CARDSHARP

User's Manual

Table of Contents

List of Tables

List of Figures

Introduction

Real Time Solutions!

Thank you for choosing Innovative Integration, we appreciate your business! Since 1988, Innovative Integration has grown to become one of the world's leading suppliers of DSP and data acquisition solutions.

Our products offer a wide range of solutions for demanding signal processing and data acquisition applications that integrate high performance DSPs, FPGAs and IO. To enhance your productivity, our hardware products are supported by comprehensive software libraries and device drivers providing optimal performance and maximum portability.

Innovative Integration's products employ the latest digital signal processor technology thereby providing you the competitive edge so critical in today's global markets. Using our powerful data acquisition and DSP products allows you to incorporate leading-edge technology into your system without the risk normally associated with advanced product development. Your efforts are channeled into the area you know best ... your application.

Scope of this User Guide

The *Cardsharp Framework Logic Manual* provides support to logic developers for the Cardsharp family products. The *Guide* provides design information to assist developers in the addition of new functionality to the logic, from the creation of the logic through to the implementation.

The Guide shows Cardsharp product logic firmware in detail and explains how to use this logic for your development. The Guide shows the overall structure of each design, shows the standard registers and memory map, discusses the details of how to modify the logic, and finally presents an overview of each component in a library section. Source code for most components is delivered with your Cardsharp product along with a Xilinx Vivado® project and a simulation testbed. Control files such as constraints are also documented and provided in the FrameWork Logic package.

While the development tools and methods are discussed in this manual, it does not intend to substitute for the tools documentation from Xilinx. The discussion is limited to using these tools during the logic development process as it pertains to the Cardsharp product family. Source code examples are also shown for illustration of use, but a knowledge of RTL (VHDL) is presumed.

Hardware information is provided in another manual, the Cardsharp Hardware Guide. Consult this manual for documentation such as pin assignments, connector information and performance data.

Getting Started

This manual is written to assist in the creation, implementation and testing of custom logic for Innovative Integration products. The scope of this manual is limited to discussion of the logic development tools, example logic designs and logic libraries provided in the FrameWork Logic toolset.

Additional documentation on each product is provided for hardware features and software in other manuals. These are used in conjunction with this manual for product development and use.

Thank you for using our products. Your comments and input are appreciated so that we can improve our support and help you to be successful on your project. Email us at techsprt@innovative-dsp.com with your input or give us a call.

Prerequisite Experience and Required Tools

The designer is expected to have experience in VHDL and FPGA design to use the FrameWork Logic tools and code. All components in the FrameWork Logic are VHDL source code whenever provided and supported by VHDL models and test code. As a standard, the code is written in VHDL 1993 version which is widely used and supported.

The design tools used are listed here. We make an effort to keep the logic supported under the newest versions, but in many cases the logic must be reworked and retested to support the newest tool version. For each product, we have listed the required tool set that was used to create the logic.

Here is the toolset we use for supporting the FrameWork Logic use and development.

Function	*Tool Vendor*	*Tool Name*
Synthesis, Implementation, Simulation	Xilinx	Vivado
Bit Image Creation	Xilinx	Vivado
Logic Debug and Testing	Xilinx	Vivado “Set up Debug” wizard
Logic JTAG Cable	Xilinx	USB

Table 1. Supported Logic Development Tools

The documentation for the development tools is provided by the tool vendor. They have on-line documentation and help that can acquaint you with their use. This manual makes no attempt to replace them, but rather supplement them with specifics on using them with FrameWork Logic application development.

While it is not expected that you are expert in these tools, these tools are used for FrameWork Logic development and are discussed in this manual. If you are using other tools, they should have similar capabilities.

Installing the FrameWork Logic

The Framework Logic is delivered as an compressed archive ZIP file (*product_name*.zip). A TCL script (make_vivado.tcl) is provided to build a Vivado IDE project and set the project options. To get started, follow these steps.

1. Unzip the logic archive into a folder. You will find the archive in the hardware directory of the product installation, by default C:\Innovative\<product name>\Hardware\Framework Logic\. For example, unzip to this folder:

   ```
   /Innovative/<product name>
   ```

2. If a previous version of the project file exists, it must be deleted. Delete or rename old copies of the <product name>.xpr project file. If you had an old project file from a previous installation, it would be in

   ```
   /Innovative/<product name>/logic/rev_*/vivado
   ```

3. Open Vivado IDE.

4. Close the project if one is open.

5. The tcl console is at the bottom of the welcome screen.

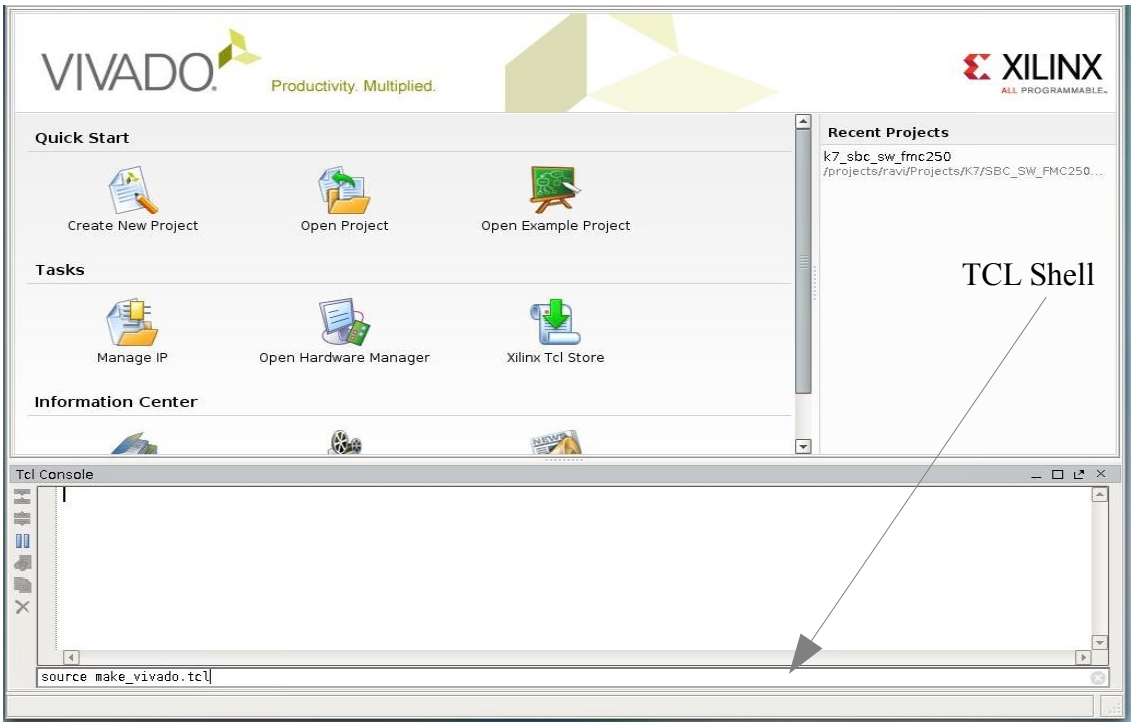


Figure 1. TCL Shell

6. In the text box on the TCL console, type

```
cd <tcl_script_location>  ie. /Innovative/<product name>/logic/rev_*/vivado
```

7. At the % prompt on the TCL screen, type

```
source make_vivado.tcl
```

This generates a directory with the Vivado project ...vivado/<device>/<product name>.xpr

Wait about a minute while the Vivado project is created.

Tool	File Name	Directory
Vivado IDE	<*product_name*>.xpr	$project\vivado\<device>\

Table 2. Project Files

Logic Directories and Files Organization

The logic files for all Cardsharp products are organized by product with a library of components used by many designs. The logic for each design provides support for simulating, creating and debugging. Design-specific source files in the source directory include the top-level and other components. Constraints for the design are located in the xdc directory for the main design, while individual components may have additional xdc files in their respective directories. The tools will combine all xdc files to create an overall xdc that includes all the physical and timing constraints required for the product. Results of the compilation are included in the vivado/<device> and syn directories showing the compilation and fitting results.

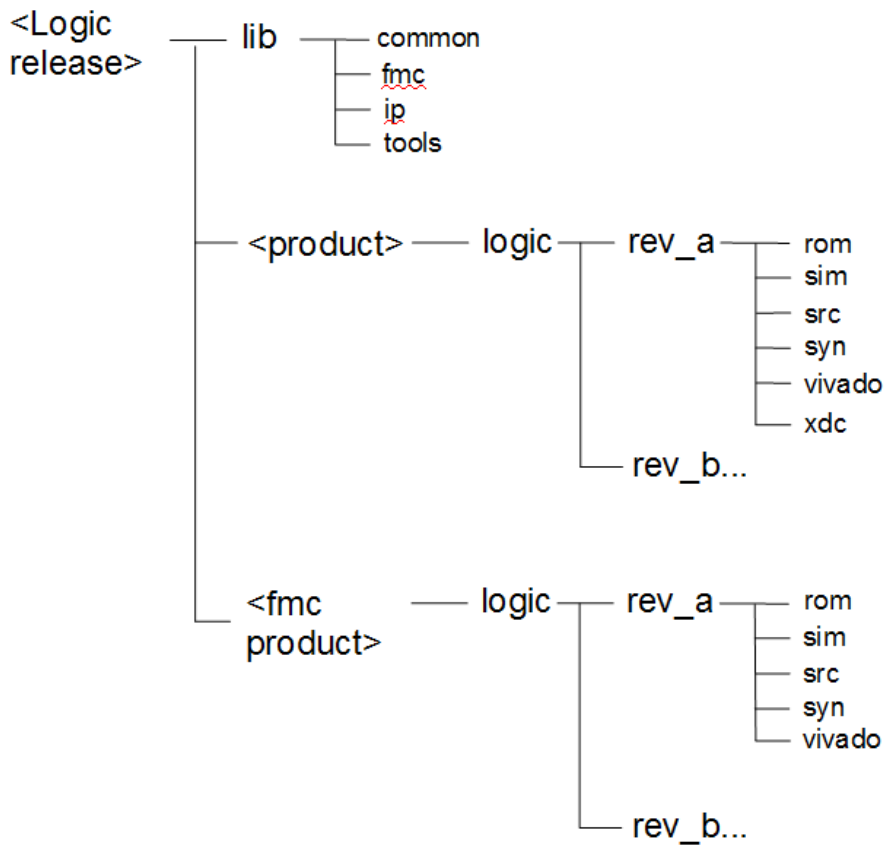


Figure 2. Cardsharp FrameWork Logic Directory Structure

Directory	Files
./product/logic/rev_*/vivado	vivado project and build directory
./product/logic/rev_*/xdc	Vivado constraint files
./fmc_product/logic/rev_*/xdc	
./product/logic/rev_*/rom	The released logic image in BIT format
./fmc_product/logic/rev_*/rom	
./product/logic/rev_*/sim	Logic simulation files. Including simulation models.
./fmc_product/logic/rev_*/sim	
./product/logic/rev_*/src	Logic source files and constraints (UCF)
./fmc_product/logic/rev_*/src	
./lib/common	Source files for Cardsharp library components common to all products
./lib/ip	Logic components in netlist files for Cardsharp library components common to all products
./lib/fmc	Source, netlist and constraint files for fmc modules
./lib/tools	Logic tools provided with this design. (This may be empty.)

Table 3. FrameWork Logic Directories Structure

Logic Component Naming Conventions

For all components provided by Innovative Integration, the standard naming convention is

```
ii_<function_name>
```

where <function_name> is a descriptor of the component.

For example,

```
ii_pcie_intf
```

is the PCI Express interface component.

Organization of this Manual

This manual covers the main topics in using the FrameWork Logic for Innovative Integration products for HDL development methods. The first few sections describe the HDL tools and development methods including synthesis, placement and routing, and simulation. Finally, the generating the logic image and debugging are discussed.

Each product supported by the logic is discussed, showing the details of the example logic are shown. The hardware interface components and application logic internals are shown.

Finally, the FrameWork Logic library components are shown with details about functionality, ports and use.

Where to Get Help

In addition to this manual, the example design for each product is provided with an HTML document that allows designers to quickly navigate the design to understand the hierarchy, entities used, ports and source code.

For help on Innovative Integration hardware or software, there are separate help manuals and an on-line help system for the software tools. These manuals are provided on the CDs delivered with the product or on the web at http://www.innovative-dsp.com/support/productdocs.htm . At this site, you can download the product information, software and logic updates.

Help for other tools such as Xilinx is provided on-line with the tool. Xilinx also has an excellent Answers Database on the web (http://www.xilinx.com/support/mysupport.htm) and many examples of techniques used in FPGA design. This is the primary site for support on Xilinx- specific problems that can include tools problems and workarounds.

Technical support from Innovative Integration is available at

Web Site www.innovative-dsp.com (product manuals, software updates, firmware and discussion forums)

Email us at techsprt@innovative-dsp.com

Phone : ++1 805-578-4260 Monday through Friday, 8 AM to 5 PM Pacific Standard Time

Logic Development Process

The FrameWork Logic system supports two logic development methods: VHDL, MATLAB Simulink, or a combination. Each system offers benefits and have strengths that in some cases complement each other.

VHDL development is very flexible, allowing the developer the full freedom of a high level language that is expressive and extensible. The FrameWork Logic system provides VHDL components for hardware interfaces that allow the designer to quickly integrate custom VHDL code into the application logic. Other library components are offered that provide some common functions used in signal processing and control. Libraries from Xilinx and third parties are also used to provide broad support for signal processing, analytical and communications applications.

***Development* Tool**	***Pro***	**Con**
VHDL	Expressive, extensible language. Gives complete flexibility to the designer.	Design and debug of DSP algorithms is more difficult and time consuming.
MATLAB Simulink	Allows design of complex DSP algorithms at a high level. Great visualization and analysis tools for design and debug.	Less capable of handling low-level details. Less visibility and control of logic design process.
VHDL + MATLAB	Best of both tools gives optimum flexibility where needed and high level design for complex DSP algorithms.	Multiple tools must be used resulting in a more complex development process.

