Cyclone V E Accelerated FIR with Built-in Direct Memory Access Example

Author: Isaac Kruger

Date: 9/11/15

Revision: 1.0

©2015 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

Table of Contents

Hardware Specifications

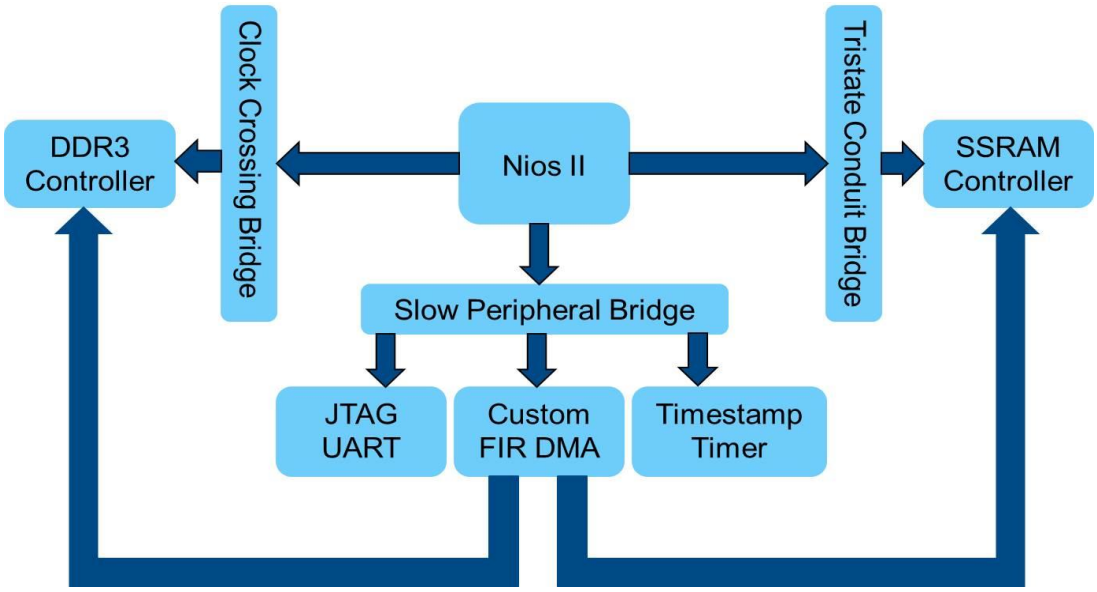
This design makes use of several peripherals on the Cyclone V E FPGA Development Kit as well as several IP components. A full list is below:

- Clock source
- Interval timers
- Performance counter unit
- JTAG UART
- Button/LED PIO (implemented in the Qsys design, but not used in this example)
- Custom FIR-DMA (described in the "Theory of Operation" section of this document)
- Avalon-MM Pipeline Bridges
- Avalon-MM Clock Crossing Bridges
- Nios II Classic Processor
- Tri-State Conduit Bridge
- Tri-State Conduit Pin Sharer
- Generic Tri-State Controller (for SSRAM and flash memory, although flash memory is not used in this example)
- Clock Bridges
- DDR3 SDRAM Controller with Uniphy
- Altera PLL
- On-Chip RAM

This design was implemented in Quartus II 15.0.

A block diagram is below. Note that this block diagram is highly simplified and leaves out several peripherals and intermediate pipelining bridges. It is meant to highlight the most "important" parts of the hardware and how they are connected. A more detailed block diagram of the Custom FIR DMA block is included in the "Theory of Operation" section of this document.

