used for this example).

As you type the name of the converter name in the *Quick Select* field, the DCP searches for all converters beginning with the letters and numbers typed. You can easily select your converter from the generated drop-down list as shown in Figure 3.

Action: Type the converter name you want to use in the *Quick Select* entry field. You can type the entire part number and press the *Enter* key, or to use the generated drop-down list after typing the first few characters of the desired converter name.

Note: In this document, the THS1206 is used as an example.


Data Converter Support
System
DSP
Files
Help
THS1206
THS1207
THS12082
THS1209
Data C
Analog-to-Digital, parallel interface
Analog-to-Digital, serial interface
Digital-to-Analog, parallel interface
Digital-to-Analog, serial interface
Codec, serial interface
Application specific

Welcome to the Texas Instruments Data Converter Plug-In

Quick Select: THS1206

Peripheral Setup:

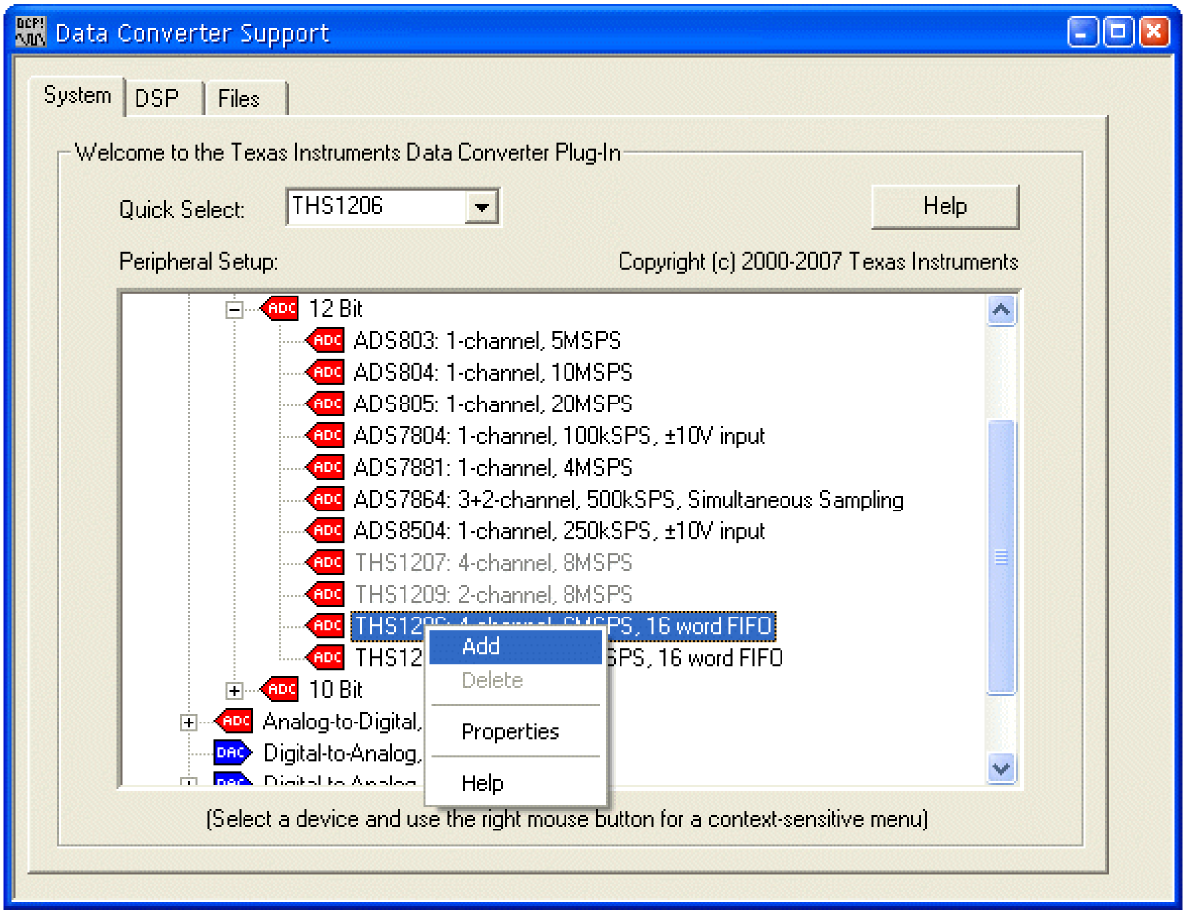
Copyright (c) 2000-2007 Texas Instruments

(Select a device and use the right mouse button for a context-sensitive menu)

Figure 3. Data Converter Selection

When a converter is selected, the DCP expands the tree, highlighting the device and placing the cursor on it. Some converters are colored gray. This shading means that for this converter on this specific DSP platform, no complete driver is available. You can still create the register settings for the converter that can later be used in your own driver. In this case, a header file (*dc_conf.h*) describing the converter setup; this process is described in Section 2.7.2. The next step is to add the converter to the system to make it available for configuration.

Action: Right-click on the highlighted converter. A pop-up menu appears (see Figure 4). Click on the first entry labelled *Add*. This selection includes the converter in the system and adds a configuration tab for it to the DCP window.



Welcome to the Texas Instruments Data Converter Plug-In

Quick Select: THS1206

Peripheral Setup:

Copyright (c) 2000-2007 Texas Instruments

- 12 Bit
  - ADS803: 1-channel, 5MSPS
  - ADS804: 1-channel, 10MSPS
  - ADS805: 1-channel, 20MSPS
  - ADS7804: 1-channel, 100kSPS, ±10V input
  - ADS7881: 1-channel, 4MSPS
  - ADS7864: 3+2-channel, 500kSPS, Simultaneous Sampling
  - ADS8504: 1-channel, 250kSPS, ±10V input
  - THS1207: 4-channel, 8MSPS
  - THS1209: 2-channel, 8MSPS
  - THS12 ... PS, 16 word FIFO
  - THS12 ... SPS, 16 word FIFO
- 10 Bit
- Analog-to-Digital,
- Digital-to-Analog,

(Select a device and use the right mouse button for a context-sensitive menu)

Figure 4. Add a Converter to the DCP

Notice that a new tab called <*Converter name*>_1 has been added to the DCP.