Table 4. Logic Environment Pros and Cons

MATLAB Simulink offers a high-level block diagram approach to logic design that allows the designer to work at a higher, more abstract level. Signal processing algorithms can be quickly developed and simulated in MATLAB then directly ported to the logic hardware. Inside of the FrameWork Logic tools, the designer can concentrate on the algorithms because the system has a hardware interface layer that integrates the hardware with MATLAB cleanly and efficiently. Application development is dramatically sped up for complex signal processing algorithms because of the powerful capabilities within MATLAB for algorithm design, visualization and analysis.

Many applications find that a mix of VHDL and MATLAB offer the best of both worlds: high level signal processing development and the full flexibility of a high level language. It is common that unique data handling, triggering and interface functions may be better expressed in VHDL, but nothing beats the power of MATLAB for things like filter design, down-conversion and mathematical analysis of data. The designer can mix VHDL components, or MATLAB-generated components with one another in either environment and reap the benefits of each system.

Developing Using VHDL

Application logic development with the FrameWork Logic in VHDL follows the typical development cycle: code creation, simulation, physical implementation and test. This flow is summarized in the following diagram showing the Vivado tools for design, synthesis, implementation and simulation tool.

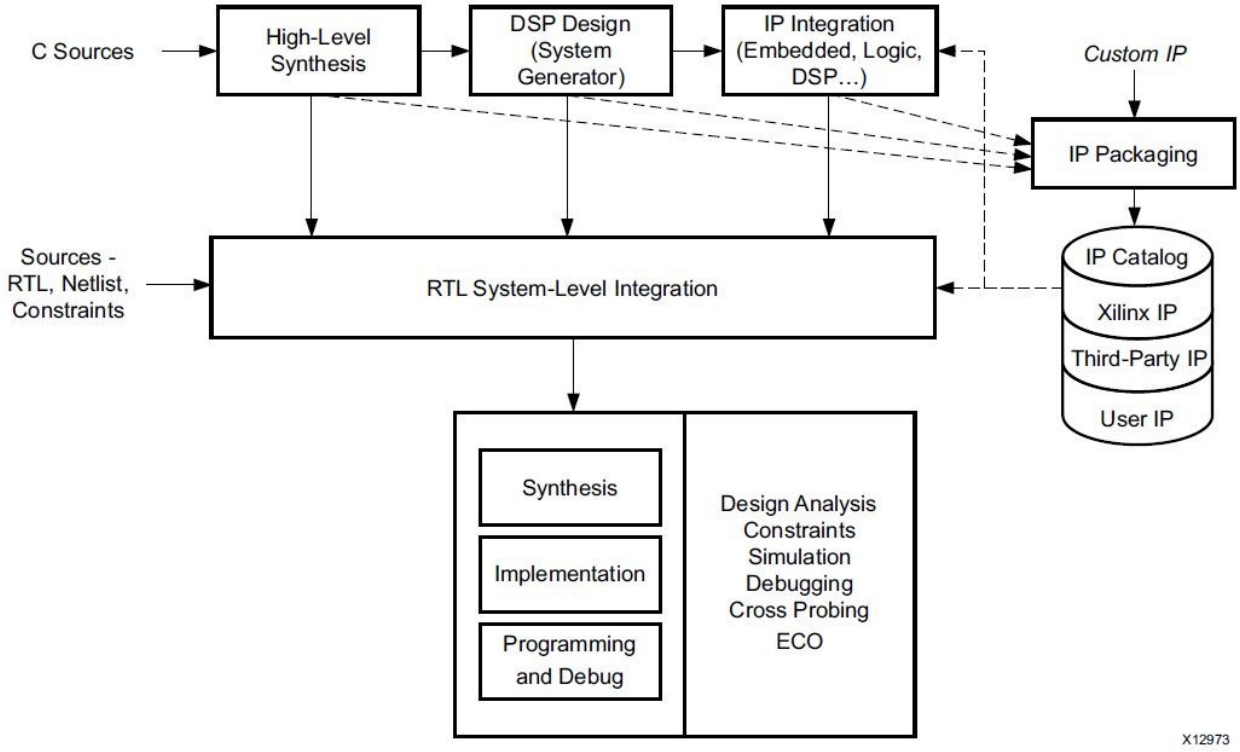


Figure 3. High-level Synthesis Design Flow

The application development begins with the FrameWork Logic code for the product you are using. In many cases, the example application code provides a good starting point for your application logic. In most cases the application logic shows a basic data flow between the IO devices, such as A/D and D/A converters, to the logic and DSP or system. You can then build on top of the example logic by inserting your algorithms into the data flow along with unique triggering and other application-specific logic.

The FrameWork Logic provides a library of components for the hardware interfaces as well as others functions, an example application showing IO interfacing and data flow, design constraints, a simulation testbench, and a Vivado project. This gives you the basic foundation to begin work. After you install the FrameWork Logic on your system, you should be able to recreate the logic and verify its operation. Once that is complete, you are ready to begin development.

At this point, you should have a look at the example logic and determine the best place to insert your logic and how you can best use the example in your development. If you can preserve many of the basic memory mappings, controls and system

interfaces, you will then be able to use the example application software delivered with the product. That saves time for both you and the software developers.

In most cases, you will see that the logic is organized as a hardware interface layer composed of components that directly interface to the hardware and an application layer that is composed of the analysis, data handling and triggering functions. The application layer is on a single clock domain so that it is easy to integrate functions into the design.

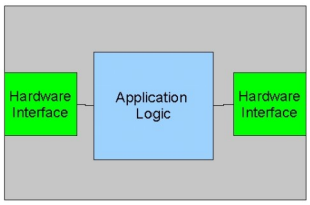


Figure 4. Logic Architecture Showing Hardware and Application Layers

Code for your application layer design can be created in a number of ways: written in VHDL or Verilog, created in MATLAB, or included as a black box netlist from a third party such as Xilinx or others. If you design your logic to run on a single clock it is then easier to integrate into the application layer of the FrameWork Logic. The other clocks in the design, such as the A/D sample clocks, or hardware-specific clocks are handled in the hardware interface layer. The use of a single clock in the application layer allows designers to use timing and physical constraints associated with the hardware interface components.

Using Vivado

Xilinx Vivado IDE is recommended for the logic development. Vivado provides code editing, core generation, synthesizing and fitting tools integrating all of the tools for the project. An example project is shown here.

The existing project should be used as a starting point. This project has all the options set and file structure required to recreate the design. The Vivado project is created by running a tcl script located in the ./logic/rev_*/vivado directory. When you open Vivado, in the tcl command window, use the command *source* to run the make_vivado.tcl file as shown in the image below. Before running the script, make sure that the DEV, SPEED and LANES properties are set correctly. The DEV and SPEED properties should match the FPGA on the baseboard. The LANES property is for the number of lanes of pcie you want integrated in your design. You have a choice of “x4” or “x8” for a 4 lane or 8 lane pcie core respectively. If you need to use the aurora interface, set the ADD_AURORA_FMC, ADD_AURORA_0 or ADD_AURORA_1 to “TRUE”

Note: if the value of a property you're modifying is in quotations or has a '-', do maintain the same format when you change the value to avoid errors when generating the project.

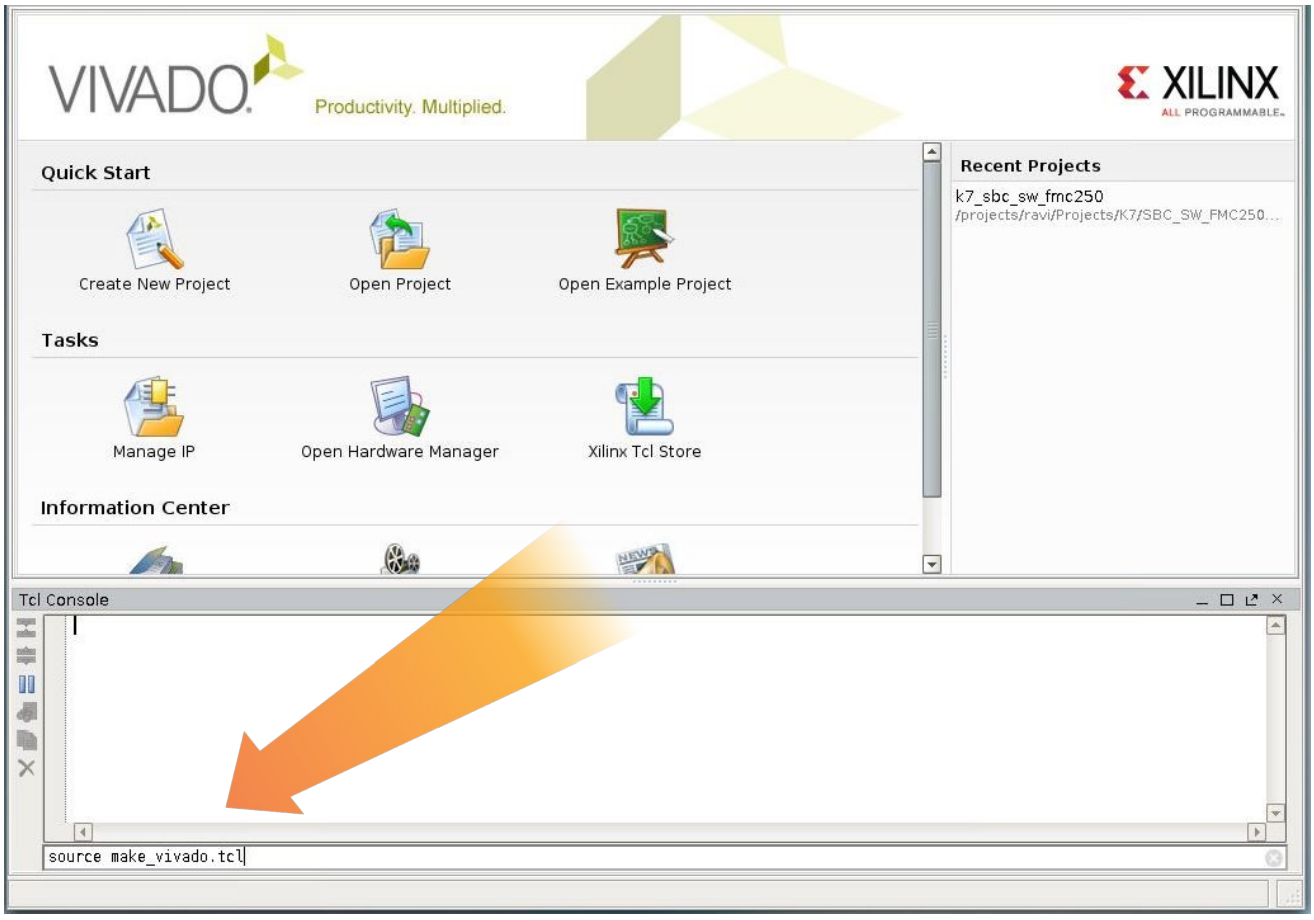


Figure 5. Generating the vivado project

The generated project is saved under a folder that is named after the device and pcie lanes. For example, if you're creating a vivado project for a 045 part of speed grade 2 with 8 lane pcie core, the vivado project is saved under a folder with the name **045-2_x8**. Click on Open Project and navigate to the *.xpr file in the created folder.

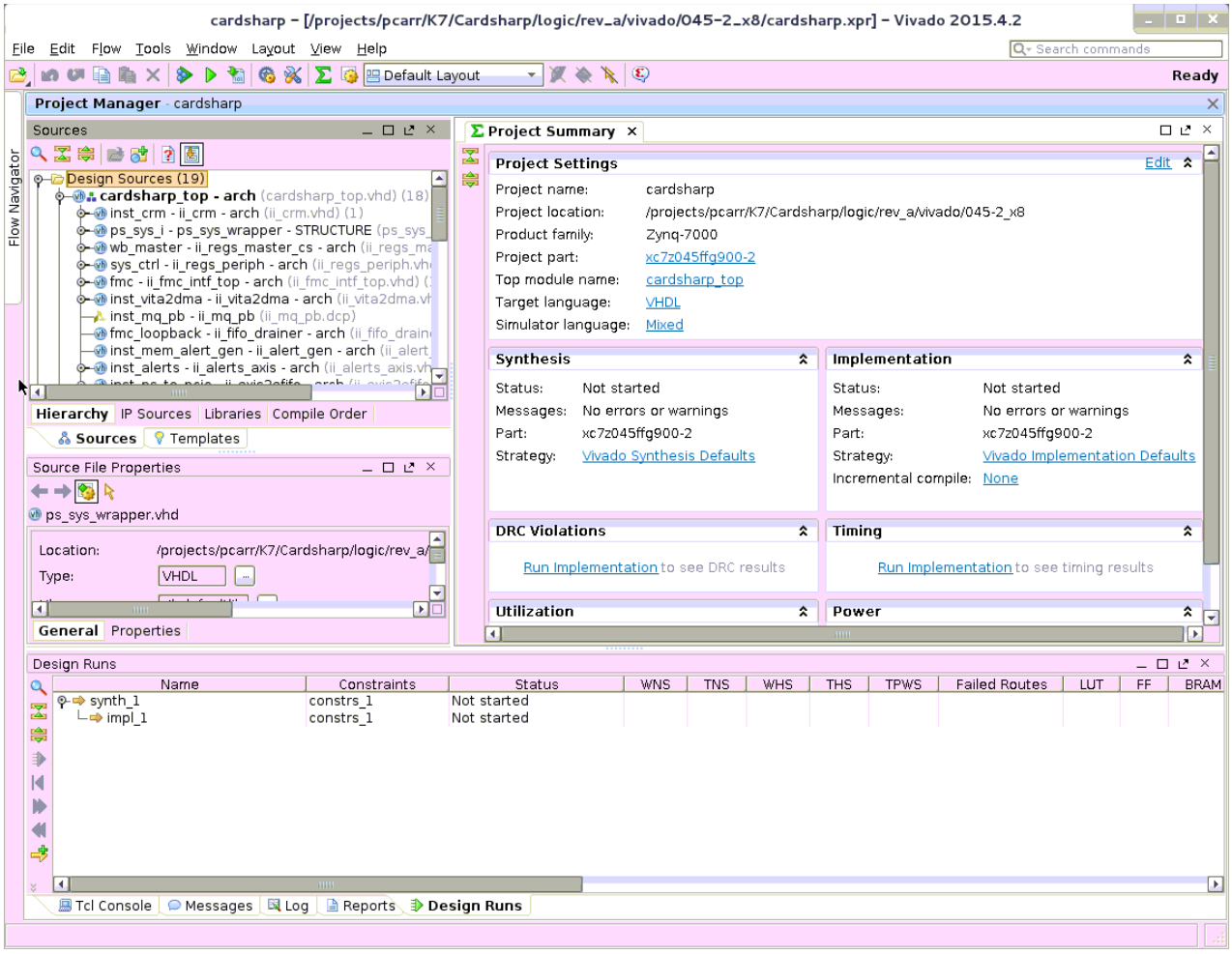


Figure 6. Example Vivado project

Note: The project options have been set to use the directory structure for the FrameWork logic design. It is important to use these options when compiling the project so that the cores and source code can be found. It is also required to preserve hierarchy on the design to use the constraints provided.