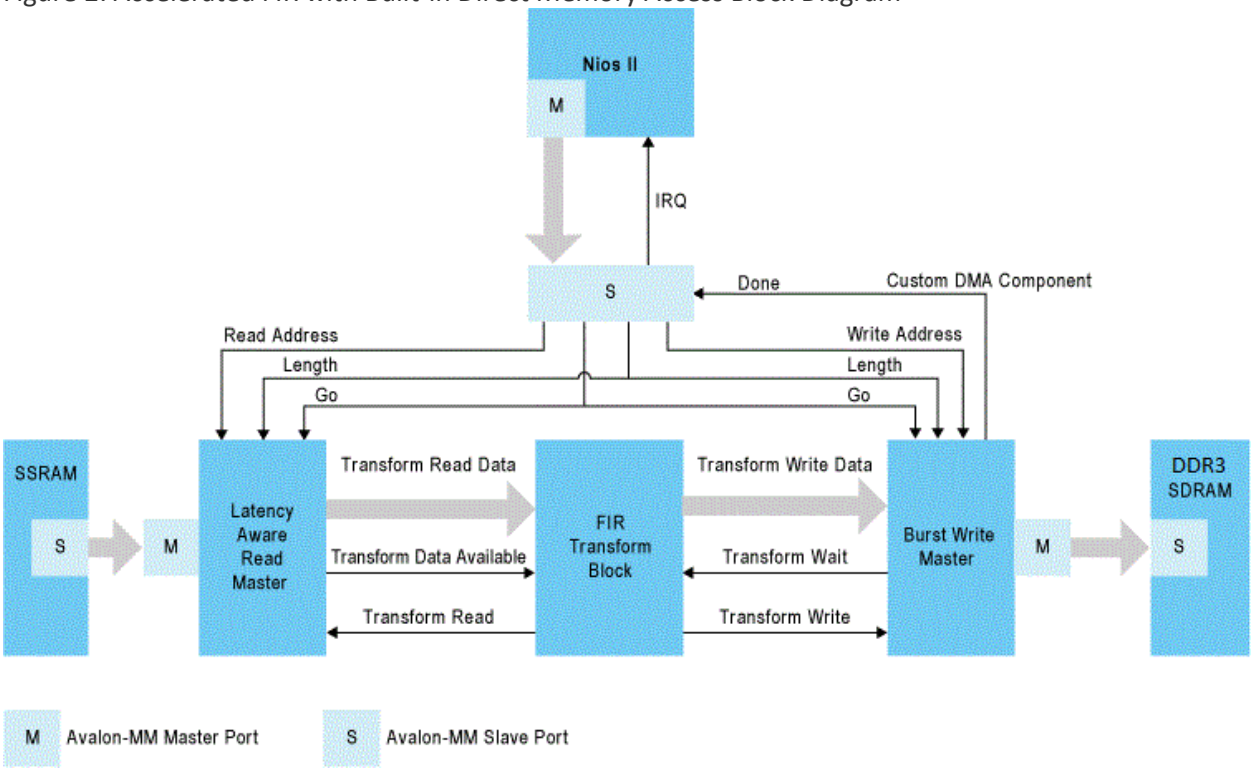
Figure 1: Simplified Hardware Block Diagram



Theory of Operation

This design example implements a FIR filter with built-in direct memory access on a Cyclone V E FPGA Development Kit. The finite impulse response (FIR) filter is a common algorithm used in digital signal processing (DSP) systems. In this example, a FIR filter has been integrated into a single Qsys component containing Avalon® Memory-Mapped (Avalon-MM) read and write masters. The read master is responsible for supplying the filter with input data (from the SSRAM in this example), while the write master is responsible for writing the filter response back to memory (the DDR3 SDRAM in this example). Since the filter has Avalon mastering capabilities, you do not need to use a separate direct memory access (DMA) engine to accomplish the filter operation. Below is a diagram of the Custom FIR DMA Qsys component.

Figure 2: Accelerated FIR with Built-in Direct Memory Access Block Diagram



When a filter is implemented in software, it requires many clock cycles to complete the calculation of a single output. Using an FPGA, all of these operations can occur concurrently with up to one output calculated every clock cycle. You can implement computationally complex algorithms in hardware to:

- Increase the overall system performance
- Offload the Nios II embedded processor so that it can perform other tasks
- Decrease the overall design frequency to reduce power consumption

The purpose of this design example is to illustrate the accelerating effect of performing computationally complex operation in hardware instead of software. Furthermore, although this design performs filter operations, you can also reuse the accelerator for your own data transforms, as described in the "Repurposing the FIR DMA Component" section of this document.