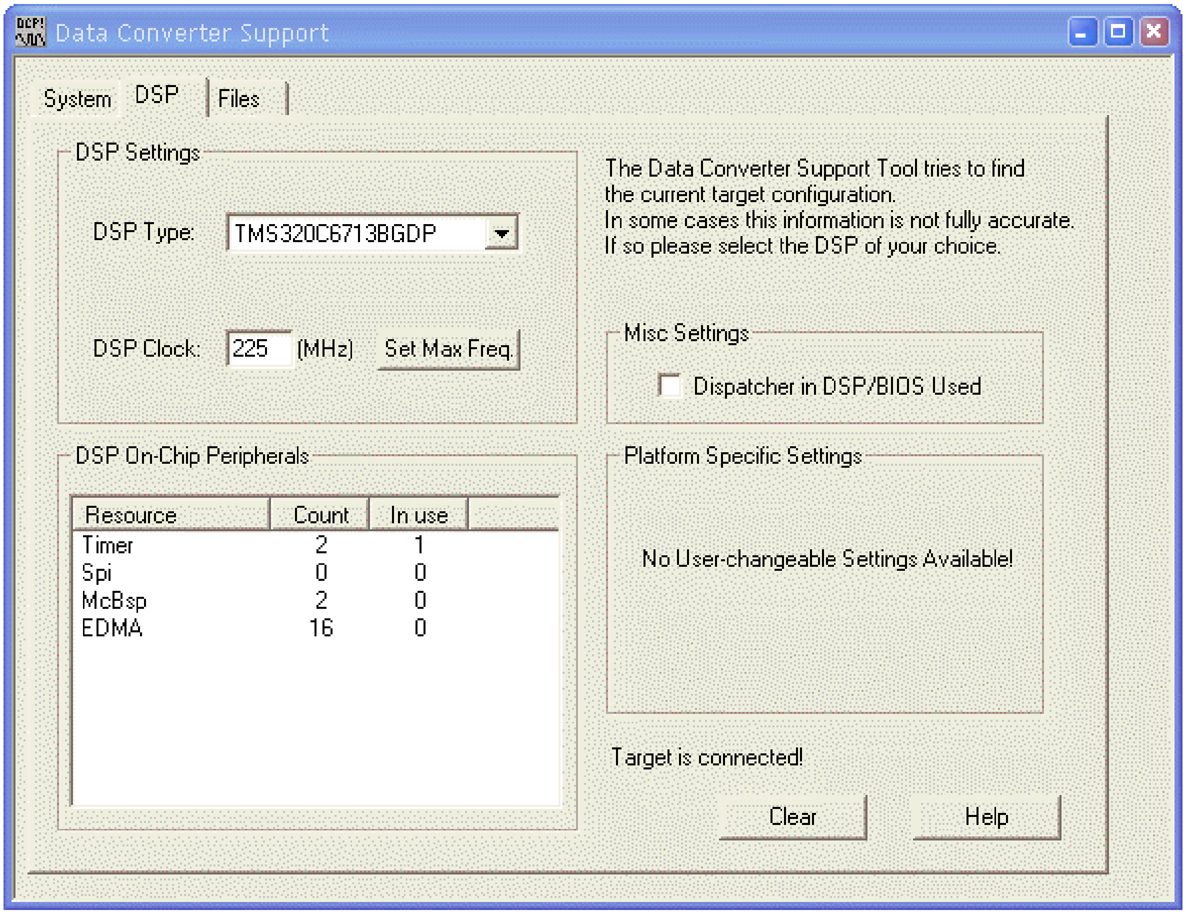
The pop-up menu (shown in Figure 4) has two more active menu items. Clicking on the *Properties* item reveals the driver seed and driver output file names used for the selected converter. The *Help* item from the pop-up menu opens the help file of the DCP.

The DCP supports multiple converters in the system. If your system has more than one converter connected to the DSP, repeat the steps from the beginning of this section.

2.5 DSP Configuration

The DSP-specific settings should be chosen before the converter settings are entered.

Action: Click on the *DSP* tab to display the DSP-specific controls.



DSP Settings

DSP Type: TMS320C6713BGDP

DSP Clock: 225 (MHz)

The Data Converter Support Tool tries to find the current target configuration.
In some cases this information is not fully accurate.
If so please select the DSP of your choice.

Misc Settings

□ Dispatcher in DSP/BIOS Used

DSP On-Chip Peripherals

Resource	Count	In use
Timer	2	1
Spi	0	0
McBsp	2	0
EDMA	16	0

Platform Specific Settings

No User-changeable Settings Available!

Target is connected!

Figure 5. DSP Tab

The DSP page opens (Figure 5). Through the configuration setup, the DCP automatically determines the DSP family to be used, and lists all supported DSPs in the *DSP Type* control.

Action: Choose the DSP you are using in your system from the drop-down list.

The *DSP Clock* control should be set to the DSP frequency used in the design. This value is used for frequency calculation for the peripherals only. It does not control the DSP frequency.

Action: Type the DSP frequency you are using in your system into the *DSP Clock* edit field.

The *Dispatcher in DSP/BIOS Used* check box under *Misc Settings* informs the DCP if you are using the dispatcher in the DSP/BIOS. If you are not using DSP/BIOS, this checkbox must be left unchecked.

Note: If you are using the dispatcher in the DSP/BIOS, the *interrupt* keyword in front of each interrupt service routine (ISR) in your driver source code must be removed. DCP does this for you.