Using the FrameWork Library

The components in the FrameWork Logic library are divided into common components used in many designs and more complex IP cores that are usually hardware-specific. All common components provide source code while IP cores are provided in netlist form.

The hardware-specific components are used in the designs for A/Ds, DACs, memories and the like that have unique interface protocols and timings. Constraints in the specific design for IO standards and specific timing requirements are usually required for use. The constraints for the hardware-specific components are found in the application example that includes that component.

All components have unique names such as *ii_sdf_adc*. The naming convention prevents inadvertent naming collisions with your design if you do not use a *ii_* prefix on your components. The hardware name is included in the name showing which design uses this component.

In the installation, you will find that hardware-specific components in the directory for that specific design. The common components are in the lib directory. To use the components, you can point at the library Vivado when they are used.

Also, you may need to include packages supporting the components in your design. For example, *ii_pcie_intf* component requires *k7_pkg* to be included. This is done by including these statements in the component and by compiling the package in your design.

```
library  work;

use  work.ii_k7_pkg.ALL;
```

Directory structure for FMC modules

This section will describe the file structure for fmc modules. The firmware directory of an fmc module has 3 sub-directories. The lib, baseboard and the fmc module. As a user, you will be working in the third sub-directory if you are adding custom logic. The lib and baseboard source files should not be modified. The lib/fmc folder has the source and constraint files of each of the fmc modules in their respective folder. The baseboard source and constraint files are in the Cardsharp folder.

When working in the fmc module directory, the rev of the baseboard and the fmc hardware will decide which rev folder you will be working in. rev_a directory is to be used when both the baseboard and fmc are revision 'a', both hardwares being revision 'b' means you will be working in the rev_b directory. In case of the rev_ab and rev_ba, the first letter of the rev is for the baseboard and the second is for the fmc. The vivado directory inside the appropriate rev folder can be used to generate the bit file.

Simulation

Simulation is an important part of the logic development process. Unless the design change is very simple, you should simulate the design. In the end, you will save time and frustration especially when larger devices are used.

The FrameWork Logic includes a test bench and models required for most simulations. These models are functionally equivalent to the component. In many cases the models are simple representations of the device that give a data pattern that is easy to follow through the simulations. More complex waveforms can always be substituted later for proving out the signal processing or data analysis portions of the design. In each design, the list of files shows the applicable test bench name and available models.

The testbench contains a set of simulation steps that exercise various functions on the FrameWork logic for basic interface testing. Behavioral procedures have been written to simulate the host timing for the Processing System (PS) and PCI Express that are useful in simulating data movement. Other features such as DDR interface, alert log and triggering are demonstrated in the testbench.

As delivered, the FrameWork Logic example provides a basic example in the use of the hardware interface components, data flow through the design, and some simple triggering control. It is anticipated that you can use this example test bench as a starting point for your application logic simulation. Your logic can be added to the simulation in many cases without modifying the test bench since the application logic does not change the external pins on the design.

Simulation can be started from within the Vivado environment, from the menu option Flow → Run Simulation → Run Behavioral Simulation.

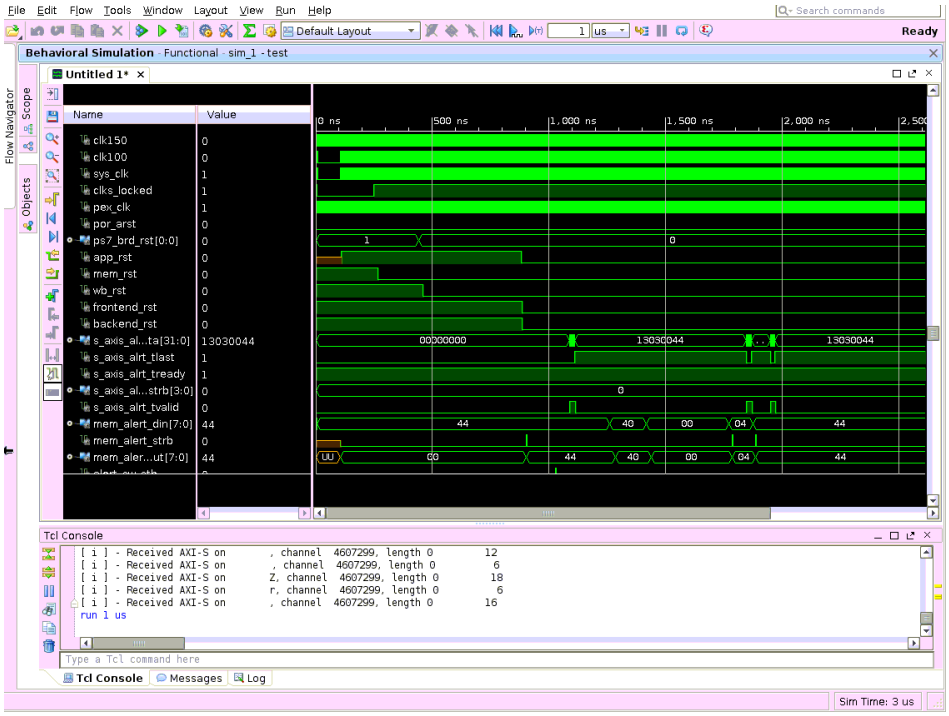


Figure 7. Behavioral Simulation Window Example

Once you are inside the Simulation environment, you should be able to use the tools to run simulations of the design. The wave window is many times the main focus since it gives a logic analyzer view of the design.

You can quickly view the design because you can probe the logic down to the lowest level. This visibility is often lost after synthesis and fitting because logic is minimized by the tools and may be trimmed out if unused, even if by accident. When you select an design unit within Simulation Scopes & Objects windows, the processes, signals and variable for that design unit are shown. You can add them to the window by selecting them and right-clicking to add to the wave window.

Logic Development using MATLAB Simulink

These tools are described in the *Cardsharp MATLAB BSP* manual for each product. Refer to that manual for details on logic development using MATLAB Simulink and Xilinx System Generator. The following description is just to orient you to what that tools are and how they may be useful in developing applications.

MATLAB Simulink provides a powerful method of developing logic using a high level design tool that integrates hardware into the MATLAB Simulink environment. Complex signal processing designs can be developed rapidly using the Simulink block diagrams interacting with the actual hardware in real time. Gateways between MATLAB Simulink and the hardware allow data to flow between the actual hardware and MATLAB, bringing the power of MATLAB to the logic development process.

Simulink blocks diagrams are directly translated into logic using the Xilinx System Generator tool. For each supported product, a hardware interface layer of Simulink components is provided that allows the hardware to be used in the the Simulink design. Simulink components from the various libraries provided by Mathworks, Xilinx and Innovative interface with this hardware interface layer for building the application logic on the product. The Xilinx place and route tools are used for the logic build as in any HDL project.

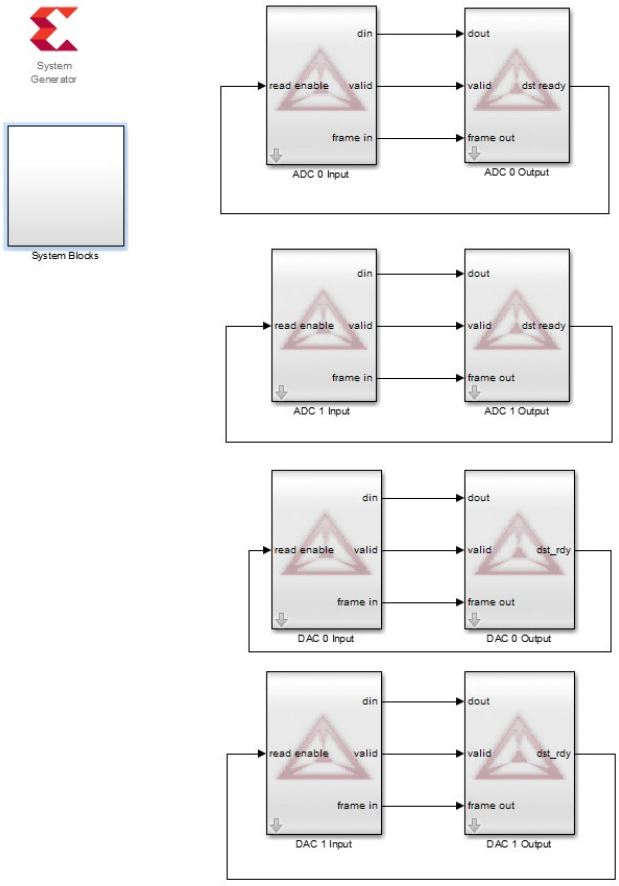


Figure 8. MATLAB Simulink Development

Here is a typical Simulink block diagram design. Notice the Xilinx icon in the upper left; this is the Xilinx System Generator control block. This block provides the link to the Xilinx synthesis and implementation tools. The other blocks are mixture of data interface components to the A/D and D/A converters, SRAM and wishbone interface if desired.

Making the Logic

The Xilinx tools are used for the physical logic creation. For HDL designs, these tools are accessed through the Vivado environment in the processes window as shown here. The main steps are synthesis, implementation and bitstream generation.

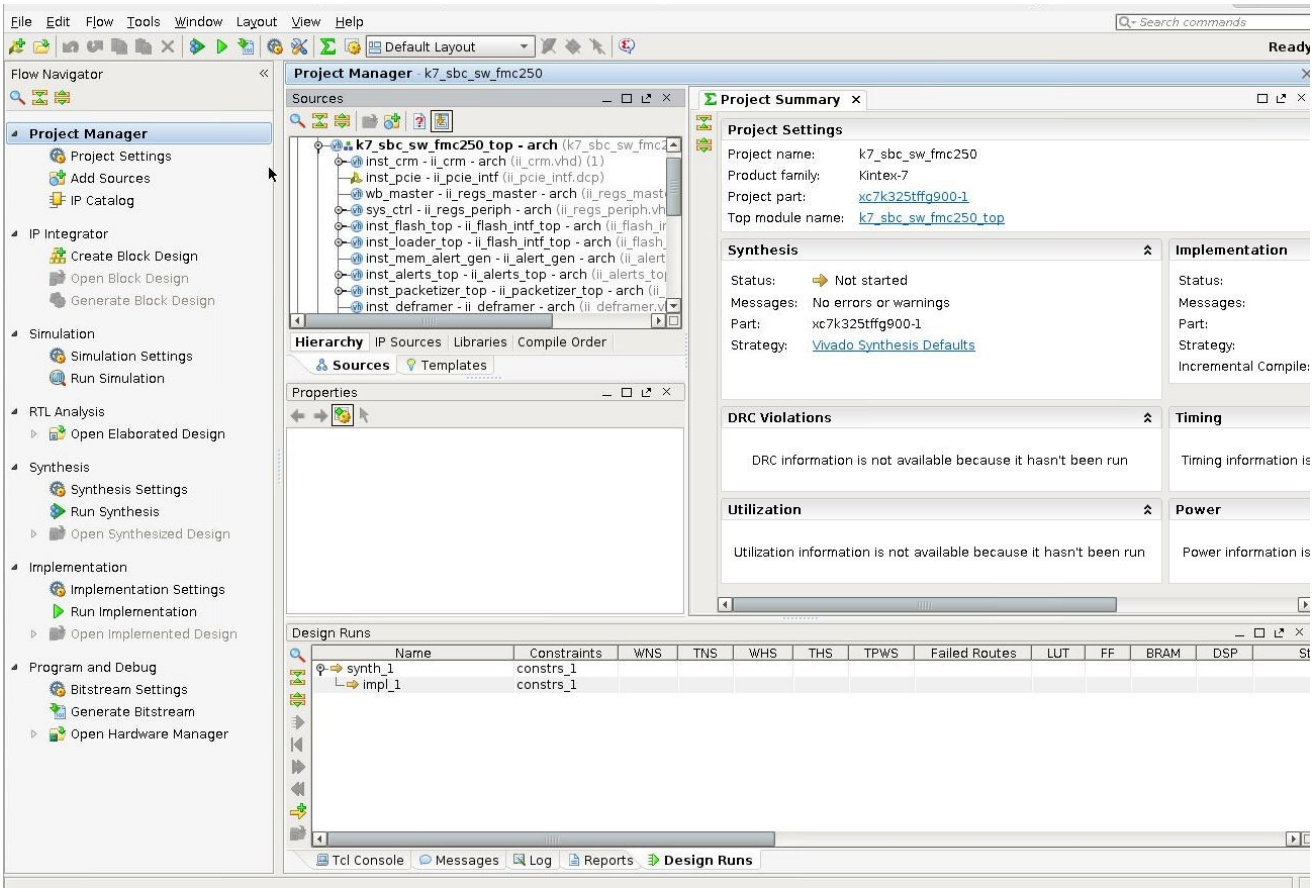


Figure 9. Vivado Design Environment

There are many options for the implementation step which are set in the individual project files for each FrameWork Logic example.

Since most of the parts on the products are very large, we have chosen to preserve the hierarchy of the design during the implementation so that area constraints and incremental design approach may be used. Area constraints allow the designer to control the placement of logic on the FPGA part for best timing control. With area constraints, the logic will be constrained to where you put it and in many cases helps the tool do a better job overall.

