Stratix II Professional FFT Co-Processor Reference Design

August 2005 version 1.0 Application Note 395

Introduction

The Fast Fourier Transform (FFT) co-processor reference design demonstrates the use of an Altera® FPGA as a high-performance digital signal processing (DSP) co-processor to the Texas Instruments TMS320C6000 family of programmable digital signal processors. The hardware interface is a connection between the TI digital signal processor's external memory interface (EMIF) and the first-in first-out (FIFO) buffers on the FPGA. The reference design utilizes TI's TMS320C6416 DSP Starter Kit (DSK), which features a TI TMS320C6416 device and the Altera Stratix® II DSP development board, which features an EP2S180F1020C3 FPGA.

For more information on the Stratix II DSP development board, refer to *Stratix II EP2S180 DSP Development Board Reference Manual*.

The reference design is supplied with Verilog HDL and TI DSP source code. Altera also supplies example software to demonstrate the use of the reference design.

Background

This section provides some background and describes some of the basic concepts for using FPGAs as a co-processor to a programmable digital signal processor.

FPGA Co-Processing

Programmable digital signal processors have been utilized in a wide range of signal processing applications. They have been designed with optimized instruction sets to execute digital signal processing algorithms like FFTs and finite impulse response (FIR) filters. Unfortunately, programmable digital signal processor performance has not kept up with the demands of the newest system applications, which often require dramatically higher data rates and increased channel counts. This has forced system designers to implement costly arrays of digital signal processors to satisfy these needs. However, these arrays tend to occupy more board real estate and require increased power consumption, which affects the overall system cost and poses significant implementation challenges, including the arbitration of shared memory between different processors.

Altera provides designers with the flexibility to implement an FPGA co-processor design that easily interfaces to a wide range of digital signal processors or general purpose processors (GPPs). This co-processor model can be adopted to fit virtually any target application because of the programmable nature of the FPGA's device fabric. Additionally, designers are able to customize and construct functions in a way that fully exploits the parallel nature of a hardware implementation within the FPGA, enabling power-efficient multichannel designs (useful in communication systems) with high data throughputs.

The following steps provide a high-level description of the FPGA co-processor design flow:

1. Profile applications in software to identify computationally intensive algorithms suitable for off-loading to co-processors.

2. Integrate an off-the-shelf co-processor like an Altera IP MegaCore® function or develop a custom co-processor block using a design tool like DSP Builder or using a hardware description language (HDL).

3. Evaluate co-processor system architectures and select a suitable processor interface.

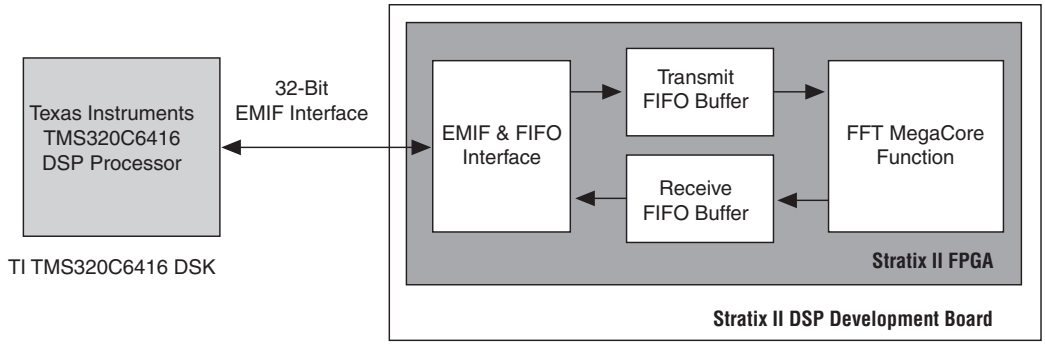
4. Integrate the hardware and software design components.

5. Verify the system in simulation and hardware.

FPGA Co-Processor Functional Description

Figure 1 illustrates the FFT FPGA co-processor reference design block diagram.

Figure 1. FFT FPGA Co-Processor Block Diagram



The direct memory access (DMA) controller within the TMS320C6416 processor transmits packets of data to be processed via the External Memory Interface (EMIF) and FIFO interface on the Stratix II EP2S180 FPGA. The EMIF and FIFO interface sends the data to the transmit FIFO buffer. The reference design monitors the fill level of the transmit FIFO buffer to determine when sufficient data is available for processing by the FFT MegaCore function.

The FFT MegaCore function processes the data in packets the size of the FFT transform length. The output of the FFT MegaCore function is sent to the receive FIFO buffer. When a whole packet of processed data is available to be read from the receive FIFO buffer, a DMA transfer request

is sent to the processor. Table 1 shows how the data packets are scheduled through the hardware system blocks to maximize system utilization and data throughput.