2.5.1 TMS320F2800 Family Specific Settings

If the TMS320F2800 family is used, three additional controls are added in the *Platform Specific Settings* area. This family does not use the DSP clock as a source clock for the peripherals directly. The DSP generates a low-speed clock (LSPCLK) from the DSP clock that is used for the timer and a high-speed clock (HSPCLK) used for the serial port. The two controls *LSPCLK field* and *HSPCLK field* allow you to select the ratio of the DSP clock to LSPCLK or HSPCLK. Again, this selection is important for DSP clock generation in the driver because peripherals such as serial ports or timers depend on LSPCLK or HSPCLK rather than on the DSP system clock.

The driver code generated by the DCP can initialize the divider by writing the appropriate register in the DSP (select *DCP Driver* button). If it is not desired for the driver to perform this initialization, select the *User Program* button

2.6 *Converter Settings*

Now, enough DSP information is in the DCP to configure the data converter.

Action: Click on the tab for the data converter you added in Section 2.4.

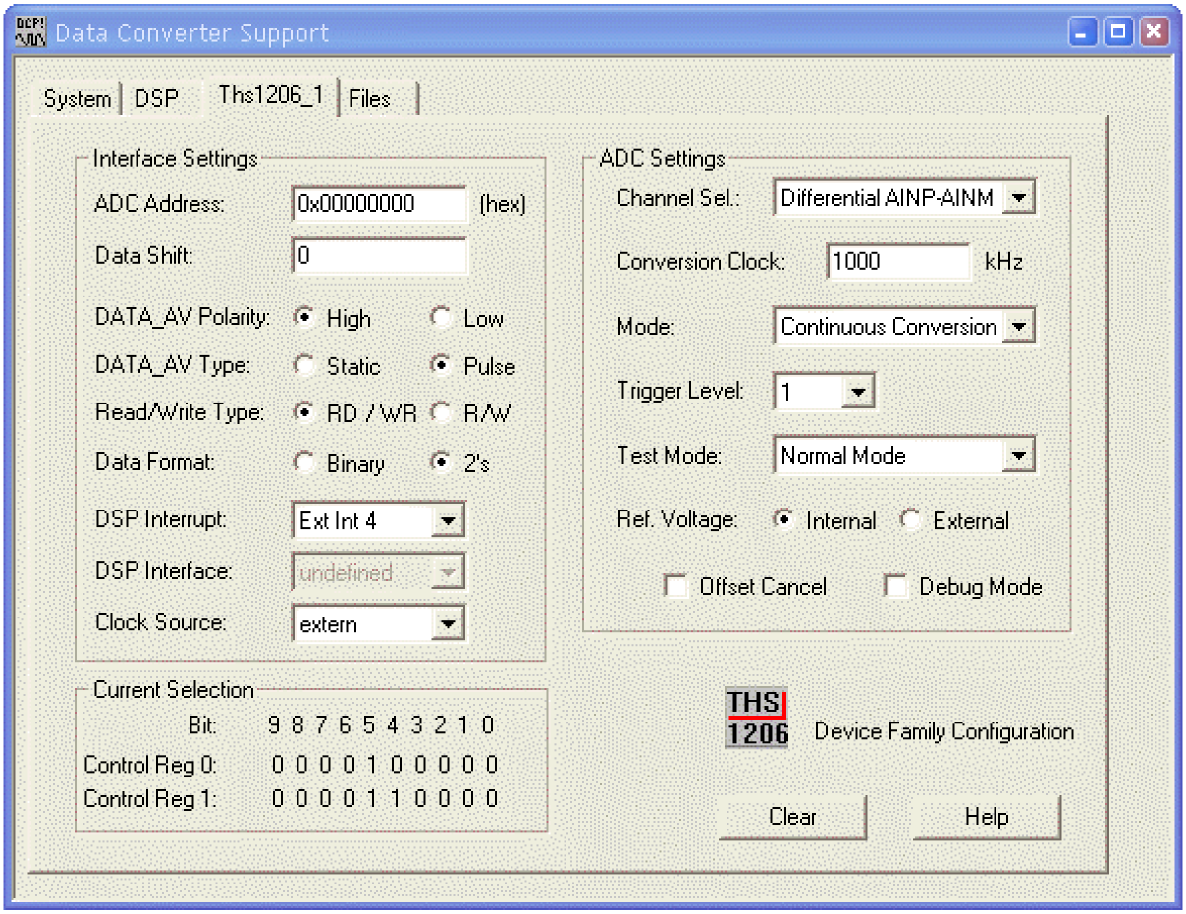
This page has several settings reflecting the possible configuration of the device. Converters differ in capabilities and interfaces. As a result, each converter has its own unique setup tab. All pages are separated into:

- Interface Settings
- Converter Settings (if applicable)

The *Interface Settings* section is where the user enters information about the physical connection between the DSP and the data converter.

The *Converter Settings* section determines the behavior of the converter after power up. These settings are completely configurable—the converter works with any combination of the controls.

The THS1206 GUI is shown in Figure 6. One important feature of the tool is that it is not possible to choose invalid combinations—the DCP validates all user inputs against the device constraints and displays only valid combinations for the particular converter selected.



Data Converter Support

Interface Settings

ADC Address: 0x00000000 (hex)

Data Shift: 0

DATA_AV Polarity: ● High ○ Low

DATA_AV Type: ○ Static ● Pulse

Read/Write Type: ● RD / WR ○ R/W

Data Format: ○ Binary ● 2's

DSP Interrupt: Ext Int 4

DSP Interface: undefined

Clock Source: extern

ADC Settings

Channel Sel.: Differential AINP-AINM

Conversion Clock: 1000 kHz

Mode: Continuous Conversion

Trigger Level: 1

Test Mode: Normal Mode

Ref. Voltage: ● Internal ○ External

□ Offset Cancel □ Debug Mode

Current Selection

Bit:	9	8	7	6	5	4	3	2	1	0
Control Reg 0:	0	0	0	0	1	0	0	0	0	0
Control Reg 1:	0	0	0	0	1	1	0	0	0	0

Figure 6. GUI For the THS1206

Action: Choose the desired settings for all converters in your system.