Synthesis and Implementation Reports

The reports generated during the synthesis and implementation steps are saved in synth_1 and impl_1 directories respectively in the <project_name>.runs folder. In the synth_1 folder, the *.vds file is the synthesis report and the <project_name>_utilization_synth.rpt details the resource utilization in the fpga.

File extension	*Contents*	*What to Look For*
.VDS	Vivado synthesis report generated after the synthesis step is completed.	There should be no errors. This program issues numerous warnings but there should be no errors. Synthesis errors point to issues in the RTL code
.VDI	Vivado implementation report generated after the implementation step	There should be no errors, but warnings are usually OK. Common problems range from incompatible logic mappings, impossible area constraints, and clock connections.
<project_name>_top_timing_summary_routed.rpt	The output of the Place and Route implementation process. Shows timing results and fit results.	Timing constraints should be met. Review the summary at the end of this report to see if timing is OK since it will complete no matter how bad it is. Also look for any incomplete routing.

Table 5. Vivado Report Files Generated During Synthesis and Implementation

In many designs, you will have to resolve timing problems that are shown in the place and route process. Xilinx has several tools to help find the problems; Timing Analyzer is usually the place to start. This tool helps you to understand the reason the logic did not meet timing – too many connections, bad routing, etc. The tool suggests how to fix it. This is usually helpful but may mean you are back to functional simulation again to add pipelining or change the logic and must re-implement the design.

Once you achieve one good result, you may want to switch to incremental mode in the tool. This allows you to use the last good result for most of the design that is unchanged when minor fixes are made. For big changes however, you will need to reroute the whole chip.

Expect that the implementation process will be in the range of ½ to 2 hours depending on your computer, how easy it is to meet constraints and how big the part is. A tightly packed, fast big part will take a while.

The final output of the implementation process is a .BIT file that represents the logic image. This file is loaded into the FLASH using the EEPROM applet or the JTAG cable. When the JTAG cable is used, the Xilinx IMPACT program uses .bit files.

Loading Logic

The Cardsharp module logic can be loaded from either FLASH ROM or through the FPGA JTAG port. The Logic Loader applet is used to burn logic images into the FLASH ROM. JTAG is usually used during the development process because it is quick, easy to to use and loading can be done from the ChipScope debug environment.

Cardsharp FLASH Images

Logic is loaded each power-up or from the FLASH ROM. As delivered, the Cardsharp module has two logic images in FLASH: the application logic and the "golden image". The application logic is where your logic is deployed. The golden image is the backup that is used in case a bad image is burnt into the FLASH. If a bad image is put into FLASH by mistake that makes the card malfunction, you can set a jumper on the card to force the logic to boot using the golden image (see hardware manual). This allows you to boot the card and reburn the FLASH with a good image. The Logic Loader also allows golden image to be rewritten.

Loading Logic Through JTAG

The FPGA can be loaded through its JTAG port using a Xilinx JTAG cable. This provides a convenient method of quickly loading the logic during the development process but is not usually used in deployed applications.

Caveats

The logic loaded over the JTAG interface is volatile. If the card is powered down it must be reloaded.

The system must be restarted (not power-cycled) for Cardsharp logic to work after a JTAG download. Since the logic has the PCI Express interface, it must go through the system enumeration process for the bus to work.

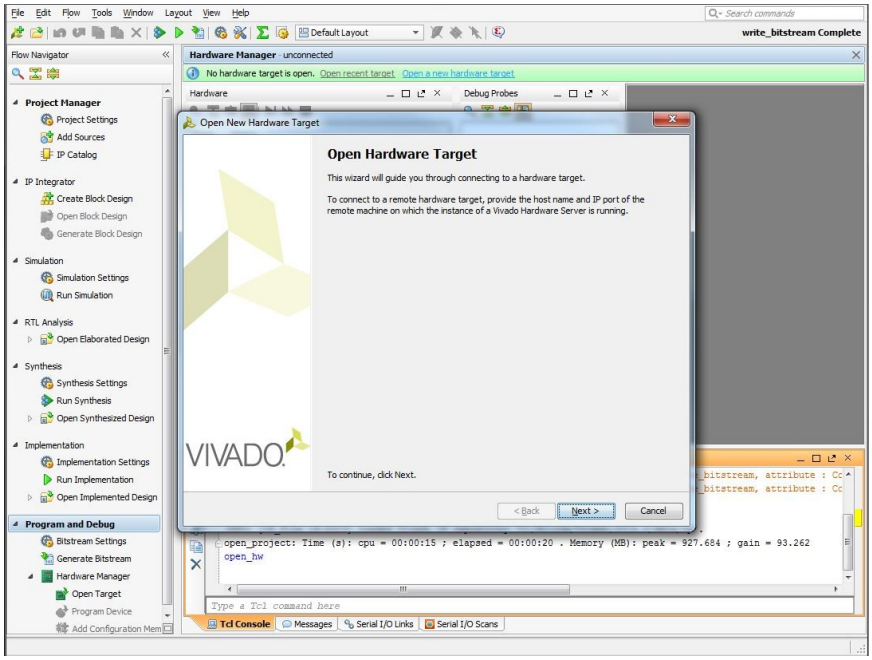


Figure 10. Getting Started with Hardware Manager

The Hardware Manager is used to load the logic into the application FPGA. The tool may be invoked from within the Vivado project or from the welcome screen when you launch vivado as shown below.

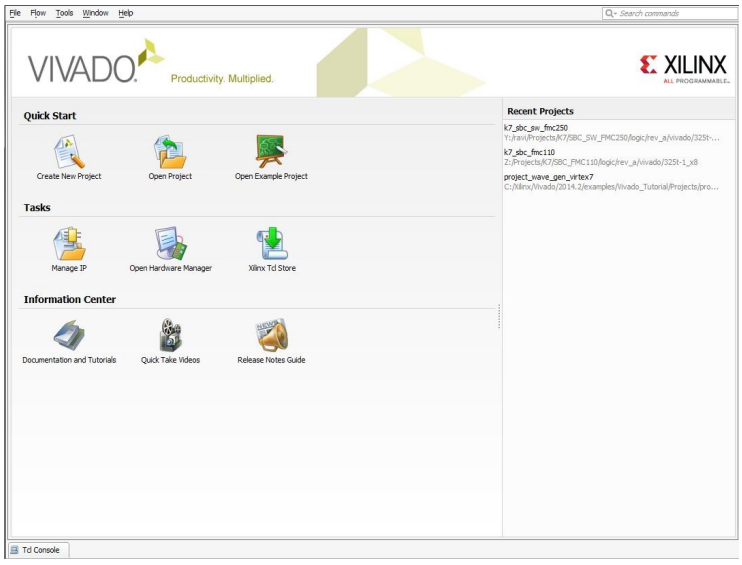


Figure 11. Hardware Manager on the welcome screen

When you enter the tool, if no target is open, you will be given an option to either open a recent target or open a new one. If you open a new target, the open new hardware target window appears as shown in earlier figure. Click next and select local server if the target is connected to the machine you are working on. On how to use remote server option, please refer to xilinx documentation.

When you click next, you are shown the devices on the chain as shown in the figure below.

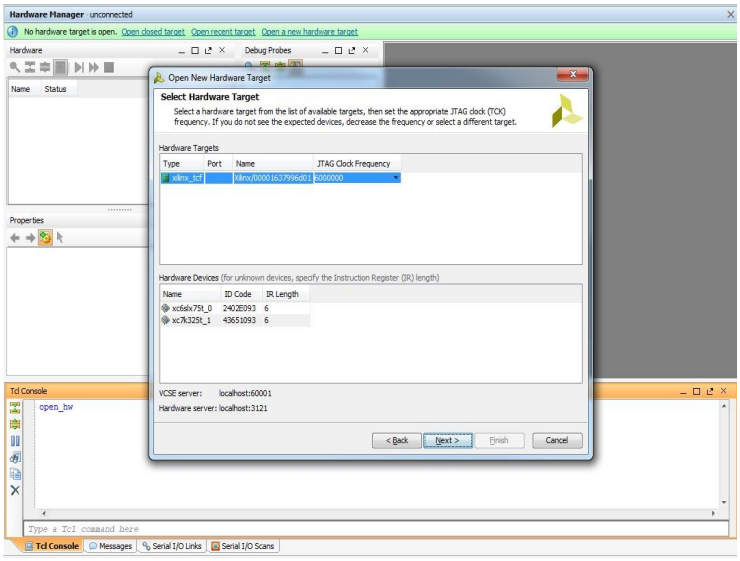


Figure 12. Hardware device chain

If it does not work, check the cable connection to the board. The cable should be detected by Vivado; if not, check that the port it is connected to on the PC is working and in the proper mode. If the chip is not detected, be sure you have the right scan path, that the board is powered up normally, and that Vivado was able to connect to the scan path. Power everything off and try again if it fails and you don't see any obvious problems. You can also check your setup and software on a good card if you have one.

The next step is to assign a logic file to each device to be programmed. The FPGA image is a .BIT file, the CPLD is a .JED file. Right click the device you need to progrm and use the option “Program Device” to assign the BIT file for programming the FPGAdevice. **Double-check that you are using the correct logic file – you could damage the chip if it gets the wrong logic or render the card inoperable.**

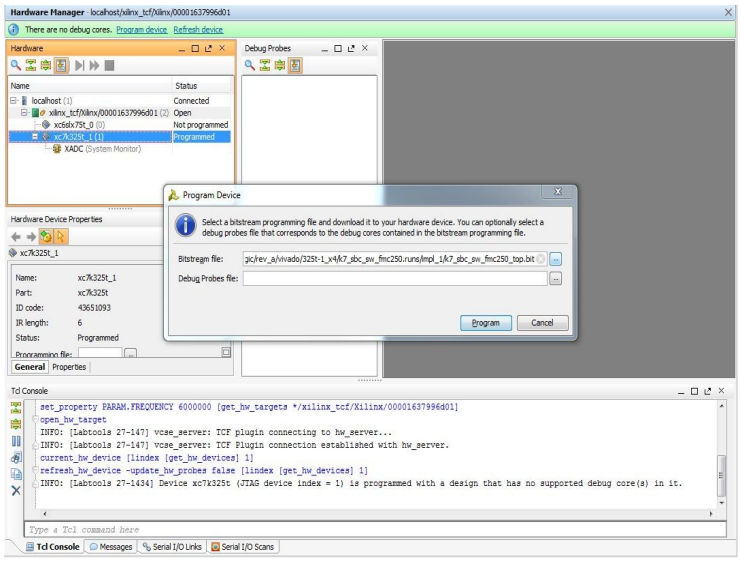


Figure 13. Selecting the Configuration Image

Click on Program to begin the logic load process. This will only program the device you are selecting at that time. If it succeeds, the status of the device in the hardware window will say “Programmed”.

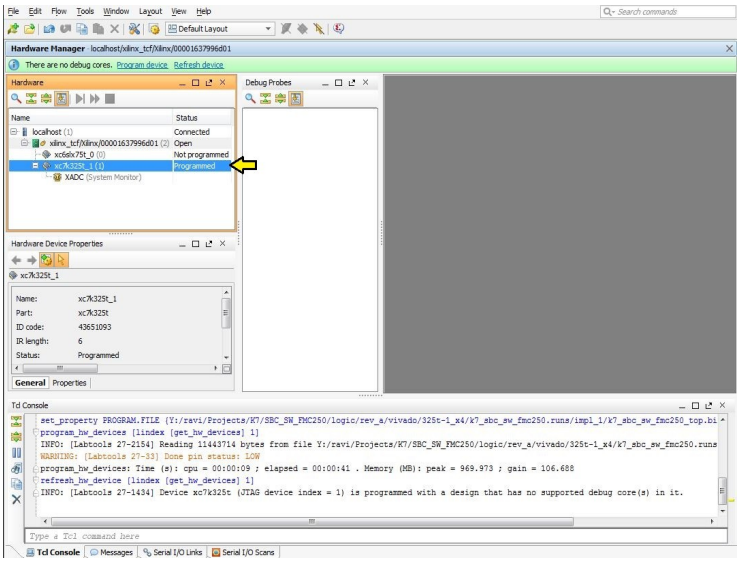


Figure 14. Programming Devices using JTAG

Note: For FPGAs that are on PCI or PCI Express interfaces, the system must be restarted to force the new logic to load. The system enumeration process must assign address mappings and resources before the logic will operate. Whenever the PCI

information such as revision number changes, it is common for the system to recognize the new device and will want the device driver reloaded. This is normal.

Logic Update Utility (Logic Loader.exe)

The Logic Loader applet is used for field-upgrades of the logic firmware on the Cardsharp module. The applet updates the FLASH ROM, which stores the "personality" of the board. The Logic Loader uses BIT format files that are created by the logic tools (BITGEN).

This applet should only be used after firmware development and debugging has been completed. During the development cycle, it is much more efficient to download and debug firmware using the Xilinx JTAG cable.

To use the applet, select the instance of the Cardsharp module to be updated. This will be target zero in single-card installations. For multi-card systems, the target number can be found by using the FINDER program that will flash the LED on a selected target number. It is important to get this target ID correct since burning the wrong image into a card could cause it to stop working.

Then browse to select the logic bit file image containing the updated firmware image. Typically, this is located in the `Innovative\<product name>\Hardware\Images` folder on your default drive.

Finally, click the Write button to program the firmware into the on-board FLASH ROM. Programming typically takes about five minutes. After power-cycling the PC, the new firmware will take effect.