Table 1. Data Packet Scheduling Through the FFT CO-Processor Reference Design

Action	Step							
	1	2	3	4	5	6	7	8
EMIF & FIFO Interface	Write 0	-	Write 1	-	Read 0	Write 2	Read 1	Write 3
Transmit FIFO Buffer	-	In 0	-	In 1	-	-	In 2	-
FFT MegaCore Function	-	-	FFT 0	-	FFT 1	-	-	FFT 2
Receive FIFO Buffer	-	-	-	Out 0	-	Out 1	-	-

FFT MegaCore Function

The Altera FFT MegaCore function is high-performance and highly parameterizable. It has been optimized for the Stratix II, Stratix GX, Stratix, Cyclone™ II, and Cyclone device families. The FFT MegaCore function implements a complex FFT or inverse FFT (IFFT) for high-performance applications.

The FFT MegaCore function has the following features:

- Radix-4 and mixed radix-4/2 implementations
- Block floating-point architecture to maintain the maximum dynamic range of data during processing
- High throughput quad-output radix 4 FFT engine
- Support for multiple single-output and quad-output engines in parallel
- Multiple I/O data flow modes: streaming, buffered burst, and burst
- Parameterization-specific VHDL and Verilog HDL testbench generation
- Transform direction (FFT and IFFT) specifiable on a per-block basis
- Bit-accurate MATLAB models
- Optimized to use Stratix II, Stratix GX, and Stratix DSP blocks and the TriMatrix™ memory architecture
- Atlantic-compliant input and output interfaces (see "Atlantic Interface" below)
- Easy-to-use IP Toolbench interface
- IP functional simulation models for use in Altera-supported VHDL and Verilog HDL simulators
- Support for OpenCore® Plus evaluation
- DSP Builder ready

For more information on the FFT MegaCore function, refer to the *FFT MegaCore Function User Guide* at www.altera.com/literature/ug/ug_fft.pdf.

Table 2 shows the FFT MegaCore function parameters used in the reference design.

Table 2. FFT MegaCore Function Reference Design Parameters

Parameter	Setting
Target device family	Stratix II
Transform length	1,024 points
Data precision	16 bits
Twiddle precision	16 bits
FFT engine architecture	Quad output
Number of parallel FFT engines	1
I/O data flow	Buffered burst
Structure	4 multipliers and 2 adders
Implement multipliers in	Logic cells
Implement appropriate logic functions in RAM	Yes

Atlantic Interface

The FFT co-processor uses an Altera Atlantic I/O interface. The Atlantic interface is a flexible interface for high-throughput packet-based data transmission of arbitrary length. It provides a synchronous point-to-point connection between two blocks of logic with flexible flow control for master-to-slave and slave-to master directions.

For more information on the Atlantic interface, refer to *FS 13: The Atlantic Interface Functional Specification* at www.altera.com/literature/fs/fs_atlantic.pdf.

Transmit & Receive FIFO Buffers

The transmit and receive FIFO buffers handle the flow control of data to and from the FFT MegaCore function. The transmit FIFO buffer receives data from the processor across the EMIF interface and buffers an entire packet of data before sending it to the FFT MegaCore function.

As data becomes available on the output of the FFT, it is sent to the receive FIFO buffer. The receive FIFO buffer buffers an entire packet of data before sending it across the EMIF interface back to the processor.

The programmable thresholds in the transmit and receive FIFO buffers are set to a little less than the length of one packet. For example, for a 1,024-point 16-bit FFT, the threshold value is set at 1,022, because each packet requires 1,024 32 bit-words. This assumes that each FIFO word size is set to 32 bits. The FIFOs are set to a depth of 2,048 to avoid any overflow of data that might occur when writing to the FIFO buffers.

Processor Interface External Memory Interface (EMIF)

The EMIF provides a glueless interface to a variety of external memory components including synchronous dynamic random access memory (SDRAM), static random access memory (SRAM), and FIFO buffers. The EMIF is the connection method for the FPGA co-processor because of the data transfer rates available and the possibility of using the EDMA controller integrated within the TMS320C6416 processor. Additionally, the DSP Development Kit, Stratix II Professional Edition includes expansion headers that are compatible with the EMIF expansion headers on the TI TMS320C6416 DSK. Table 3 shows the peak data transfer rates achievable by the EMIF for the given clock rates.

Table 3. Peak EMIF Data Rates

EMIF Mode	Peak Transfer Rates (MBps)		
	@66 MHz	@100 MHz	@133 MHz
32-bit asynchronous *(1)*	53	80	106
32-bit synchronous	264	400	532
64-bit synchronous	528	800	1,064

Note to Table 3:
(1) Each asynchronous access is assumed to be 5 EMIF clocks.

The FPGA co-processor may be implemented as a memory mapped device using either a synchronous or asynchronous EMIF connection as determined by the system's performance requirements. FIFO buffers may be used to allow EMIF burst accesses to proceed without wait states, independently of the rate at which the co-processor consumes or produces data.