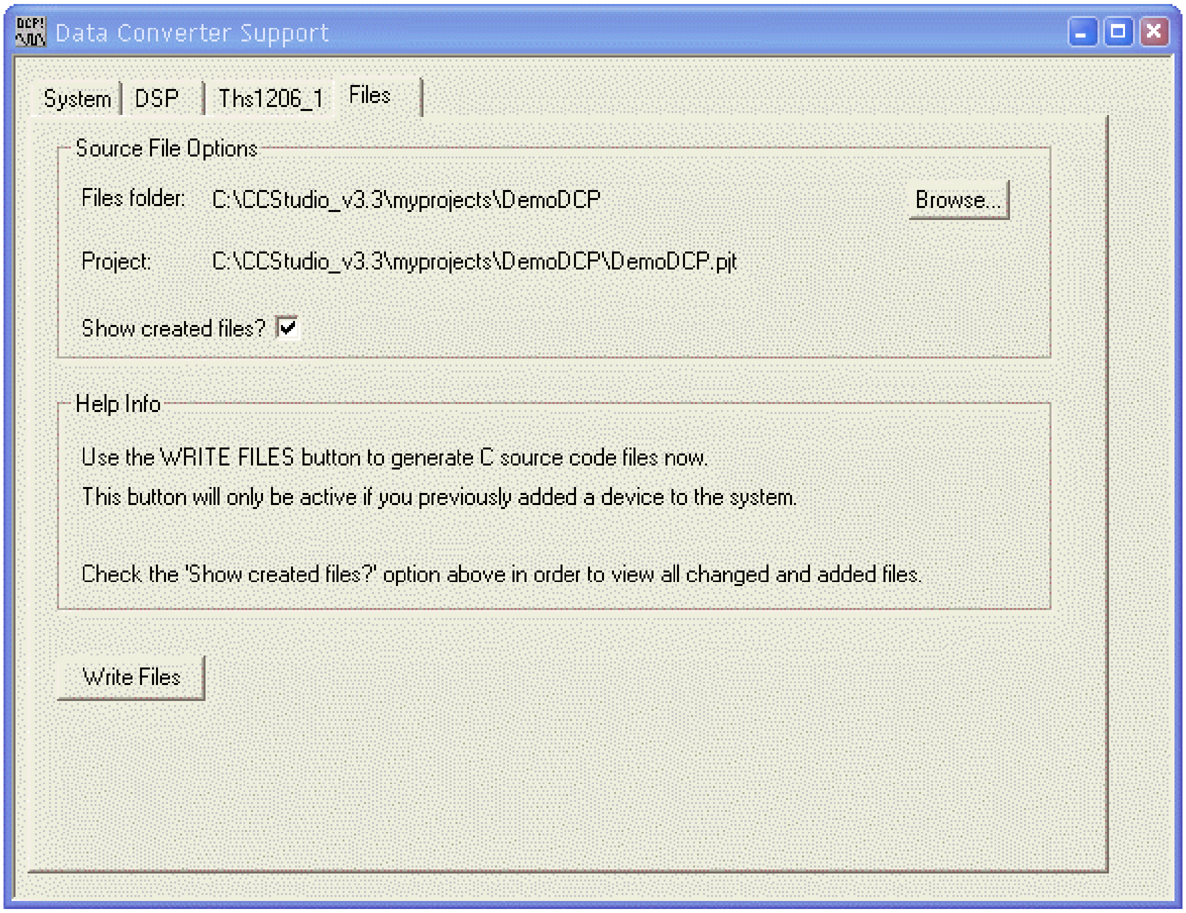
2.7 Code Generation

Once the converter is set up, the driver code can be generated.

Action: Click on the *Files* tab, as shown in Figure 7.

The last property page opens. The *Show Files* checkbox specifies whether the generated source files remain open in the CCS environment (checked) or are closed (unchecked) once they are generated.

Verify the that directory where the driver files will be generated is the project directory. In our example, the directory is named *c:\CCStudio_v3.3\myprojects\DemoDCP*.



Files folder: C:\CCStudio_v3.3\myprojects\DemoDCP

Project: C:\CCStudio_v3.3\myprojects\DemoDCP\DemoDCP.pjt

Show created files? ☑

Use the WRITE FILES button to generate C source code files now.

This button will only be active if you previously added a device to the system.

Check the 'Show created files?' option above in order to view all changed and added files.

Figure 7. Files Property Page

Action: Click on the *Write Files* button to start the file creation process.

Five files are generated, unless you selected a gray-colored converter in the tree. In this case, the *driver source code file* and the *driver header file* are not generated.

2.7.1 'defines' Header File (dc_conf.h)