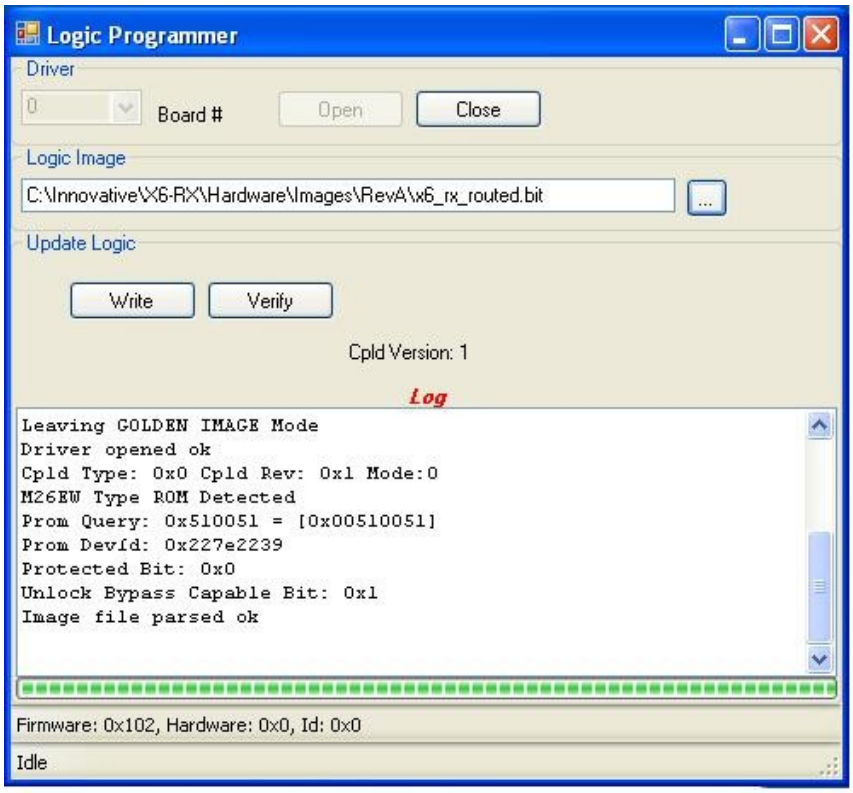


Figure 15. Logic Loader Download Applet example

The Logic Loader applet also shows the logic revisions, hardware revisions and other information. The logic revision is reported from the logic itself, while the hardware revision and hardware variant are set by the hardware PCB and PCIe interface FPGA.

What To Do If FLASH Programming Fails

The Event Log window reports any failures during the programming. If anything goes wrong, DO NOT TURN OFF THE COMPUTER. Parsing failures usually mean that the file was not EXO format. A load failure is more serious in that the hardware failed somehow. Try to rerun the Logic Loader applet. If this doesn't work, then turn off the computer and set the jumper to use the Golden Image. Restart the system and attempt to burn the logic again. If this fails, then the card is faulty and must be repaired.

Debugging

It is inevitable that the logic will require some debugging and it is best to have a strategy for debug before you actually use the hardware. Debugging on actual hardware is difficult because you have poor visibility into the FPGA internals.

For HDL designs, the best and easiest debug method is simulation for functional and timing problems. This gives the best visibility and interactivity to debug problems before the real hardware is tested. A good set of test cases that stress the design should be run prior to working on the real hardware. You will save time in the overall design process by doing a thorough job in simulation.

There are several techniques that have worked for us on projects: Xilinx ChipScope, built-in test modes, and judicious use of test points. Between these techniques and the capabilities of each method, it is usually possible to find and fix bugs that are either functional design errors or timing problems.

MATLAB Simulink developers can use the "hardware in the loop" features of system to debug the design at a high level. Simulink can be used to generate test data or for viewing and analyzing real hardware data. This is invaluable in debugging complex signal processing designs.

Here we will discuss a few of these techniques.

Built-in Test Modes

Another good way to debug your design is to have built-in test modes in the logic. If you plan ahead for test, then you can more quickly validate your design later and spot problems. When you finish the design, if the test generators and checkers can be left in the design, they are there later as production debug or test.

In many designs, test pattern or data generators are invaluable since they provide known data into the FPGA so that the output is known. If the data source is analog in the real design, substituting perfect data is nice because it helps spot problems that may be hidden in the noise. The test pattern may be an easily recognized stream, like incrementing numbers, that are easy to check in the logic or on the test equipment. Also, its easier to test the extreme cases of the design that may be difficult to reproduce with real signals.

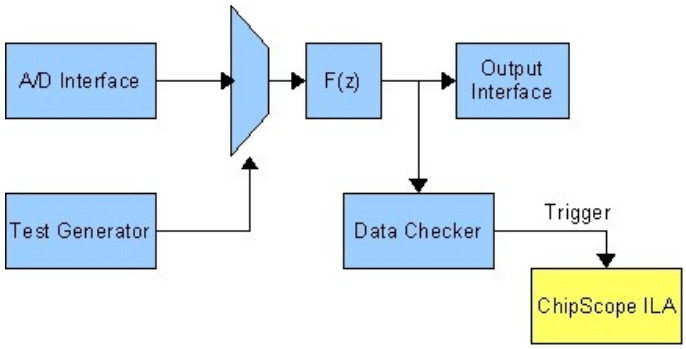


Figure 16. Typical Debug Block Diagram

Another built-in test method is to use data checkers in the logic sprinkled through the data chain help to spot the source of problems. If you have a missing timing constraint or a clock domain issue, these can be hard to catch since they may be rare. A data checker gives you a way to look for bad data and then trigger ChipScope or the logic analyzer. In many cases, rare errors are impossible to catch without this sort of data checker. This technique has saved time because the trigger at the bad data point allows you to inspect all the suspect signals and find the culprit.

ILA debug cores

Xilinx allows for logic debugging by inserting ILA debug cores generated using the “set up debug” wizard. This tool works over the FPGA JTAG port using any of the standard Xilinx JTAG cables. Vivado connects to ILA core that you embed in your logic. Refer to Xilinx documentation on how to insert the ILA debug cores.

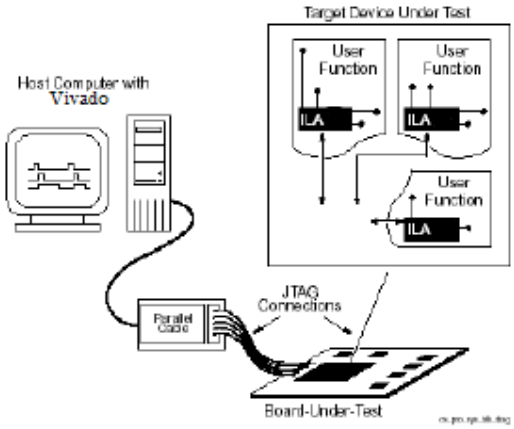


Figure 17. Debugging with Vivado

The ILA core alows you to monitor internal FPGA signals using triggers and a sample clock. It is as though you can embed a logic analyzer in the logic itself, hence the ILA core name (integrated logic analyzer). Other core supported is theVirtual IO (VIO) core, which allows you to monitor and control some internal signals, and cores for working with the PowerPC cores in some logic devices.

The ILA core is very configurable and it allows you to set the number of signals you can monitor, the trigger methods and the signals used for triggers is set up when you generate the core. The core size is determined by the number of signals monitored and the number of samples stored. If the core gets too big, it will affect your design and tends to muddle the debug process. Sometimes it is better to have a small core that has a small footprint and does not interfere with the other logic for this reason.

The clock is used as the sample clock for the logic so it should be synchronous to the inputs signals or sufficiently fast to sample them accurately. If you sample signals on other clock domains, be aware that the clock used by the ILA core is used for the sampling of the signals so the signals will not precisely represent the real signal running on another clock domain.

You will interact with the Vivado software over a JTAG cable to the target device. This link is limited to about 1-20 Mb/s depending on the target device JTAG chain, so it is not really real-time, but rather just a means to get the data from the FPGA to Vivado. The signals are captured in the FPGA block RAMs so the record length is somewhat short being limited in most cases to 256 to 1K points. In some experiments though we have made larger captures of up to 16K points in large devices, useful for capturing a signal.

Because of these limitations in JTAG speed and capture size, it is important to devise triggering methods that allow you to catch the error condition. It is common to devise a piece of error detection logic that serves as a trigger to ILA to best use the capture RAM. It is possible to pre-trigger or post-trigger in the software which makes trigger design easier. You can also selectively store data so that the memory is preserved just for useful data by using an option on the trigger panel in Vivado.

Here is one of the common cables used for debug, just for reference.

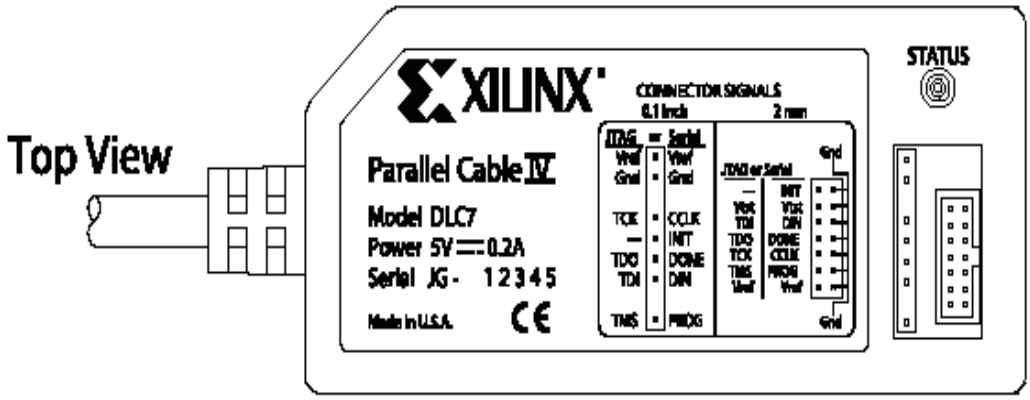


Figure 18. Xilinx Parallel IV Cable for Debug and Development

Figure 19. Xilinx Target Debug Cable

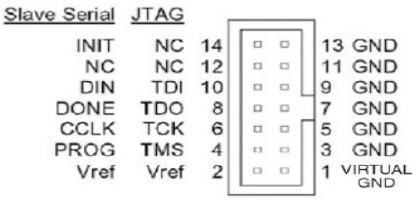


Figure 20. Xilinx Parallel Cable IV Pinout on IDC 5x2 2MM Header

CAUTION:

The user MUST make sure that Xilinx JTAG cable connector is plugged in the proper polarity to the Innovative target connector. If by mistake, the user connects the Xilinx cable incorrectly, this may damage the target card and Xilinx POD. See the hardware manual for each product to locate the connector and its pinout.

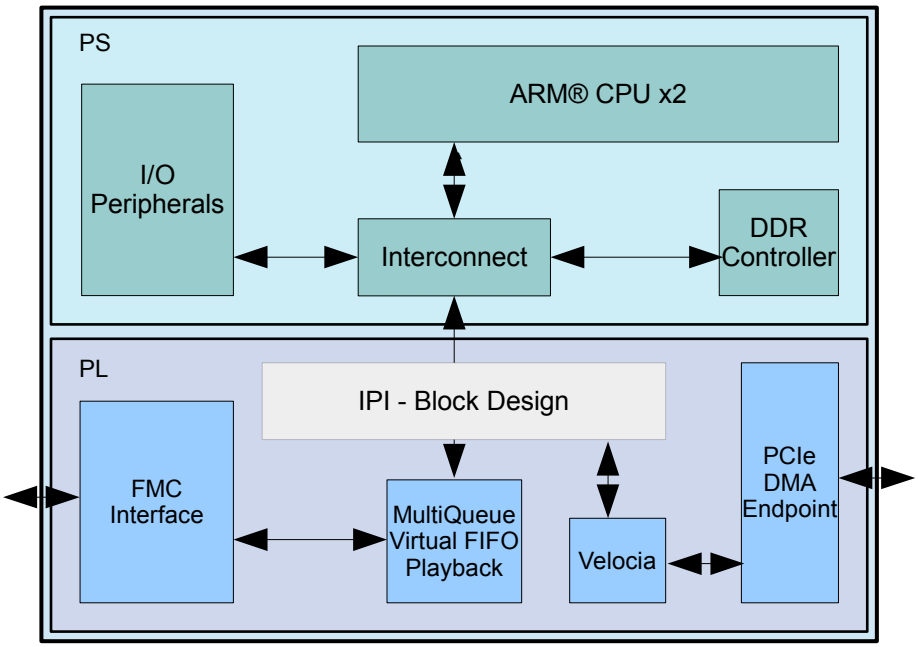
Cardsharp Top Level

Overview

Cardsharp is based on Xilinx Zynq-7000 System-on-a-Chip (SoC) architecture. It integrates a feature-rich dual-core ARM® Cortex™ -A9 based processing system (PS) and 28 nm Xilinx programmable logic (PL) in a single device. This chapter will detail the PL section structure and functionality.

Block Diagram

The following block diagram shows the two main sections on Cardsharp within the Zynq part. It includes the Processing System (PS) and the Programmable Logic (PL), using the Block Design to interface them. The PS runs the embedded software stack including I/O peripherals under software control, including Ethernet, USB, eMMC (flash storage), PS DDR3, UART, etc. The PL holds the interfaces to FMC, PCI Express, PL DDR3 and PPS timing controls.