In this FFT FPGA co-processor reference design, the co-processor is connected to the processor via a 32-bit asynchronous EMIF. The co-processor appears within the digital signal processor's memory map in the chip select 3 address space. Transmit packets (from the processor to the FPGA) are written to the transmit FIFO buffer and receive packets (from the FPGA to the processor) are read from the receive FIFO buffer.

FIFO status signals are available to the processor (from the receive FIFO buffer) and to the FFT MegaCore function (from the transmit FIFO buffer).

Table 4 provides a description of the FFT FPGA co-processor reference design signals.

Table 4. FFT FPGA Co-Processor Reference Design Signal Descriptions (Part 1 of 2)

Signal Name	Width	Direction	Description
clk	1	I	Clock
rst_n *(1)*	1	I	Asynchronous reset, active low
soft_reset	1	O	Software controlled reset for co-processor
emif_ea [21:0]	22	I	EMIF address bus
emif_ce_n	1	I	EMIF device enable, device selected must be set up as asynchronous.
emif_be_n[3:0]	4	I	EMIF byte enable
emif_aoe_n	1	I	EMIF asynchronous output enable
emif_are_n	1	I	EMIF asynchronous read enable
emif_awe_n	1	I	EMIF asynchronous write enable
emif_ardy	1	O	EMIF asynchronous ready
emif_ed [31:0]	32	I/O	EMIF data
addr	1	O	Control port address
wdata	1	O	Control port write data
hwa_ce	1	O	Control port chip enable
write	1	O	Control port write strobe
hwa_rdata	1	O	Control port read data
tx_dma_evrq	1	O	Transmit DMA event request. Asserted low to request a new block of data to be encoded
rx_dma_evrq	1	O	Receive DMA event request. Asserted low to signal that a block of encoded data is available
tx_full	1	I	Transmit (software to hardware) FIFO buffer status
rx_full	1	I	Receive (hardware to software) FIFO buffer status
tx_dav	1	I	Atlantic master source (transmit) data available
tx_ena	1	O	Atlantic master source (transmit) enable
tx_dat [31:0]	32	O	Atlantic master source (transmit) data
tx_adr [3:0]	4	O	Atlantic master source (transmit) address
tx_eop	1	O	Atlantic master source (transmit) end of packet

Table 4. FFT FPGA Co-Processor Reference Design Signal Descriptions (Part 2 of 2)

Signal Name	Width	Direction	Description
tx_sop	1	O	Atlantic master source (transmit) start of packet
rx_ena	1	O	Atlantic master sink (receive) enable
rx_dat [31:0]	32	I	Atlantic master sink (receive) data
rx_eop	1	I	Atlantic master sink (receive) end of packet
rx_sop	1	I	Atlantic master sink (receive) start of packet
rx_err	1	I	Atlantic master sink (receive) error
rx_dav	1	I	Atlantic master sink (receive) data available
rx_val	1	I	Atlantic master sink (receive) data valid

Notes to Table 4:
(1) rst_n is reset.

Registers

Table 5 summarizes the registers present in the FFT FPGA co-processor reference design.

Table 5. FFT FPGA Co-Processor Reference Design Registers

Register Mnemonic	Address (Hex) *(1)*	Access	Description
TX_CREDIT	-	-	Transmit credit register—not directly accessible
RESET_HW	526C	Write	Sends a soft reset to the hardware
TX_CREDIT_INC	523C	Write	Increment TX_CREDIT register

Note to Table 5:
(1) The addresses are offsets from the FPGA base address of 0xB0080000.

Transmit Credit Register (TX_CREDIT)

The FPGA co-processor must signal a DMA event to the EDMA to trigger the writing of packets in the transmit direction. This action is ultimately under control of the processor, which must write to the TX_CREDIT register within the FPGA co-processor to give the co-processor permission, or credit, to process a packet, for example, once the associated transmit and receive buffers have been allocated in memory. When the FPGA co-processor has one or more transmit credits, it requests a transmit DMA packet whenever the EMIF interface is idle. The TX_CREDIT register is decremented when a packet is moved from the transmit FIFO buffer to the FFT MegaCore function. The TX_CREDIT register is not directly accessible.

Soft Hardware Reset Register (RESET_HW)

A soft hardware reset is performed prior to initializing the chip support Library, the EDMA, and the DMA receive interrupt service routine. This ensures all registers are at a known reset state prior to initialization. The soft hardware reset is done once within the entire example C routine included with the reference design.

Transmit Credit Increment Register (TX_CREDIT_INC)

Table 6 shows the transmit credit increment register format.

Table 6. Transmit Credit Increment Register Format

Data Bit	Mnemonic	Description
0	S	A write with the **S** bit asserted causes a soft reset of the FPGA co-processor
1	I	A write with the **I** bit asserted causes the TX_CREDIT register to be incremented
31:2	0	Always write **0** for future compatibility

Software Access to the FFT Co-Processor

The reference design contains example C code that generates the input data stimuli and demonstrates how blocks of data are streamed through the FFT co-processor. The software project is built using the digital signal processor and BIOS libraries included with the TI Code Composer Studio (CCS) software to configure the EDMA controller and interrupts.