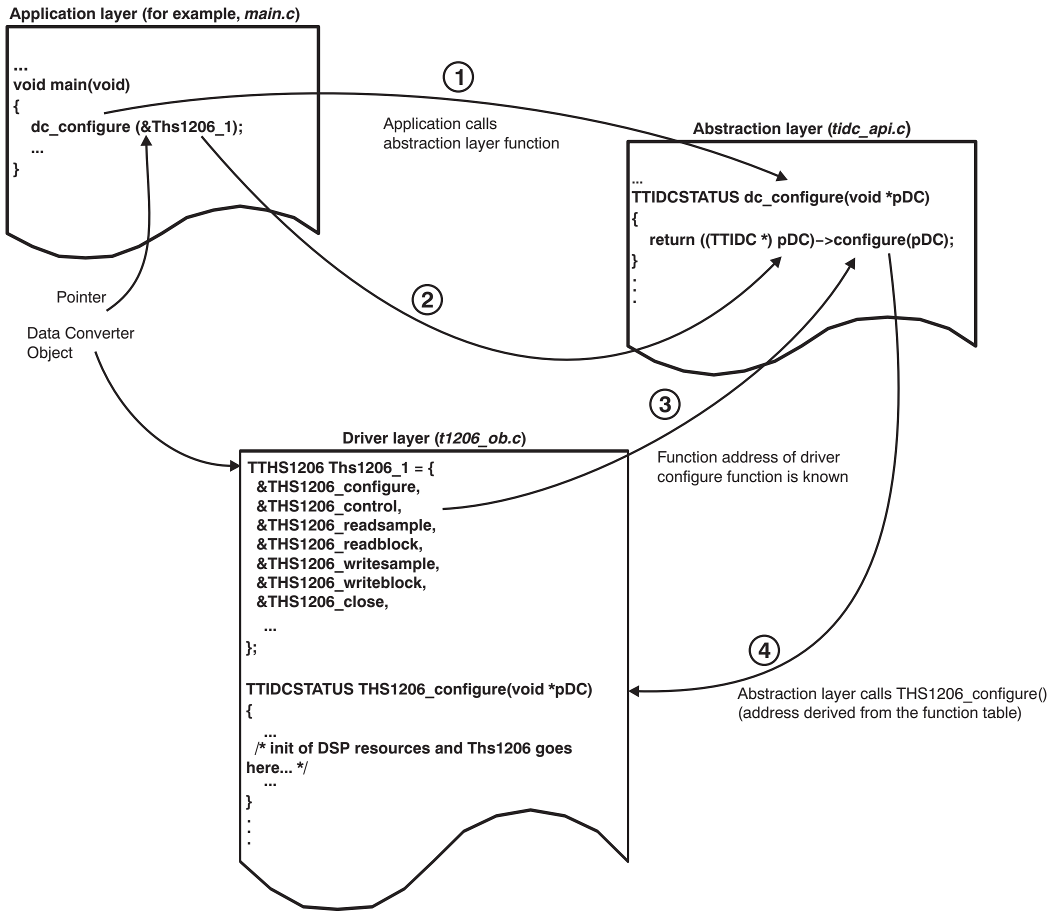
This file is always called *dc_conf.h* and includes definitions only. These definitions describe the interface and register settings of the converter as well as the DSP used. In a multiple-converter environment, the settings of all converters are collected here.

2.7.2 Abstraction Layer Files (tidc_api.c, tidc_api.h)

Every call to a driver function is made through a unique API that consists of seven functions, as in the driver source code file described in Section 2.7.3.

- dc_configure()
- dc_control()
- dc_readsample()
- dc_readblock()
- dc_writesample()
- dc_writeblock()
- dc_close()

The application layer normally calls the driver source code function through this standard API rather than calling the driver source code functions directly, providing a simple, common interface to the driver from the application point of view. The underlying driver function that is actually called (important in a multiple-converter environment) is selected based on the data converter object passed as an argument. Figure 8 illustrates these details.



Application layer (for example, *main.c*)
1
Application calls
abstraction layer function
Abstraction layer (*tidc_api.c*)
Pointer
Data Converter
Object
2
3
Function address of driver
configure function is known
Driver layer (*t1206_ob.c*)
4
Abstraction layer calls THS1206_configure()
(address derived from the function table)

```
...
void main(void)
{
   dc_configure (&Ths1206_1);
   ...
}
```

```
...
TTIDCSTATUS dc_configure(void *pDC)
{
   return ((TTIDC *) pDC)->configure(pDC);
}
.
.
.
```

```
TTHS1206 Ths1206_1 = {
  &THS1206_configure,
  &THS1206_control,
  &THS1206_readsample,
  &THS1206_readblock,
  &THS1206_writesample,
  &THS1206_writeblock,
  &THS1206_close,
    ...
};

TTIDCSTATUS THS1206_configure(void *pDC)
{
    ...
  /* init of DSP resources and Ths1206 goes
here... */
    ...
}
.
.
.
```

Figure 8. Flow of an API Function Call

1. The application layer calls the abstraction-layer *dc_configure(void* pDC)* function with the converter (identified by the data converter object pointer) as a parameter.
2. This data converter object pointer is used as the basis to access the function pointer table of the correct object.
3. The abstraction layer calls the function identified by the address located at position 'configure' in the function pointer table.
4. The *THS1206_configure(void *pDC)* function is called and executed.

2.7.3 Driver Source Code File (txxxx_ob.c)

The source code of the driver is included in this file. It contains the seven standard functions, while some of them may not be implemented (depending on the converter type):

- <*Converter name*>_configure: Allocates and configures the DSP resources used and configures the converter .
- <*Converter name*>_control: Controls the converter; for example, channel selection, or power down/up.
- <*Converter name*>_readsample: Reads one sample from the converter.
- <*Converter name*>_readblock: Reads a block of data from the converter.
- <*Converter name*>_writesample: Writes one sample to the converter.
- <*Converter name*>_writeblock: Writes a block of data to the converter.
- <*Converter name*>_close: De-allocates DSP resources used by this software.

Almost all drivers support an interrupt service routine (ISR) for block transfer; this routine can be found in this DCP file. The user must map this ISR. This process is described in Section 3.1. Furthermore, the driver source file can support auxiliary functions that are only used by standard functions. Therefore, these functions only have a file-level scope and are not exposed to the application layer.

2.7.4 Driver Header File (txxxx_fn.h)

This file is the header file for the driver source code file. It contains all prototypes for the driver functions, as well as the structure for the data converter object to be exposed to the application by including this header file with the application file. This object is filled in the source code file. Complex drivers may also have register unions and bit-field definitions in this file for easier and more understandable source code to facilitate changes in converter features.

The generated source files are automatically added to the project as soon as they are created.