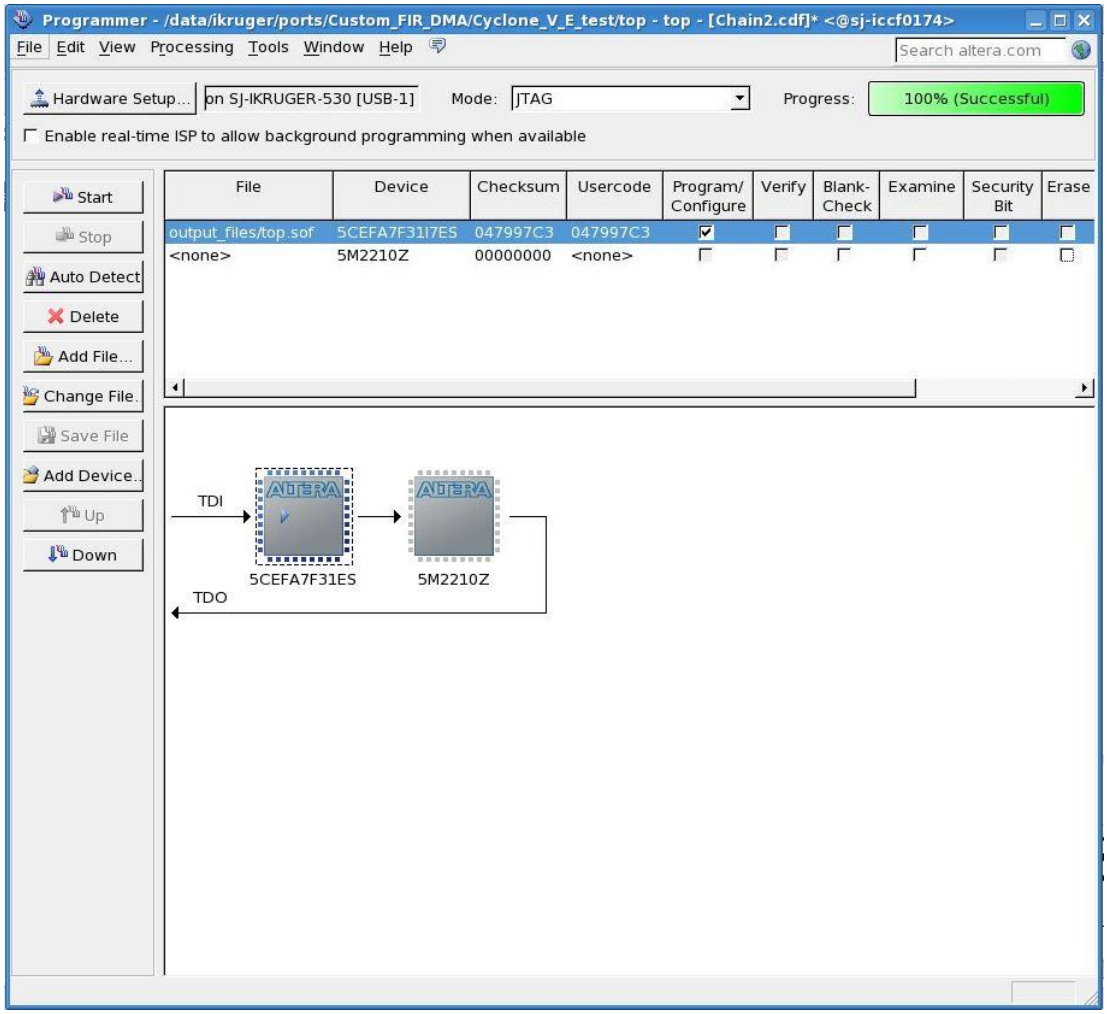
Executing the Example Design

Compiling and Programming the Hardware

Follow the steps on the Design Store web page to extract and install the Custom_FIR_DMA_Cyclone_V_E platform file. The following steps describe how to setup a project in the Quartus II 15.0 software in order to program the Cyclone V device with the design example.