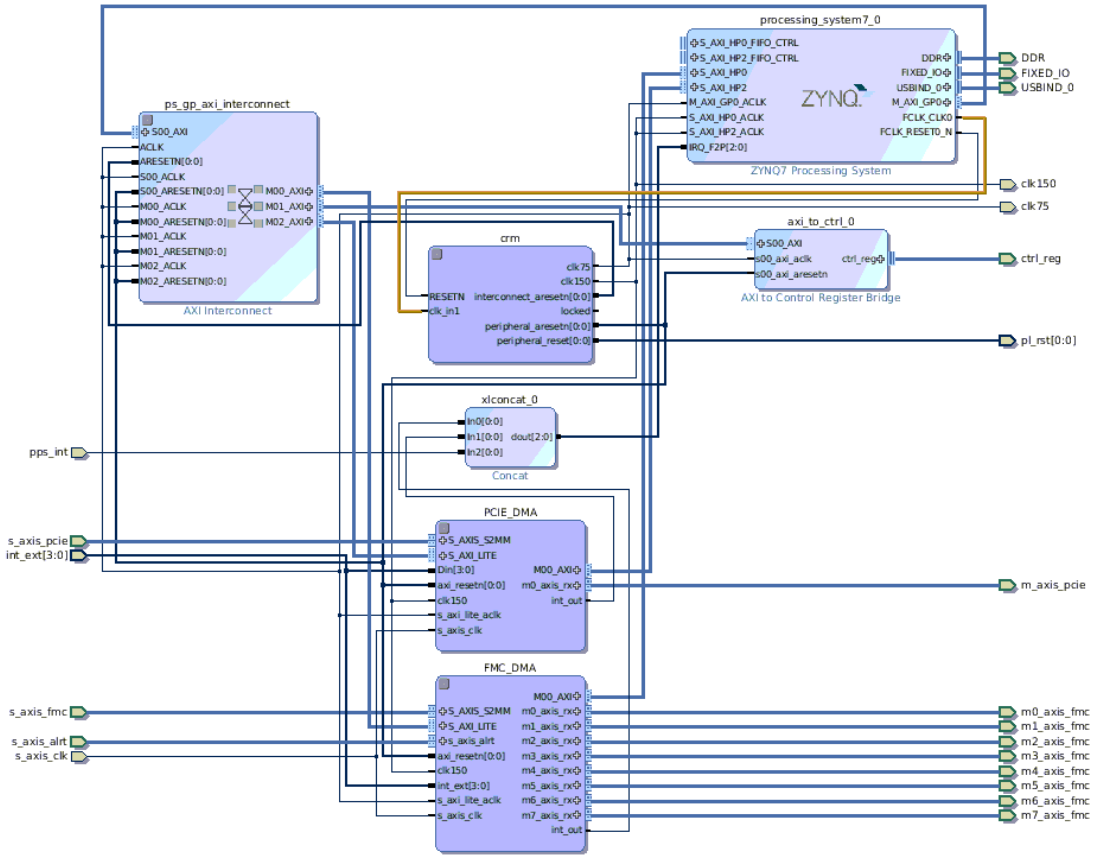


Logic Hierarchy

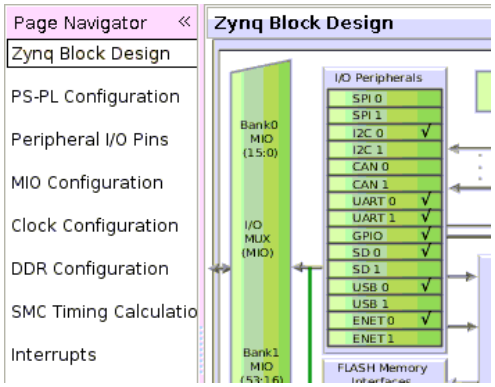
The PL logic is organized hierarchically as shown in the following diagram. The top level instantiates the main functional blocks, namely the FMC interface, Multiqueue Virtual FIFO, PCIe DMA, Block Design with Zynq subsystem, Register Files, Alerts module, clock generators, Velocia Packetizer/Deframer modules, and miscellaneous interfacing modules. Also shown is the main datapath structure moving data from/to the FMC interface to the Zynq for processing.

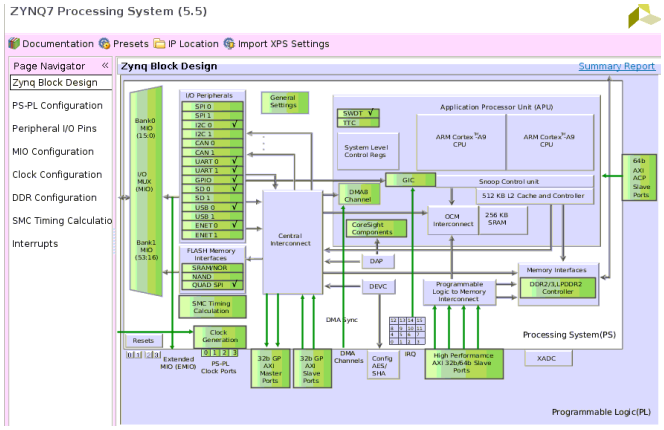
Block Design

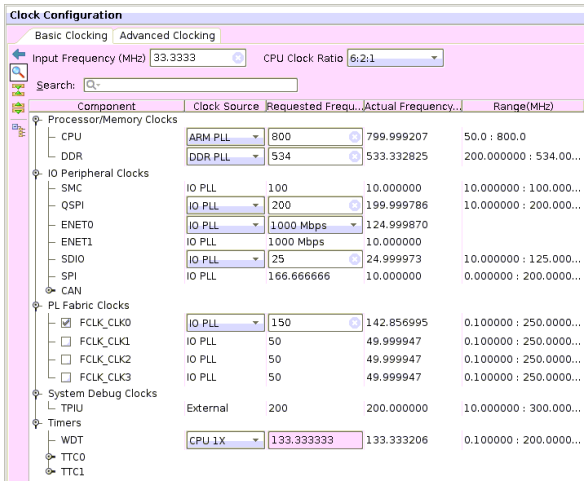
The Block Design is a Xilinx IP Integrator instance and contains the Zynq PS sybsystem, two custom DMA engines, the AXI to Wishbone bridge for register access and the necessary AXI Interconnects. Its main function is to adapt and interface the streams from/to the FMC and PCI Express interfaces into a format appropriate for the PS to process. The DMA engines help move streaming data from/to the Zynq PS memory.

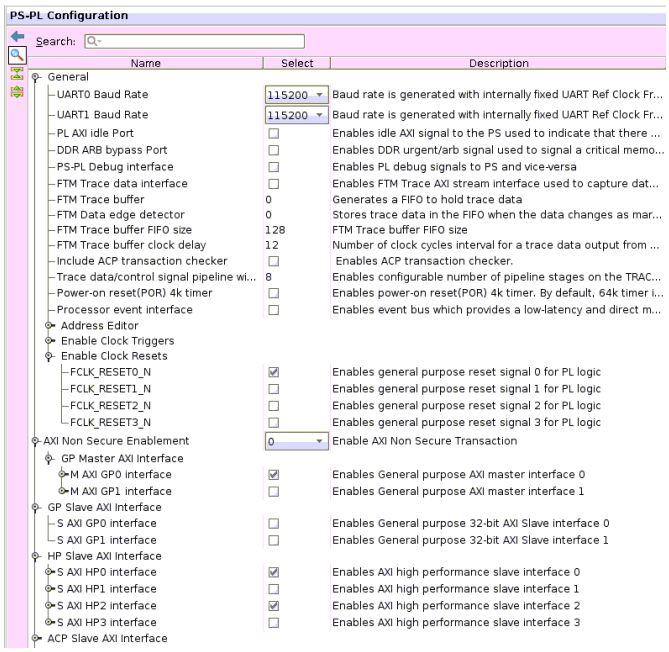


The Zynq PS is customized for Cardsharp applications to maximize processing power and memory bandwidth, clocking the CPU PLL at 800MHz and the DDR PLL at 533MHz. The IO PLL serves to clock several PS peripherals such as Ethernet, QSPI, eMMC and the PL clock. This clock (FCLK_CLK0) is programmed to run at 150MHz and clocks all the high performance master AXI interfaces to/from the PS. The Zynq PS is customized by double-clicking on its instance within the block design. The following images show its various configuration options, including the main overview of its internal architecture, clock configuration, PS-PL interface, peripherals I/O pins, MIO configuration, etc.




ZYNQ7 Processing System (5.5)
Documentation
Presets
IP Location
Import XPS Settings
Page Navigator
Zynq Block Design
PS-PL Configuration
Peripheral I/O Pins
MIO Configuration
Clock Configuration
DDR Configuration
SMC Timing Calculatio
Interrupts
Zynq Block Design
Summary Report
Processing System(PS)
Programmable Logic(PL)


Clock Configuration
Basic Clocking
Advanced Clocking
Input Frequency (MHz)
33.3333
CPU Clock Ratio
6:2:1
Search:
Component
Clock Source
Requested Frequ...
Actual Frequency...
Range(MHz)
Processor/Memory Clocks
CPU
DDR
IO Peripheral Clocks
SMC
QSPI
ENET0
ENET1
SDIO
SPI
CAN
PL Fabric Clocks
FCLK_CLK0
FCLK_CLK1
FCLK_CLK2
FCLK_CLK3
System Debug Clocks
TPIU
Timers
WDT
TTC0
TTC1


PS-PL Configuration
Search:
Name
Select
Description
General
UART0 Baud Rate
UART1 Baud Rate
PL AXI idle Port
DDR ARB bypass Port
PS-PL Debug interface
FTM Trace data interface
FTM Trace buffer
FTM Data edge detector
FTM Trace buffer FIFO size
FTM Trace buffer clock delay
Include ACP transaction checker
Trace data/control signal pipeline wi...
Power-on reset(POR) 4k timer
Processor event interface
Address Editor
Enable Clock Triggers
Enable Clock Resets
FCLK_RESET0_N
FCLK_RESET1_N
FCLK_RESET2_N
FCLK_RESET3_N
AXI Non Secure Enablement
GP Master AXI Interface
M AXI GP0 interface
M AXI GP1 interface
GP Slave AXI Interface
S AXI GP0 interface
S AXI GP1 interface
HP Slave AXI Interface
S AXI HP0 interface
S AXI HP1 interface
S AXI HP2 interface
S AXI HP3 interface
ACP Slave AXI Interface

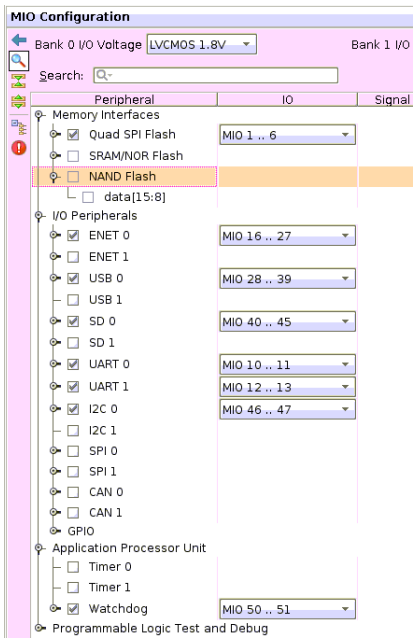

MIO Configuration
Bank 0 I/O Voltage
LVCMOS 1.8V
Bank 1 I/O
Search:
Peripheral
IO
Signal
Memory Interfaces
Quad SPI Flash
SRAM/NOR Flash
NAND Flash
data[15:8]
I/O Peripherals
ENET 0
ENET 1
USB 0
USB 1
SD 0
SD 1
UART 0
UART 1
I2C 0
I2C 1
SPI 0
SPI 1
CAN 0
CAN 1
GPIO
Application Processor Unit
Timer 0
Timer 1
Watchdog
Programmable Logic Test and Debug

Table 6. PS Configuration

The Zynq PS is configured with a number of peripherals connected over the MIO pins. The following image shows the Zynq Peripheral I/O Pins configuration screen showing the enabled peripheral units and their respective pin assignments, shadowed in green color.

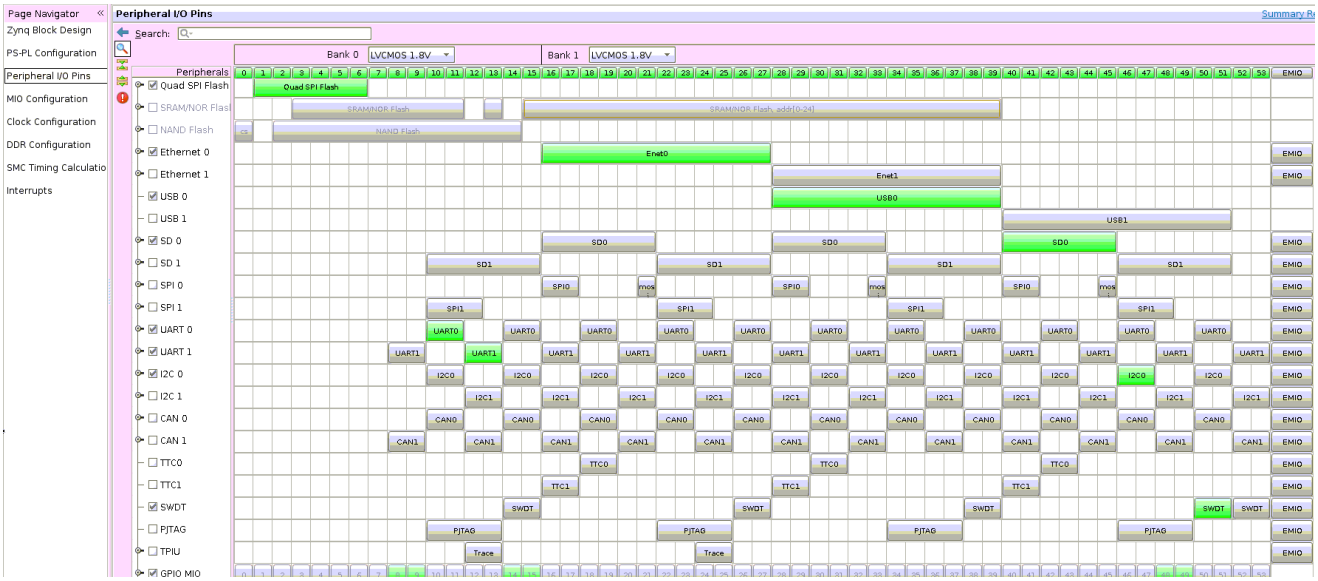


Illustration 1: MIO Peripheral I/O pins

MIO Pins	Peripheral	Direction	Description
MIO 0	Ethernet Reset	Out	Ethernet PHY reset.
MIO 1-6	Quad SPI Flash	Inout	Quad SPI Flash control signals.
MIO 7	USB Reset	Out	USB PHY reset.
MIO 8	GPIO	Out	GPS Reset (active low)
MIO 9	GPIO	In	GPS Lock-out Okay
MIO 10-11	UART0	In/Out	GPS UART0 Tx/Rx.
MIO 12-13	UART1	In/Out	GPS UART1 Rx
MIO 14-15	GPIO	Inout	
MIO 16-27	Ethernet	In/Out	Ethernet data & control to PHY.
MIO 28-39	USB	In/Out	USB PHY signals.
MIO 40-45	SD0	In/Out	SD signals (eMMC memory).
MIO 48-49	GPIO	In/Out	
MIO 50-51	Watchdog	In/Out	Watchdog timer external clock in & reset out.
MIO 52-53	Ethernet	In/Out	Ethernet serial control to PHY.