3 Embedding the Driver in the Application

3.1 Project Environment

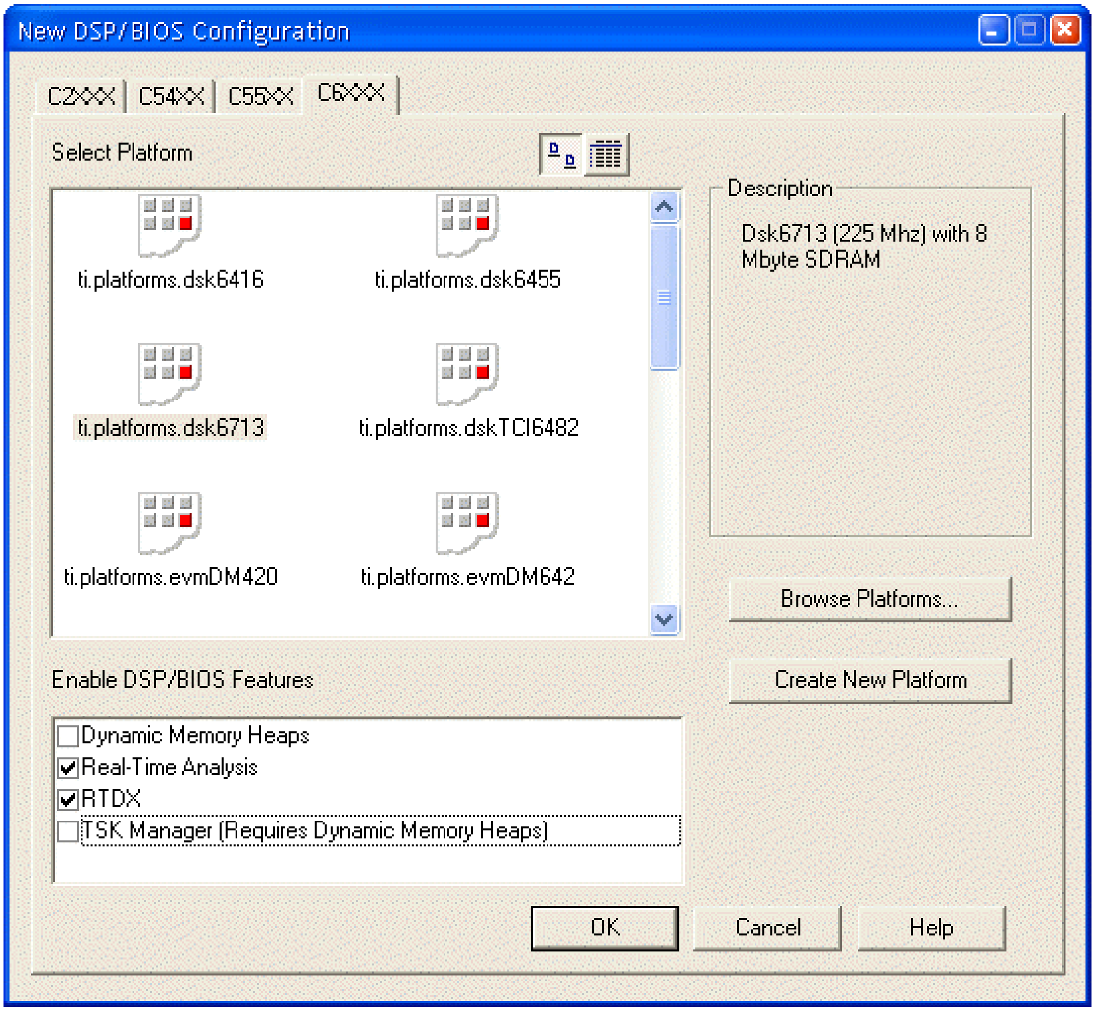
To set up a proper project environment, create a *Textual Configuration File* (TCF).

Action: CCS Menu → File → New… → DSP/BIOS Configuration...

A window with all available TCF seeds opens. Select the seed most similar to your system, as illustrated in Figure 9.

Action: uncheck the *Dynamic Memory Heap* and *Task Manager* check boxes; they are not used here.

Action: Select the seed and press *OK* to generate the file.