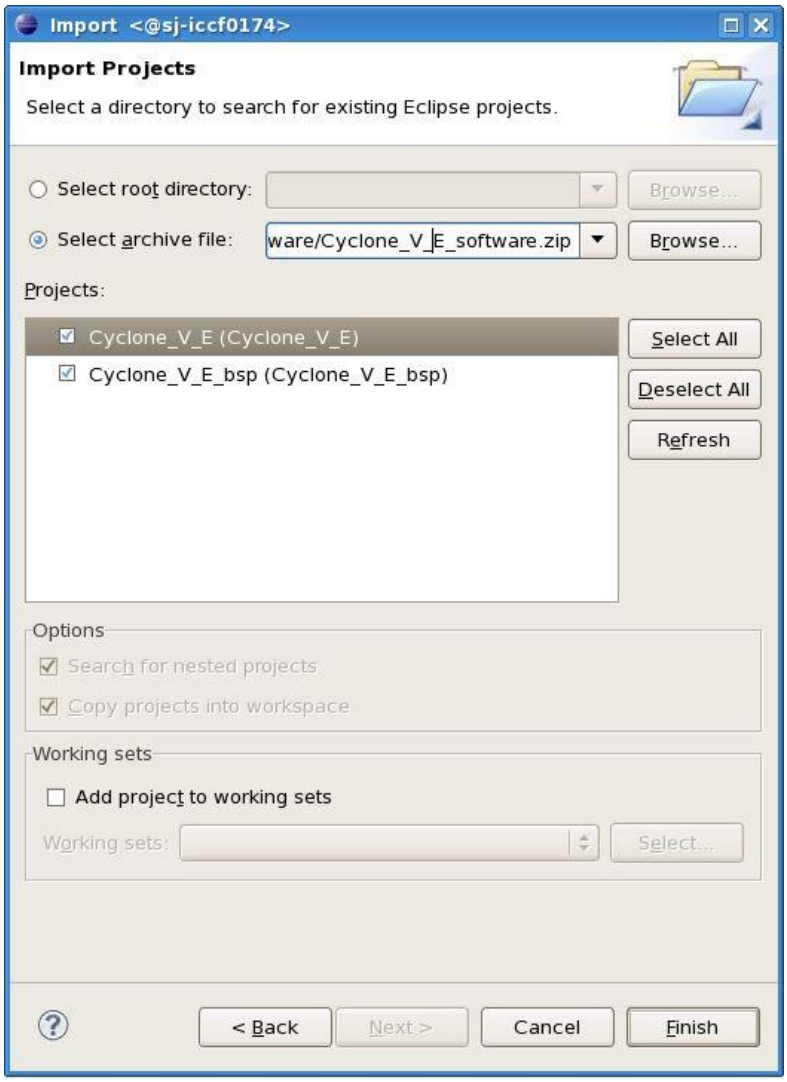
1. Launch Quartus II and open the top.qpf project file by clicking File → Open Project.

2. Ensure that the settings of the project match your device. By default, the project is set to program device numbers 5CEFA7F31C7ES or 5CEFA7F31I7ES. If you have a different device number (in particular is you have a non-“ES” device number) you must first modify the project settings. Click Assignments → Device and find the correct device from the list at the bottom. If this is set incorrectly, the project will still compile but the device may not program.
3. Compile the project by clicking Processing → Start Compilation.
4. Once the project is done compiling, launch the Quartus II Programmer from the tools menu by clicking Tools → Programmer.
5. Plug in your development kit and make sure the appropriate hardware (usually “USB-Blaster II”) is displayed next to the “Hardware Setup…” button. If it is not, click on the “Hardware Setup…” button and select the correct hardware.
6. Once the hardware is selected, click “Auto-Detect”. If a dialog box appears asking for which device is on the JTAG chain, select the option that matches your device number. Make sure that if you have an “ES” device (ie, your device number ends with “ES”) you select a device that also ends in “ES”.
7. Double click on the File list next to the device you just selected where it says “<none>”. Navigate to the output_files subfolder of the project folder and select the file top.sof.
8. Check the “Program/Configure” box next to the device you selected in part 5. Then click “Start” to program the device. The software will also need to be compiled and downloaded to the Nios II processor, as described in the next section.