Two DSP general-purpose I/O (GPIO) pins are dedicated for use as event triggers for the EDMA: one for transmit data (digital signal processor to co-processor) and one for receive data (co-processor to EDMA). The co-processor requests a new transmit DMA whenever the TX_CREDIT register is non-zero and the FFT MegaCore function is ready for more data. It requests a receive DMA whenever a packet of data is available from the FFT MegaCore function in the receive FIFO buffer.

Each time a DMA is completed, the EDMA sends an interrupt request to the digital signal processor. The software tracks the number of packets transmitted and received. When a predefined number of packets are complete, the software calculates the performance of the FFT co-processor.

Apart from a software reset and incrementing the TX_CREDIT register, all accesses to the FPGA co-processor are performed by the EDMA controller. In the reference design the EDMA is set up by calling the initEdma() function in the **fft_ping_pong.c** source file.

The example software `main()` routine is in the **fft_ping_pong.c** source file included with the reference design. `main()` performs the following tasks:

- Sets up `timer0` for performance measurement
- Initializes the memory buffers with the sine wave data for the EDMA
- Resets the FPGA co-processor and FIFO buffers
- Initializes the TI chip support library
- Calls `initEdma()` to initialize the EDMA controller
- Starts the timer
- Increments `TX_CREDIT`
- Waits until all blocks are processed
- Calculates average time to process one FFT

Each time a transmit or receive DMA interrupt occurs, `edmaHwi()` is called to handle the interrupt. This function is in **fft_ping_pong.c**. Counters are updated to track the number of transmit and receive interrupts received until a predefined limit is reached, at which point the EDMA and interrupts are disabled. The transmit interrupt handler increments the `TX_CREDIT` register to allow further blocks to be processed.

No further processing of the data is performed by the example software.

Figures 2 to 5 show various plots of the FFT input and output data using the CCS graphical display utility. Figure 2 shows a plot of the real input data in CCS.

The sampled sine wave continues for 1,024 samples. The imaginary input samples are all zero.

Figure 2. Real Input Data Plot

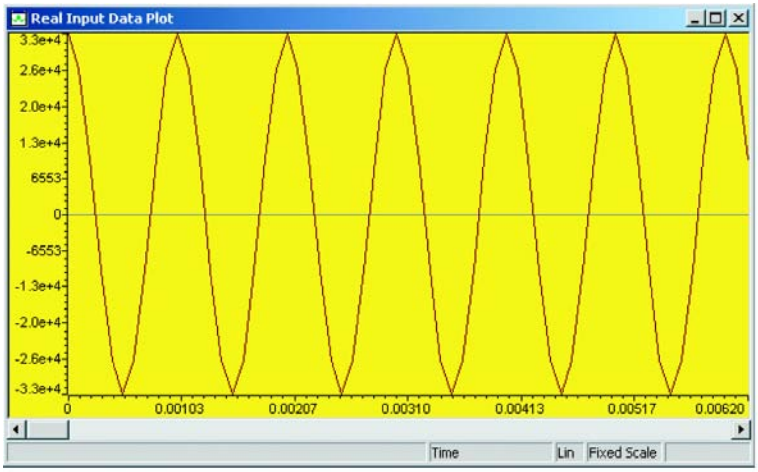


Figure 3 shows a plot of the real output data in CCS.

Figure 3. Real Output Data Plot

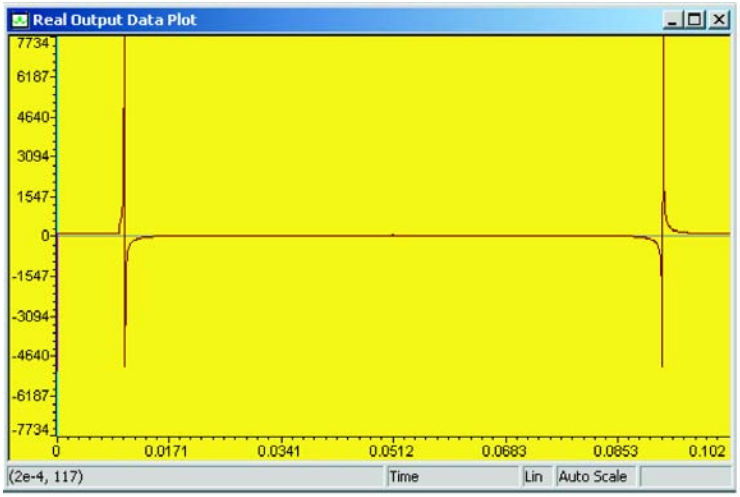


Figure 4 shows a plot of the imaginary output data in CCS.