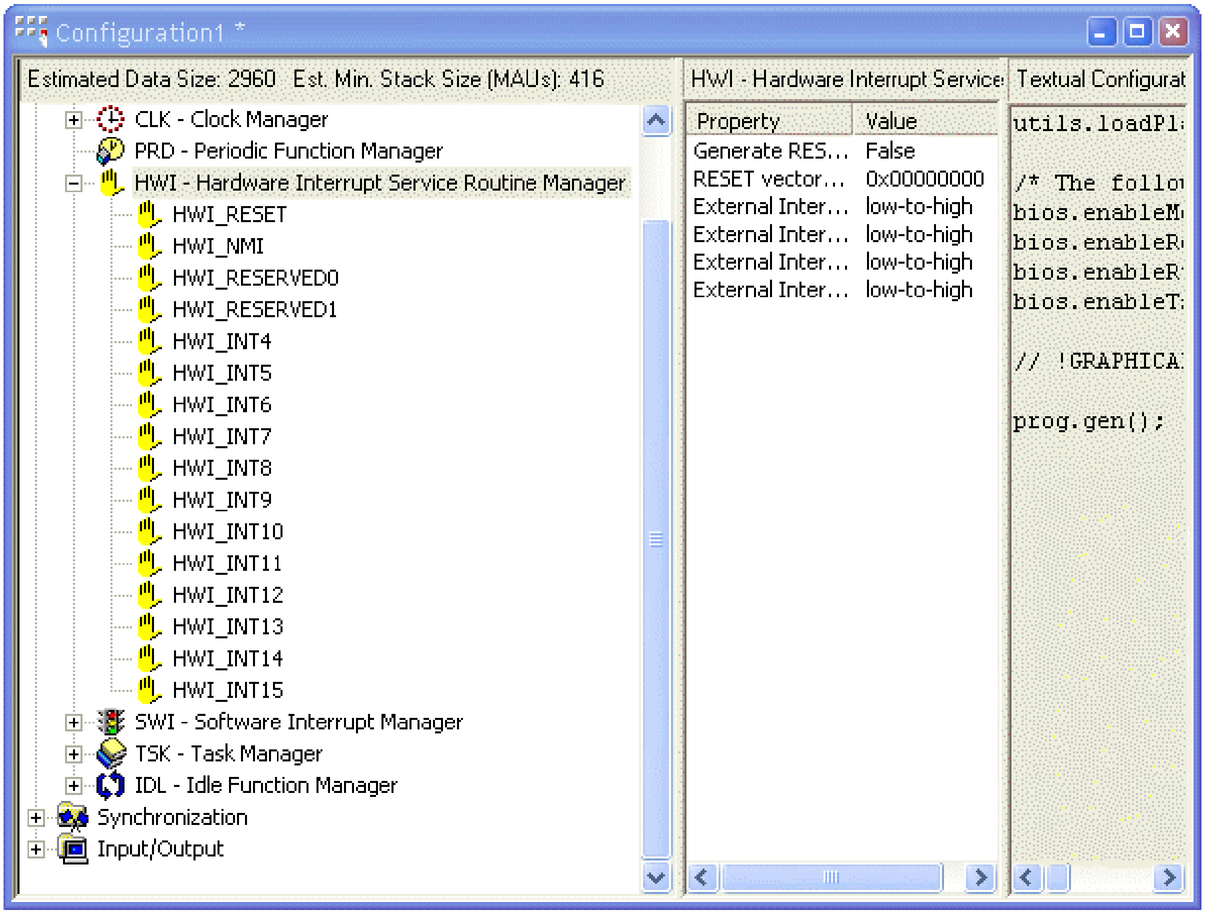
Dsk6713 (225 Mhz) with 8 Mbyte SDRAM

□ Dynamic Memory Heaps
☑ Real-Time Analysis
☑ RTDX
□ TSK Manager (Requires Dynamic Memory Heaps)

Figure 9. Seed Selection Window

Most drivers generated by the DCP use one or more interrupts; for example, the DMA interrupt for background data transfers on converters with a parallel interface, or the serial port receive interrupt for converters using the serial port. This interrupt must be registered in the TCF to generate an appropriate entry in the interrupt vector table.

Action: Open the Scheduling tree item in the newly-created TCF window. Next, expand the HWI–Hardware Interrupt Service Routine Manager item (see Figure 10).



Property	Value
Generate RES...	False
RESET vector...	0x00000000
External Inter...	low-to-high
External Inter...	low-to-high
External Inter...	low-to-high
External Inter...	low-to-high

```
utils.loadPl

/* The follo
bios.enableM
bios.enableR
bios.enableR
bios.enableT

// !GRAPHICA

prog.gen();
```

Figure 10. TCF File Main Window

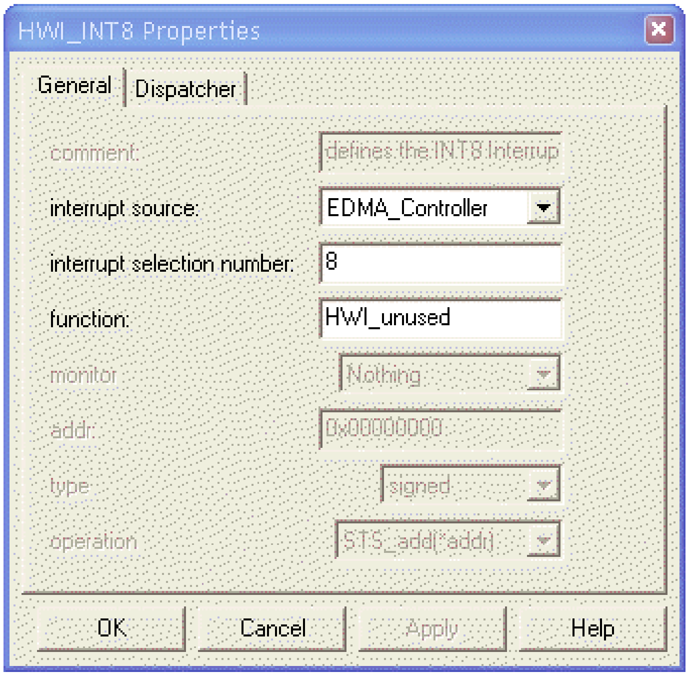
In order to register the ISR in the TCF, the name of the ISR must be known.

Action: Open the driver header file *(t<converter>_fn.h)* and find the prototype of the ISR. Copy the name of the ISR to the system clipboard (use <Ctrl+C>).

Depending on the interrupt source, select the correct HWI entry in the graphical configuration window. The TMS320C6000 family uses an interrupt-multiplex mechanism. As a result, every event can be mapped to every interrupt source. On all other DSP platforms, the interrupts are tied to their sources. Consult the product data sheet for your specific DSP to find the correct peripheral interrupt number.

Action: Right-click the appropriate HWI item (for example, *HWI_INT8*, the default EDMA controller interrupt in TMS320C6000 family, as shown in Figure 11), and click *Properties* on the resulting submenu. In the dialog, insert in the function field the name *_DCPDISP_dispatchEdmaIsr*, which is the DCP dispatcher ISR. This function can be found in file **tidc_api.c**.

Note: The DCP dispatcher is only available on the TMS320C6000 and TMS320C6400 DSP platforms.



comment: defines the INT8 Interrup

interrupt source: EDMA_Controller

interrupt selection number: 8

function: HWI_unused

monitor: Nothing

addr: 0x00000000

type: signed

operation: STS_add(*addr)

Figure 11. Interrupt Properties Dialog

The leading underscore is important because the ISR, written in the C language, is called from an assembler environment. More information on the TCF can be found in the *TMS320 DSP/BIOS User's Guide* (SPRU423).

Action: If you checked the *DSP/BIOS used* button in the DSP tab earlier, you must also check *use Dispatcher* in the *Dispatcher* tab.

Action: Save the TCF file to *Config.tcf* in your project directory

Saving the TCF file also generates the linker command file *(Configcfg.cmd)*.

Action: Add the files *Config.tcf* and *Configcgf.cmd* (generated during the TCF file save) to your project: Select *CCS Menu* → *Project* → *Add File to Project*… in order to get the project framework.