Compiling and Downloading the Software

1. In Quartus II, click Tools → Nios II Software Build Tools for Eclipse. This will open Eclipse. Use the "Browse…" button to choose a folder to be your workspace. This can be any folder, including the software subfolder within the Quartus project folder.
2. Click File → Import. Under the "General" tab select "Existing Projects into Workspace" and then click "Next". Click the radio button next to "Select archive file:" to select it. Click the "Browse…" button and navigate to the Quartus project folder. Enter the software subfolder, select the file Cyclone_V_E_software.zip, and click OK.



3. Make sure both Cyclone_V_E and Cyclone_V_E_bsp are checked and then click Finish.
4. Once the projects have been imported, you can open up the example code by opening “main.c”, “hw_fir.c” and “sw_fir.c” in the Cyclone_V_E project.
5. Right-click on Cyclone_V_E_bsp in the Project Explorer. Click Nios II → Generate BSP.
6. Build the project by clicking Project → Build All.
7. Click Run → Run Configurations. Double click on “Nios II Hardware” at the left of the window that pops up to create a new configuration. In the “Project name:” drop down menu, select Cyclone_V_E. Click on the “Target Connection” tab and ensure that the same hardware you selected when programming the device is listed under both “Processors:” and “Byte Stream Devices:”. If it is not, click “Refresh Connections” at the right of the window.
8. Once all settings are correct, click “Run” at the bottom of the window. For future runs, you can skip step 5 by right-clicking the Cyclone_V_E project in the Project Explorer and clicking Run As → Nios II Hardware. Step 5 must be performed every time the development kit is disconnected or power cycled.

Behavior

Upon running, the program will generate a source buffer of sample data and store it in the kit's external SSRAM. It will then perform a filtering operation in two ways (in software and in hardware), and store the results of both operations in the kit's external DDR3 SDRAM. First, it will perform the operation in software, using the Nios II processor. Then it will perform the operation in hardware, using the custom FIR-DMA Qsys component. Both operations will be timed using the timestamp timer. The results of both operations will be validated/compared to ensure that they match- if they do then the program will display the processing time of both operations, and the hardware-versus-software speed-up factor.

The program will display several status updates in the Nios II Console. Updates will be printed to the console upon writing the test data, starting and finishing both the software and hardware filtering operations, and validating the results. Error messages will be printed if any of these steps cannot be completed successfully or correctly.

Repurposing the FIR DMA Component

The custom FIR DMA component can be repurposed to implement your own data transform. The source files for the component are located in the subfolder dma within the Quartus project folder. Open the Verilog file **transform_block.v**. This block joins the FIFO interfaces for the read and write master blocks, and instantiates the custom FIR module (which is defined in **custom_FIR.v**). **transform_block.v** can be edited to implement your own custom logic, and **custom_FIR.v** can be edited or replaced entirely by your own custom module. The header of **transform_block.v** contains more information on how to implement your own transforms.

After making modifications to the DMA source files, it is necessary to regenerate the Qsys HDL. Open Qsys by clicking Tools → Qsys. In Qsys, click File → Open, navigate to the project folder, and open Cyclone_V_E_qsys.qsys. Finally, click Generate → Generate HDL. Then recompile the project and reprogram the device, as described in the "Compiling and Programming the Hardware" section of this document.

In software, the function *hw_fir.c* controls the transfer of data to and from the DMA. This function can be reused without major modifications, as can the header file *dma_avalon_access.h*. The function *sw_fir.c* would need to be rewritten to implement the new data transform if you still want to be able to compare the speeds of hardware vs software implementations. The function *main.c* is responsible for generating sample data, calling *hw_fir.c* and *sw_fir.c*, managing the timers, and printing status updates to the console. This function can also be modified to change the data in the buffer or to change the general flow of the program.