Table 7. Zynq PS Peripheral I/O Configuration

The DMA engines are controlled by embedded software running on the CPU's. The FMC DMA is a custom multichannel DMA that takes AXI-Stream data (s_axis_fmc) from the PL and routes it to one of 8 busmaster memory regions allocated within PS memory. In turn, data from 8 busmaster memory regions is routed to one of 8 Master AXI-Streams (m?_axis_fmc). The PCIe DMA is a similar DMA with only one channel enabled, that exchanges data coming and going from a host over the PCI Express link.

Port name	Direction	Format	Description
DDR	Out	DDR Interface	Dedicated PS DDR3 I/O pins.
FIXED_IO	In/Out	FIXED_IO Interface	Dedicated PS I/O Peripherals (MIO).
USBIND_0	In/Out	USBIND Interface	Dedicated PS I/O USB pins.
pl_rst	Out	[0]	Main PL reset.
s_axis_clk	In	[0]	Slave AXI-Stream clock.
clk150	Out	[0]	AXI clock.
clk75	Out	[0]	Wishbone Clock.
ctrl_reg	In/Out	ctrl_reg Interface	Control bus to Wishbone Master.
pps_int	In	[0]	PPS Interrupt to PS.
int_ext	In	[3:0]	External Interrupt Inputs.
s_axis_pcie	In	AXIS Interface	Slave AXI Stream from PCI Express link in PL to PCIe DMA.
m_axis_pcie	Out	AXIS Interface	Master AXI Stream from PCIe DMA to PL en-route to PCI Express link.
s_axis_fmc	In	AXIS Interface	Slave AXI Stream from PL to FMC DMA.
m*_axis_fmc	Out	AXIS Interface	Master AXI Streams from FMC Multichannel DMA out to the PL.
s_axis_alrt	In	AXIS Interface	Slave AXI Stream carrying Alerts from PL.

Table 8. Block Design interfaces & ports

The PS has two high performance slave AXI ports enabled, namely S_AXI_HP0, S_AXI_HP2, used by two custom DMA engines used to move streaming data from the PL to the PS memory. The s_axis_pcie & s_axis_fmc AXI-Stream interfaces take data from the PL into their respective DMA engines. In turn, the m_axis_pcie & m*_axis_fmc deliver data from the PS memory to the PL also in AXI-Stream format.

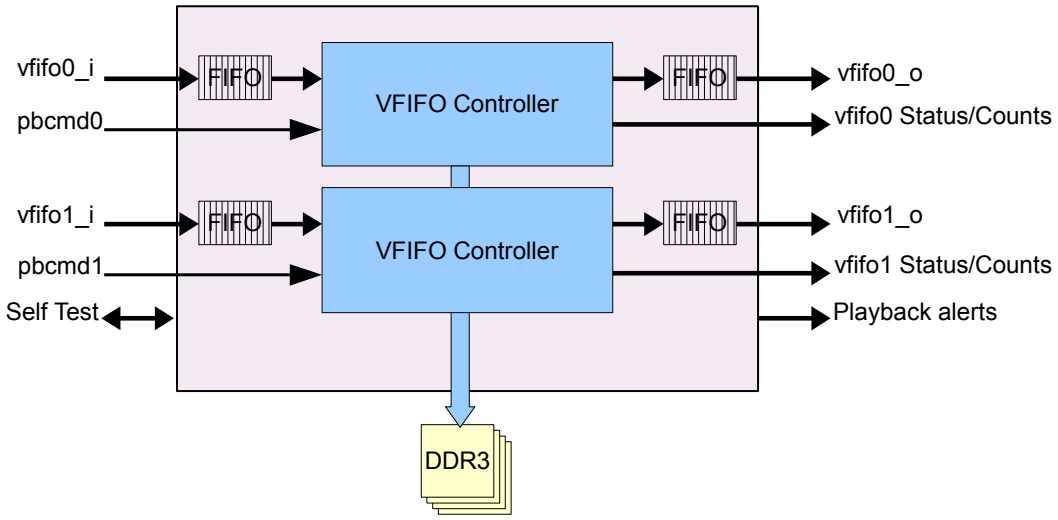
The AXI to Control Register component acts as a bridge between the AXI-Lite signaling from the PS to the Wishbone bus used in the PL register files. It takes register reads & writes from the PS as AXI-Lite transactions and transforms them into a format appropriate for the Wishbone Master to accept to and respond.

The Clock and Register Module (crm) within the Block Design has a PLL that generates clocks & resets that interface with the PS. A 75MHz clock is used for AXI-Lite transactions from the PS General Purpose Master AXI port (M_AXI_GP0) to the DMA's and the AXI to Control Register Bridge (axi_to_ctrl) component. A 150MHz clock is used to interface to the PS through High Performance Slave AXI ports (S_AXI_HP0/2).

The block design is scripted in Tcl and run when the project is being built by executing the ps_sys.tcl script that resides in the project directory under the bd/ subdirectory. If any modifications are done to the block design, it is suggested to write the changes by exporting it by using the menu option File/Export/Block Design. The block design has a wrapper that instantiates it, and it's managed by Vivado, so in case of modifications it will be updated automatically. The wrapper allows the block design to be synthesized in out-of-context mode if desired. This may help reduce synthesis time after the component has been synthesized once.

Multiqueue VFIFO

This component uses a bank of SDRAM memory as a buffer to implement either a “virtual FIFO” or an arbitrary pattern generator in two independent queues. The operating mode is selected during initialization. These operating modes support either a flow-through data architecture or an arbitrary waveform generator architecture for the design. The buffer depth is the size of the memory bank. Each of the external SDRAM devices is a bank of 512MB, arranged as 256Mx16, for a combined total of 256M x64 memory locations, or 2 GB capacity.



Each of the two queues operates with independent data input/output interfaces used in both operating modes, and independent Playback command interfaces used in the arbitrary waveform generator mode.

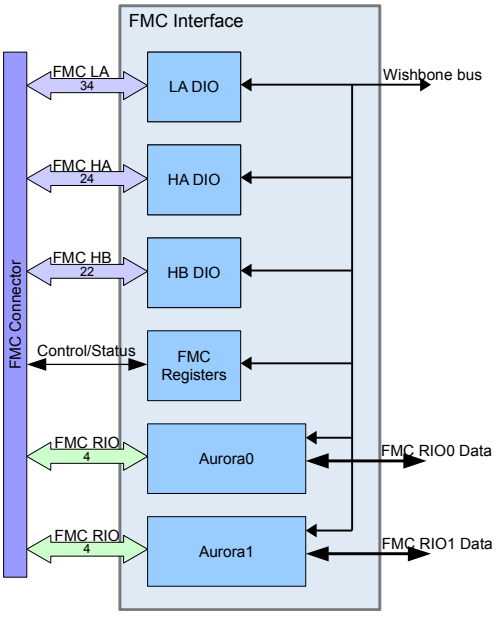
The flow-through mode or Virtual FIFO mode receives incoming data on the vfifo?_i_* interface, buffers the data in SDRAM and moves it to the vfifo?_o_* interface, waiting to be read.

Port name	Direction	Description
run	In	Enable data flow.
playback_en	In	Enable Playback mode.
test_en	In	Enable self-test mode.
test_error	Out	Self-test error results. Each bit corresponds to a SDRAM device.
pbcmd?_fifo_*	In	Playback Command FIFO Interface. Receives Command instructions for Arbitrary Waveform Generation mode.
vfifo?_i_*	In	Data Stream Interface. Receives streaming data to be buffered in VFIFO or data to be played back in Arbitrary Waveform Generation mode.
vfifo?_o_*	Out	Data Stream Interface. Sends buffered data out in VFIFO mode or playback data in Arbitrary Waveform Generation mode.
ddr3_*	In/Out	Interface to SDRAM DDR3 memory devices.

Table 9. Multiqueue VFIFO main interfaces

FMC Interface

The FMC Interface contains basic functionality to be able to test all the pins on the FMC connector. This functionality is bound to be modified and adapted to the FMC daughter card to be plugged in. In this configuration, there's register controlled Digital I/O's on each of the FMC data buses, namely LA, HA & HB ports. Additionally, the high speed serial I/O lines are controlled by two Aurora instances with their own data streams exposed to the top level.

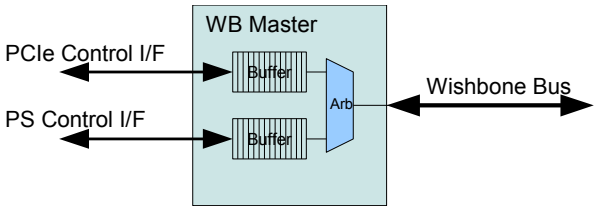


Port name	Direction	Description
wb_*	In/Out	Wishbone bus.
fmc_rio0_src*	In	Data stream input en-route to Aurora0 links.
fmc_rio0_dest	Out	Data stream output coming from Aurora0 links.
fmc_rio1_src*	In	Data stream input en-route to Aurora1 links.
fmc_rio1_dest	Out	Data stream output coming from Aurora1 links.
fmc_la_*	In/Out	FMA LA bus.
fmc_ha_*	In/Out	FMC HA bus.
fmc_hb_*	In/Out	FMC HB bus.
fmc_clk?_*	In/Out	FMC module-to-carrier clocks.
fmc_rio_*	In/Out	FMC High speed serial links (Aurora data)
fmc_gbtclk?	In	FMC GBT clocks (Aurora reference clocks).

Table 10. FMC main interfaces

Wishbone Master

This component is a bridge between two register control interfaces and the Wishbone Bus that reaches all registers in the PL. It can receive commands from the PS under embedded software control, or alternatively from the PCI Express bus if the system is plugged onto a host over the XMC connector. This would allow a system to control all PL registers from either control software, namely Zynq PS or a PC host in a transparent way. More information about Wishbone Master can be found in the K7 Logic Library guide.



Simulation

Simulating the Cardsharp logic is possible within Vivado, using its built-in simulation capabilities. In order to make the simulation faster, a number of Bus Functional Models (BFM) have been created to replace components that are too complex or simply unavailable for simulation; for example, the Zynq PS, the PCIe DMA and the Multiqueue components have been replaced by their BFM counterparts. The AXI BFM simulation model license is required to run the top level simulation. The license is non-free and should be acquired through Xilinx vendors.

The simulation suite has a hierarchy that makes it flexible for modifications and customization, counting with a top level "Test" that instantiates the actual testbench, and it includes accesses simulating software accesses from its various interfaces. Various components in the suite are written in Verilog HDL, making possible to access points deep through the hierarchy. VHDL doesn't allow for this.

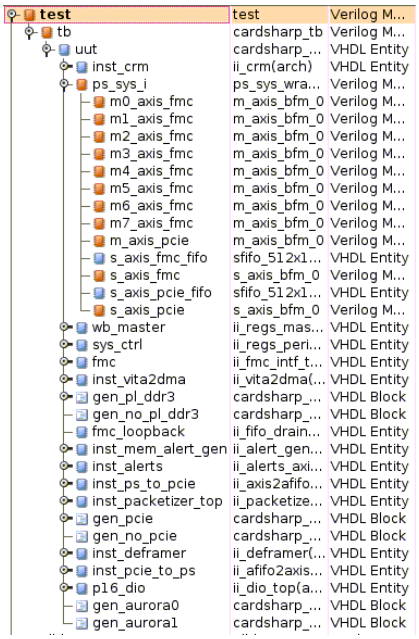
Test (test.v): Through hierarchical function calls, it accesses tasks in the BFMs to perform a number of functions, such as resets, register accesses, data streaming, etc. Testbench (cardsharp_tb.v): It instantiates the Unit Under Test and contains clock generators and other devices interfacing with the design, such as memories and analog parts like ADCs and/or DACs within a FMC model. UUT (cardsharp_top.vhd): Design top level. Instantiates all components or its correspondent BFMs. Block Design (ps_sys_i): This is a custom BFM that emulates the Zynq PS design, with all its streaming interfaces. PCIe (ii_pcie_wrapper.v): Custom BFM that emulates the PCI Express interface to the host PC. Multiqueue (ii_mq_pb.v): Custom BFM that emulates the Virtual FIFO multiqueue in flow-through mode. Pattern mode is not supported in the BFM.	Testbench Hierarchy 

Table 11. Simulation suite hierarchy

Testbench hierarchy

The following block diagram shows the testbench hierarchy and its main components. At the top, there's the Test which calls tasks within the Testbench. These tasks make use of particular components within the BFM's to provide simulation stimulus to the design.

Here's a brief description of the function calls to tasks within the different BFMs.

- tb.porb: Direct signal assignment to the PS BFM PORB port; active low.
- tb.ps7_rst(value): Sets Block Design pl_rst output to argument value.
- tb.pex_app_wr(address, data): PCIe slave write access to address, data on Wishbone Bus.
- tb.pex_app_rd(address, rdata): PCIe slave read access to address on Wishbone Bus, storing read value on rdata variable.
- tb.pex_app_rd_poll(address, bit_pos, expected): PCIe slave read access to address on Wishbone Bus, polling the selected bit position until its value matches the expected argument.
- tb.ps7_app_wr(address, data): Zynq PS slave write access to address, data on Wishbone Bus.
- tb.ps7_app_rd(address, rdata): Zynq PS slave read access to address on Wishbone Bus, storing read value on rdata variable.

- tb.ps7_app_rd_poll(address, bit_pos, expected): Zynq PS slave read access to address on Wishbone Bus, polling the selected bit position until its value matches the expected argument.
- tb.axi4_bfm_send(channel, axis_data, tlast): Sends a single AXI-Stream point to the corresponding AXI-Stream FMC DMA channel. Intended to be used by the testbench itself through the put_packet task.
- tb.put_packet(dma, channel, seed, length): Generates a ramp on the AXI-Stream Master interface selected by “dma” parameter (“FMC”, “PEX”). The seed is the initial value for the ramp; length selects the stream length in points.
- tb.put_vita_packet(dma, channel, packet count, frame size, stream Id, initial value): Generates a VITA packet on the selected AXI-Stream Master interface. Packet count is the VITA packet Id. Frame size is the VITA packet size. Stream Id is the VITA Stream Id field. Initial Value is the ramp initial value.
- tb.get_packet(dma): Reads the selected Slave AXI-Stream interface, either from “FMC” or “PEX” DMAs.