3.2 Call the Driver Function From Your Application Layer

The last step is to call the driver functions from your application layer. In this exercise, an example application source file is created.

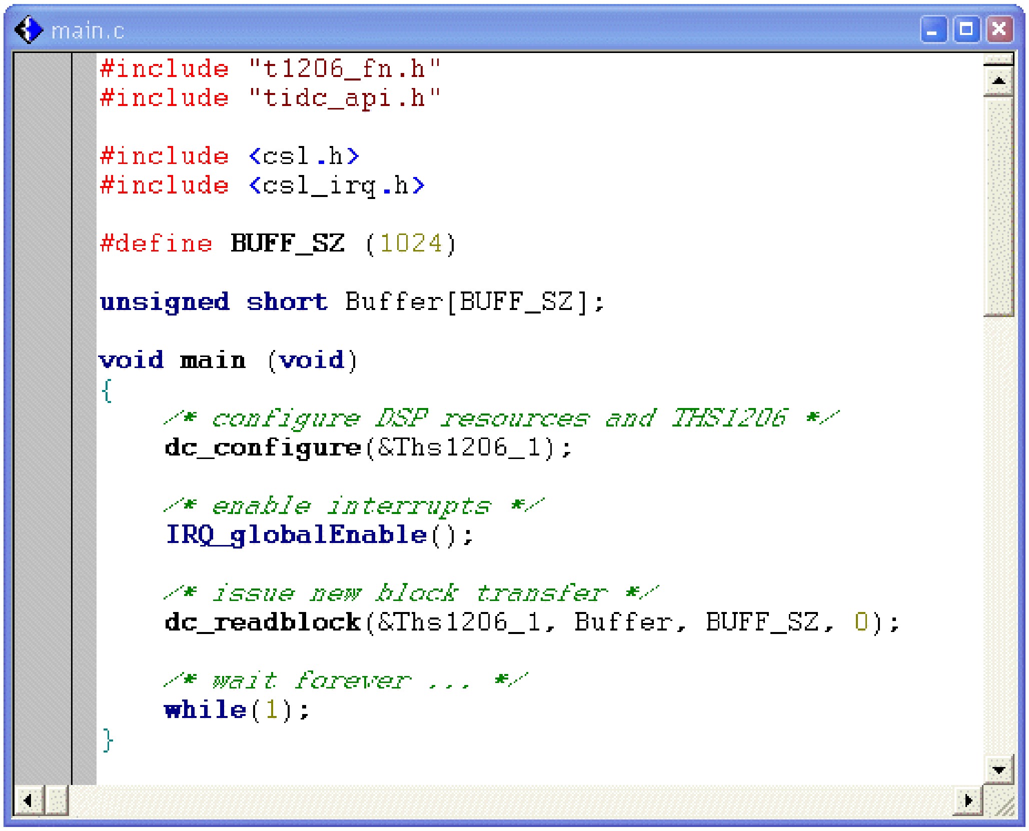
Action: Select *CCS Menu* → *File* → *New*. Save the file with the name *main.c* in the project directory. Add the file to the project.

The following steps are necessary to use the driver:

- Make the data converter object visible to the application layer by including the driver header file *(txxxx_fn.h)* in *main.c*.
- Make the common driver function visible to the application layer by including the file *tidc_api.h* in *main.c*.
- Create the program entry point—the *main()* function.
- Call *dc_configure()* in the *main()* body with the pointer of the data converter object as the parameter in order to initialize the converter.
- Enable the global interrupt bit. This step is not done in the driver in order to have more flexibility in the application layer. In some drivers, depending on the driver structure, this bit is already enabled in the driver source code.
- Call a data transfer function *(dc_readsample(), dc_readblock(), dc_writesample() or dc_writeblock())* to get data from the converter. The function used depends on the converter type you are using. For example, there is no such thing as using *dc_readblock()* on a DAC, because data are only written to the device.

The data converter object passed as a parameter to every function has the same name as the converter instance in the DCP (that is, the name of the converter tab).

A simple application layer accessing the driver functions might look like Figure 12.



```
#include "t1206_fn.h"
#include "tidc_api.h"

#include <csl.h>
#include <csl_irq.h>

#define BUFF_SZ (1024)

unsigned short Buffer[BUFF_SZ];

void main (void)
{
    /* configure DSP resources and THS1206 */
    dc_configure(&Ths1206_1);

    /* enable interrupts */
    IRQ_globalEnable();

    /* issue new block transfer */
    dc_readblock(&Ths1206_1, Buffer, BUFF_SZ, 0);

    /* wait forever ... */
    while(1);
}
```

Figure 12. Application Layer Example

3.3 Run the Program

Now, all that is left to do is to build the project.

Action: Click on *CCS Menu →Project→Rebuild All* to build the entire project.

If you did not change the project settings, a debug version of your application has been created. After the project build completes, the newly created, downloadable *DemoDCP.out* file is located in the *Debug* directory of your project.

Action: Select *Load DemoDCP.out* file to your target: *CCS Menu → File → Load Program…*

Action: Once the file is downloaded, press <F5> to run the application.

Run the application a few seconds; then stop the program by pressing <Shift+F5>. If samples are collected from the converter (when using either an ADC or codec), you can use a graph window (*CCS Menu → View → Graph → Time/Frequency…*) to display the buffer where the samples are stored. For more information on how to display data, see the *Code Composer Studio User's Guide* (SPRU328).

Now you have successfully created an application that communicates with a data converter. You can easily add your own signal processing function to the application layer to process the data.

4 References

These documents are available for download through the Texas Instruments web site (www.ti.com).

1. *Code Composer Studio User's Guide* (SPRU328)
2. *TMS320 DSP/BIOS User's Guide* (SPRU423)

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 TI 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. Information of third parties may be subject to additional restrictions.

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.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

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
RFID	www.ti-rfid.com	Telephony	www.ti.com/telephony
Low Power Wireless	www.ti.com/lpw	Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

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