Figure 4. Imaginary Output Data Plot

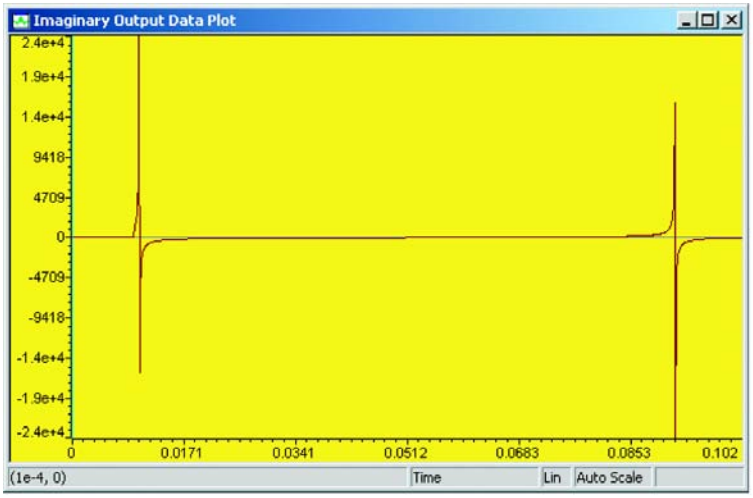
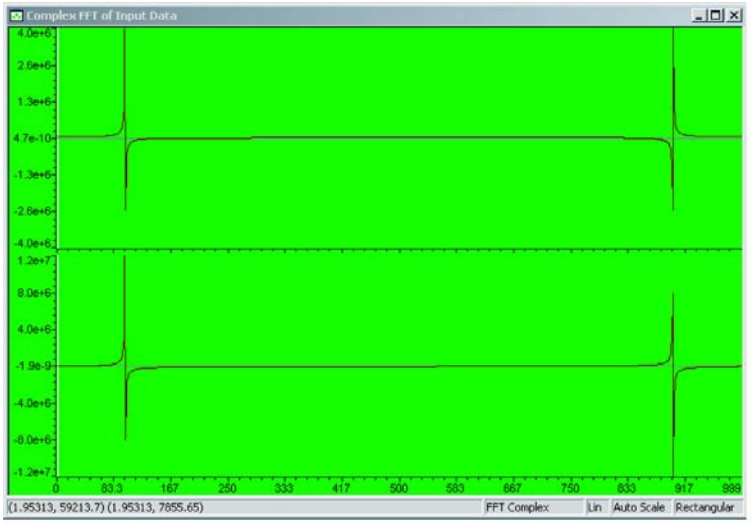


Figure 5 shows the CCS plot of a complex FFT applied to the input data implemented entirely in C code.

Figure 5. FFT Plot of the Input Data in CCS



Performance Improvements

This section discusses the performance analysis and improvement suggestions for the reference design.

Enhancements

Several enhancements to the solution presented in this reference design can be considered if the user is starting a new board-level FPGA co-processor design. First, both the TI digital signal processor and the FPGA can be integrated on the same board. Care must be taken when routing board-level interconnections between the TI digital signal processor and the FPGA to ensure that the maximum data throughput and clock rate of the EMIF can be leveraged to reduce the round-trip data delay time. The current C64x family of digital signal processors can support up to 64-bits of EMIF data at clock rates up to 133 MHz.

The use of FIFO buffers in the transmit and receive paths allow the FFT co-processor (or any other co-processor function) to run at a different clock rate than the EMIF. If the co-processor function is a performance bottleneck, then the co-processor function may be run at a higher clock rate than the EMIF interface to decrease the overall processing time. Alternatively, the co-processor may be reconfigured to exploit further parallelism of the FPGA. If system issues other than the co-processor are causing the performance bottleneck, then the co-processor clock rate could be reduced to lower the dynamic power consumption in the FPGA.

The FFT co-processor in this reference design is a relatively simple example. For larger co-processing systems that may consist of several co-processor functions, design considerations need to be made to maximize the data processing within the FPGA. The larger co-processing reduces the data-transfer overhead between the digital signal processor and the FPGA relative to the data processing time, thereby maximizing the overall system throughput performance.

Alternative high-speed interfaces — for example, Serial Rapid I/O, or SerialLite can be considered to improve the high-speed data transfers between the digital signal processor and the co-processor, further increasing overall system performance.

Getting Started

This section describes the software and hardware requirements, as well as the required steps to install the reference design files, program the DSP development board, and load the TI C64x binary executable program.

Software Requirements

The reference design requires the following software:

- Quartus® II software version 5.0 Service Pack 1 or higher
- Altera FFT MegaCore function version 2.1.3
- Texas Instruments Code Composer Studio version 2.21 or higher

Hardware Requirements

To run the FFT FPGA co-processor reference design you must have the following development kits:

- Stratix II EP2S180 DSP development kit
- Texas Instruments TMS320C6416 DSK

Connect both boards to their power supplies and programming cables according to the instructions listed in their respective data sheets and reference manuals.

For more information on the DSP Development Kit, Stratix II Professional Edition, refer to the *Stratix II EP2S180 DSP Development Board Reference Manual*.

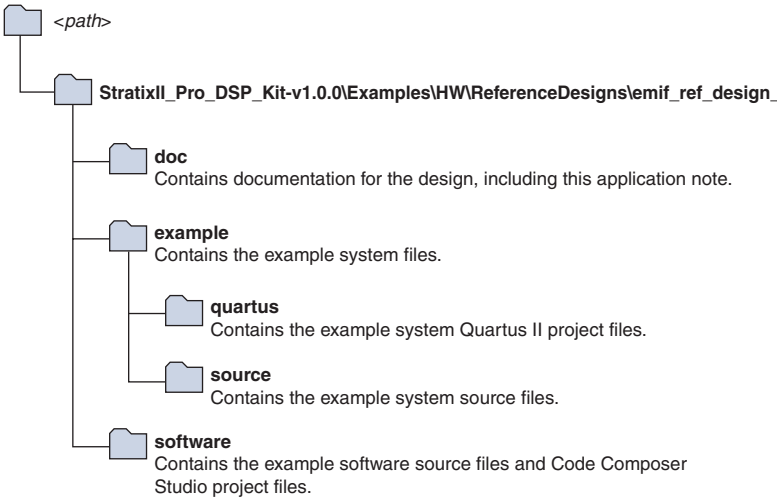
For more information on the Texas Instruments TMS320C6416 DSK, refer to the *TMS320C6416 DSK Technical Reference* by Spectrum Digital.

Connect the EMIF of both boards so that the J31 and J33 male headers on the bottom of the Stratix II DSP board plug into the J3 and J4 female headers on the TI TMS320C6416 DSK, respectively. The Stratix II EP2S180 DSP development board should be on top of the TI TMS320C6416 DSK.

Install the Reference Design Files

Install the FFT FPGA co-processor reference design from the *DSP Development Kit, Stratix II Professional Edition 1.0.0 CD-ROM*. The installation is an automated process. Figure 6 shows the directory structure of the reference design, where <*path*> is the directory where the DSP Development Kit, Stratix II Professional Edition files were installed.

Figure 6. FFT FPGA Co-Processor Reference Design Directory Structure



Program the Stratix II DSP Development Board

The compilation of the FFT co-processor design produces a **.sof** (SRAM object file) that is used to program and configure the EP2S180 Stratix II device. The **.sof** file has already been created and is pre-installed in this reference design.

Follow these steps to program the Stratix II EP2S180 device:

1. Start the Quartus II software.
2. Choose **Open Project** (File menu) and browse to the <*path*>**\StratixII_Pro_DSP_Kit-v1.0.0\Examples\HW\ReferenceDesigns\emif_ref_design_FFT\example\quartus** directory.
3. Choose **emif_ref_ex.qpf** and click **Open**.

4. If a message appears asking whether you want to overwrite the database written with a prior or different version of Quartus II, click **OK**.

5. From within the Quartus II software, open the **Quartus II Programmer** (Tools menu).

6. The **emif_ref_ex.sof** is automatically detected by the **emif_ref_ex.cdf** (Configuration Design File) and loaded into the Quartus II software. If it does not load the file, click **Add File**, browse to the *<path>***\StratixII_Pro_DSP_Kit-v1.0.0\Examples\HW\ReferenceDesigns\emif_ref_design_FFT\example\quartus** directory, and select the **emif_ref_ex.sof** file. Click **Open** (see Figure 7).

7. Ensure that the appropriate programming hardware is selected in the **Quartus II Programmer** window. If it is not, click **Hardware Setup**. Select the appropriate hardware from the list (either **ByteBlaster II** or **USB Blaster**) and click **Select Hardware**. Click **Close** (see Figure 7).

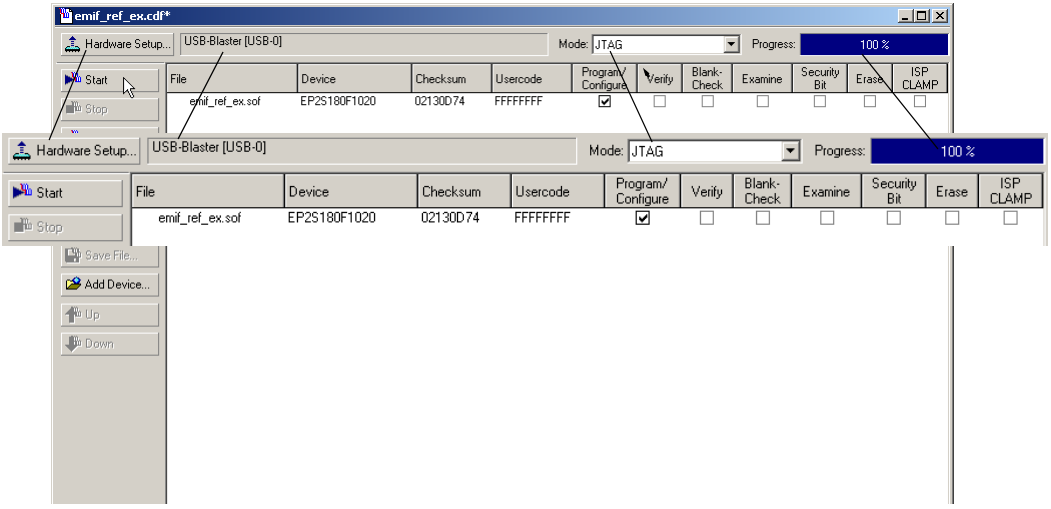
For details on installing the USB Blaster software driver on the host PC (located at *<quartus_install_dir>***\drivers\usb-blaster**), see the *USB-Blaster Download Cable User Guide.*

8. Ensure that the **Program/Configure** box is **checked** (see Figure 7).

9. Click **Start** to begin programming or configuring the Stratix II DSP development board (see Figure 7).

10. Programming and configuration is complete when the **Progress** bar indicates 100%. The Quartus II software reports `Info: Ended Programmer operation at` *<date, time>* and the CONF_DONE LED (LED5) on the DSP development board is illuminated.

Figure 7. Quartus II Programmer



Run the Example Software

To run the example software in Code Composer Studio, follow these steps:

1. Ensure that the TI TMS320C6416 DSK's USB cable is connected between the PC and the development board.

2. Start **Code Composer Studio**. The code Composer Studio window appears (see Figure 8).

3. Ensure that the correct GEL file is loaded for the TI TMS320C6416 DSK under GEL files in the CCS Project window. To load the appropriate GEL file, right click on **GEL files** and select **Load GEL**. Browse to the **<***CCS installation directory***>\cc\gel** directory. Select the **DSK6416.gel** file and click **Open**.

4. Choose **Project** > **Open** (Project menu). Browse to the *<path>***\StratixII_Pro_DSP_Kit-v1.0.0\Examples\HW \ReferenceDesigns\EMIF_ref_design_FFT\software\64xx** directory and select the **fft_ping_pong.pjt** file.

5. If a message appears saying the **csl6416.lib** file cannot be found, click **Browse** to browse to the **<***CCS installation directory***> \c6000\bios\lib** directory and select the **csl6416.lib** file. Click **Open**.

6. Choose **Workspace** > **Load Workspace** (File menu) to open the **fft_ping_pong.wks** file. This opens a saved CCS workspace that allows you to view the FFT input and output data plots in the CCS user interface. Disregard any Code Composer Studio messages that refer to insufficient resources or GEL file location.

7. Choose **Resets**>**ClearBreakPts_Reset_EMIFset** (GEL menu). Disregard any Code Composer Studio messages that refer to start address identifiers.

8. Choose **Memory Map>SetMemoryMap** (GEL menu).

9. To load the design onto the TI TMS320C6416 DSK, choose **Load Program** (File menu). Browse to the **debug** directory and select the **fft_ping_pong.out** file. Click **Open**.

10. Choose **Enable Clock** (Profiler menu) so that the profiler can keep count of the number of clock cycles it took to run the FFT function.

    You may want to choose **Enable Clock** again in the profiler menu to ensure that there is a check mark beside **Enable Clock**.

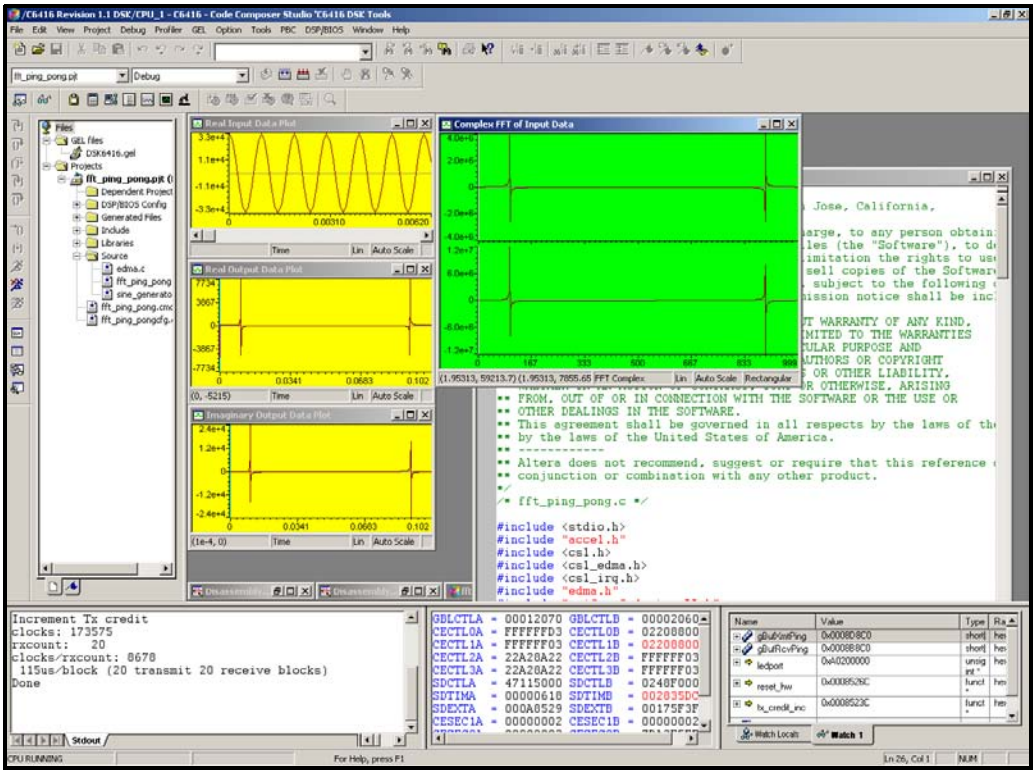
11. Choose **Run** (Debug menu) to run the software on the TI TMS320C6416 DSK.

12. As the design runs, messages appear in the Stdout window. They show the status of the program and the number of clock cycles it take to run the FFT co-processor function.

13. The program prints **Done** in the stdout window as the last message, indicating that the program has executed and is complete.

14. The graphs in the CCS user interface show the various input and output FFT plots (see Figure 8). If the signals do not appear appropriately, right-click in the respective graph windows and click **Refresh** to update the signals.

Figure 8. Code Composer Studio Workspace Graphs & User Interface



Optional: Compile the FFT Co-Processor Design

To compile the FFT FPGA co-processor reference design, you must first install the FFT MegaCore function. The function can be run with a full license or using free OpenCore Plus evaluation.

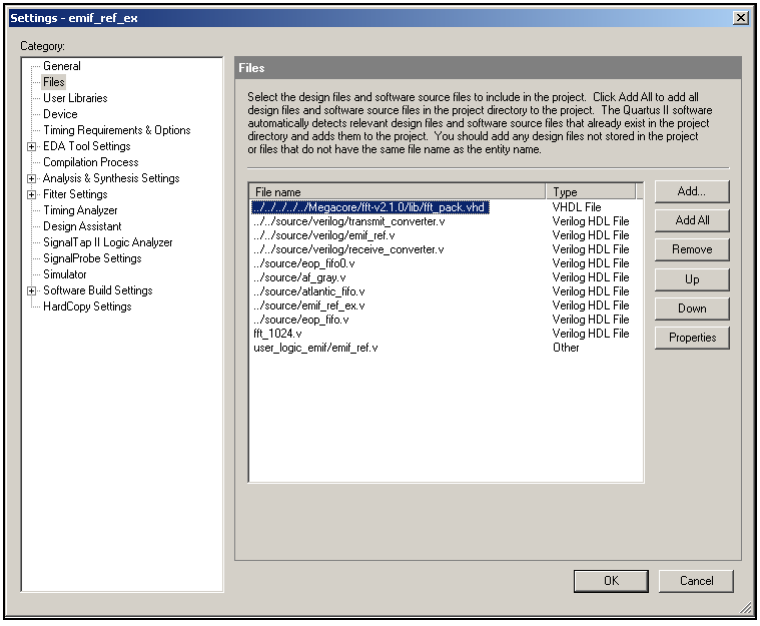
For more information on OpenCore Plus, refer to *AN 320: OpenCore Plus Evaluation of Megafunctions*.

The **example\quartus** directory contains a Quartus II project (**emif_ref_ex.qpf**) that contains all the necessary files required for the project as well as the pin mappings for the DSP Development Kit, Stratix II Professional Edition.

If you need to modify and recompile the design, perform the following steps:

1. Start the Quartus II software.
2. Choose **Open Project** (File menu) and browse to the *<path>***StratixII_Pro_DSP_Kit-v1.0.0****Examples****HW****ReferenceDesigns****example****quartus** directory.
3. Choose **emif_ref_ex.qpf** and click **Open**.
4. If a message appears asking whether you want to overwrite the database written with a prior or different version of Quartus II, click **OK**.
5. Choose **Add/Remove Files** in **Project** (Project menu) and verify that the **fft_pack.vhd** file for the FFT MegaCore function v2.1.0 is pointing to the correct location (see Figure 9).
6. If your **fft_pack.vhd** file is not in the same location listed in the project, remove the existing **fft_pack.vhd** file by selecting it and click **Remove**. To add the appropriate **fft_pack.vhd** file, browse to the *<fft-v2.1.0 installation directory>*/**lib**/ directory, select **fft_pack.vhd** and click **Add** (see Figure 9).

Figure 9. Verifying the fft_pack.vhd File in the Setting Window



7. Choose **Start Compilation** (Processing menu) to begin compiling the design.

8. Click **OK** in the message window that appears.

101 Innovation Drive
San Jose, CA 95134
(408) 544-7000
www.altera.com
Applications Hotline:
(800) 800-EPLD
Literature Services:
literature@altera.com