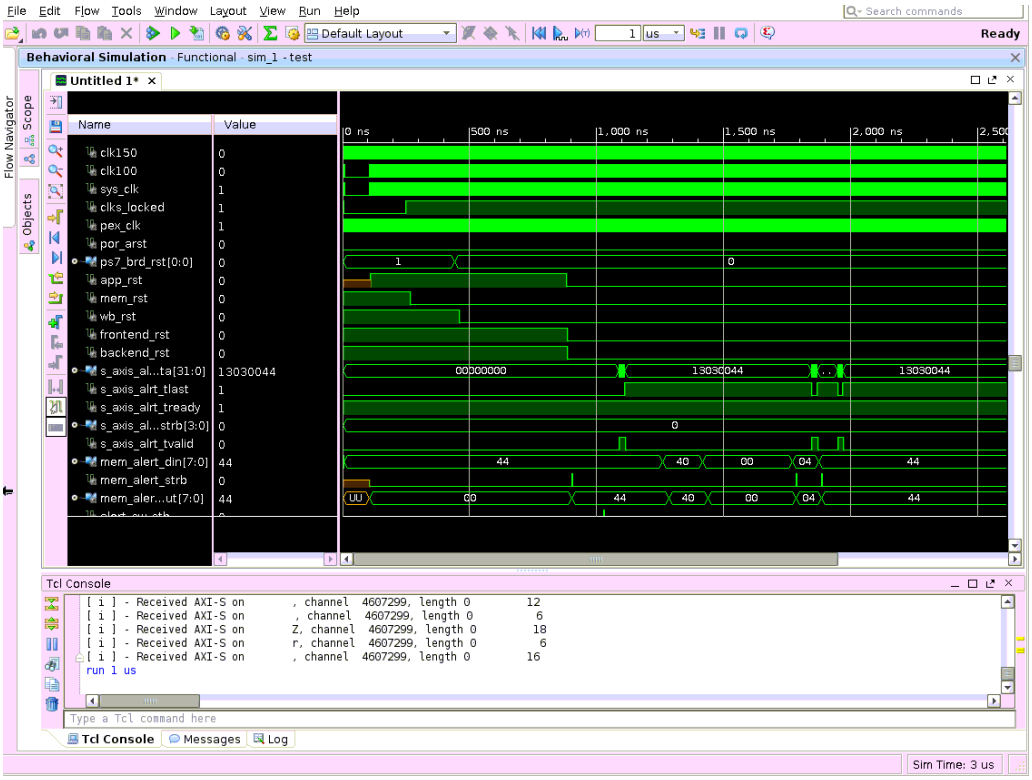


Illustration 2: Behavioral Simulation window

Cardsharp PL Memory Map

The memory map is shown for the components on the Wishbone Bus. The AXI to Control bus on Cardsharp is on base address 0x43C00000. Each component is mapped to a BASE address, with its registers offset from that BASE. The simulation define is the BASE address for that device used in the simulations. Individual registers with bit assignments are shown for each component on the Wishbone Bus.

Note: All addresses are word-aligned.

WB Base Address	WB Component	Simulation Define	Description	Module
0x000	Peripheral	MR_PRF	Peripheral registers : logic and hardware versions, resets, top level controls and status	Cardsharp
0x100-0x200	Reserved			
0x300	Packetizer	MR_PKT	Data packetizing controls	Cardsharp
0x400-0x500	Reserved			
0x600	Digital I/O	MR_DIO	Digital I/O for testing	Cardsharp
0x700	Matlab	MR_BSP	Matlab BSP General purpose registers	Cardsharp
0x800 - 0xb00	Reserved			
0xc00-0x1100	FMC	MR_FMC*	See FMC section	Cardsharp
0x1200-0x1300	P15 Aurora	MR_AU0/1	Aurora High Speed Serial Interfaces	Cardsharp
0x1400 - 0x1900	Reserved			
0x1A00	Application	MR_APP	Application logic Interface registers	Cardsharp
0x2000-0x3F00	IP Cores			Cardsharp

Table 12. Memory Map

Peripheral Registers (WB Device 0)

These are the top level registers used for system functions.

WB Base	WB Address	Register	Simulation Define	R/W	Description	Modules
0x000	0x00	Information	MR_PRF_INFO	R	Hardware and logic version information.	
	0x01	Reset	MR_PRF_RST	R/W	Reset control	
	0x03	Sub revision	MR_PRF_SUB_REV	R	Sub revision	
	0x04	Bypass VITA padder	MR_PRF_BYPASSV	R/W	Bypass VITA padding in Velocia packets. Test feature.	
	0x07	PL DDR3	MR_PRF_DDR		DRAM power down controls	
	0x08-0x0A	Define PID	MR_PRF_DEF_PID		Peripheral ID assignment for stream 0	
	0x0B	alert_enable	MR_ALR_ENAB	R/W	Alert enables.	
	0x0C	control	MR_ALR_CTRL	R/W	Alert monitor controls.	
	0x0D	sw_alert	MR_ALR_SW	R/W	Software Alert	
	0x0E	alert_clr	MR_ALR_CLR	R/W	Clear alert FIFO	
	0x0F	alert_cnt	MR_ALR_CNT	R/W	Number of Alerts to process	
	0x10	QSFP0 control	MR_PRF_QSFP0_CTRL	R/W	QSFP0 control & status	
	0x11	QSFP0 I2C	MR_PRF_QSFP0_I2C	R/W	QSFP0 I2C clock control	
	0x12	QSFP1 control	MR_PRF_QSFP1_CTRL	R/W	QSFP1 control & status	
	0x13	QSFP1 I2C	MR_PRF_QSFP1_I2C	R/W	QSFP1 I2C clock control	
	0x14	QSFP XO I2C	MR_PRF_QSFP_XO_I2C	R/W	QSFP I2C clock generator	

Table 13. System Peripherals Registers

Information (MR_PRF_INFO, Base+0x00)
This register provides hardware and logic revision information.

Bits	Field	R/W	Description
15:0	revision	R	Logic revision code. Split in two byte words: major:minor. (ie. 0x0102=v1.2).
19:16	cfg	R	Hardware configuration code (variant code; ie. AC coupled)
23:20	hw_rev	R	Hardware revision (Rev. A, B, etc.)
27:24	hw_type	R	Hardware type code (Module Id, ie. SBC=2, etc.)
29:28	fpga_type	R	FPGA type. 0x01 = 045 0x10 = 100

31:30	unused		

Table 14. System Info Register

Reset (MR_PRF_RST, Base+0x01)
This register provides hardware reset for the FPGA (soft reset)

Bits	Field	R/W	Description	Modules
0	app_rst	R/W	Application reset. Default=1.	Cardsharp
1	Green LED	R/W	LED controls. '1' = on (default)	Cardsharp
2	Front panel LED	R/W	LED controls. '1' = on (default)	Cardsharp
3	Unused	RO		
4	run	R/W	Enable backend data flow. '0'=off (default).	Cardsharp
31:5	-			

Table 15. System Reset Register

Sub Revision (MR_PRF_SUB_REV, Base+0x03)
This register provides the sub revision code which is used to track logic builds.

Bits	Field	R/W	Description	Modules
23:0	-			
31:24	sub_rev	R	Sub revision	Cardsharp

Table 16. System Sub Revision Register

Bypass VITA padding in Velocia packets (MR_PRF_BYPASSV, Base+0x4)
VITA padding packets within Velocia packets allows an integer number of VITA packets to fit in a Velocia packet. For test reasons, this feature may be bypassed.

Bits	Field	R/W	Description	Modules
0	Bypass_vita_pad	R/W	Bypass VITA padder (default = 0)	
1	VITA header error	R	Error on VITA header	
2	VITA trailer error	R	Error on VITA trailer	
31:3	-	-	-	

Table 17. System Sub Revision Register

DDR3 Control and Status (MR_PRF_DDR Base+0x07)
This register provides the DDR2 bank power control, VFIFO mode control, and PHY initialization status.

Bits	Field	R/W	Description	Modules
5:4	Ddr3_init_done	R	DRAM PHY initialization status. '1' = PHY init completed successfully.	
13:12	mem_test_en	R/W	Enable memory test mode	
17:16	mem_test_error	R	Memory test status. '1' = error.	
31:20	-			

Table 18. System DDR3 DRAM Control and Status

Outgoing PID Defines (MR_PRF_DEF_PIDx Base+0x08... +0x0B)
This register defines the PID for Velocia packet streams. Multiple streams may be assigned beginning with register 0x8 up to register 0xB.

Offset	Description	Modules
0x8	Test LoopBack PID	
0x9	DAC data and playback command PID	

Table 19. Outgoing PID Map

Bits	Field	R/W	Description	Modules
23:0	-			
31:24	def_pid_addr	R/W	PID address. (default = 0x0)	

Table 20. System PID Define Register

Alert Enables (MR_PRF_AL_EN, Base+0xB)
This register enables each alert.

Bits	Field	R/W	Default	Description
31 : 0	alert_enables	R/W	0 = off	Alert enables. One bit per alert source.
others	-			

Table 21. Alert Monitor Enables Register

Alert Defines

Bits	Alert	Alert Data Word	Description	Modules
0	timestamp_rollover	(x"1303000" & "000" & timestamp_rollover)	Alert timestamp rollover.	All
1	alert_sw_stb	alert_sw	Software Alert.	All
2	Tag_*	tag_rep_value(15 downto 8) & tag_load_value(15 downto 8) & tag_rep_value(7 downto 0) & tag_load_value(7 downto 0	VFIFO Arbitrary Waveform Generator tags events.	Cardsharp
3	mem_alert_dout	(x"1303000" & mem_alert_dout)	VFIFO status alerts from ii_alert_gen	
31 : 4	-			

Table 22. Alert Monitor Defined Alerts

Alert Controls (MR_PRF_AL_CTRL, Base+0xC)
This register provides reset and timestamp enable functions for the Alert monitor.

Bits	Field	R/W	Default	Description
0	timestamp_run	R/W	0	Enable Alert monitor timestamp.
1	alert_fifo_rst	R/W	0	Alert monitor reset. Clears all pending alerts and the FIFO.
others	-			

Table 23. Alert Monitor Controls

Software Alert (MR_PRF_AL_SW, Base+0xD)
This register fires a software alert whenever written to.

Bits	Field	R/W	Default	Description
31:0	sw_data	R/W	0	Software alert

Table 24. Software Alert

Clear Alerts (MR_PRF_AL_CLR, Base+0xE)

Bits	Field	R/W	Default	Description
31:0	alert_clr	R/W	0	Clear alerts.

Table 25. Alert Monitor Controls

Alerts count (MR_PRF_AL_CNT, Base+0xF)
This register sets the number of alerts inputs that will be processed

Bits	Field	R/W	Default	Description
31:0	alert_cnt	R/W	0	Number of alert inputs to be processed, ignoring the others.

Table 26. Alert Monitor Controls

QSFP0 port control/status register (MR_PRF_DEF_PIDx Base+0x10)
This is the QSFP0 port control status register.

Bits	Field	R/W	Default	Description	Modules
0	qsfp_modesel_n	R/W		When “low”, the module responds to 2-wire serial communication. When “high”, the module shall not respond or acknowledge any 2-wire communication	
3:1	-			Unused	
4	qsfp_reset_n	R/W		A low level on this pin resets the qsfp module	
11:3	-			Unused	
12	qsfp_lpmode	R/W		Setting this bit high sets the qsfp module in low power mode. When low, the qsfp module is in high power mode	
15:13	-			Unused	
16	qsfp_int_n	R		When low, indicates possible module fault	

19:17	-			Unused	
20	qsfp_modpres_n	R		A high on this bit indicates module absent	
23:21	-			Unused	
31:24	-			Unused	

Table 27. QSFP port control/status register

QSFP0 port I2C register (MR_PRF_DEF_PIDx Base+0x11)
This is the QSFP0 port I2C register.

Bits	Field	R/W	Default	Description	Modules
3:0	qsfp_sda_o	R/W		I2C data out	
7:4	qsfp_scl	R/W		I2C clock	
11:8	qsfp_sda_i	R		I2C data in	
15:12	qsfp_scl_i	R		I2C clock (readback)	
31:16	-			Unused	

Table 28. QSFP port I2C register

QSFP1 port control/status register (MR_PRF_DEF_PIDx Base+0x12)
This is the QSFP1 port control status register.

Bits	Field	R/W	Default	Description	Modules
0	qsfp_modesel_n	R/W		When “low”, the module responds to 2-wire serial communication. When “high”, the module shall not respond or acknowledge any 2-wire communication	
3:1	-			Unused	
4	qsfp_reset_n	R/W		A low level on this pin resets the qsfp module	
11:3	-			Unused	
12	qsfp_lpmode	R/W		Setting this bit high sets the qsfp module in low power mode. When low, the qsfp module is in high power mode	
15:13	-			Unused	
16	qsfp_int_n	R		When low, indicates possible module fault	
19:17	-			Unused	
20	qsfp_modpres_n	R		A high on this bit indicates module absent	
23:21	-			Unused	
31:24	-			Unused	

Table 29. QSFP port control/status register

QSFP1 port I2C register (MR_PRF_DEF_PIDx, Base+0x13)
This is the QSFP1 port I2C register.

Bits	Field	R/W	Default	Description	Modules
3:0	qsfp_sda_o	R/W		I2C data out	
7:4	qsfp_scl	R/W		I2C clock	
11:8	qsfp_sda_i	R		I2C data in	
15:12	qsfp_scl_i	R		I2C clock (readback)	
31:16	-			Unused	

Table 30. QSFP port I2C register

QSFP SIO XO I2C register (MR_PRF_QSFP_XO_I2C, Base+0x14)
This is the QSFP SIO XO I2C register that controls the QSFP reference clock generator.

Bits	Field	R/W	Default	Description	Modules
0	qsfp_sio_sdo	R/W		XO I2C data out	
1	qsfp_sio_sck	R/W		XO I2C clock	
2	qsfp_sio_sdi	R		XO I2C data in	
3	qsfp_sclk_i	R		XO I2C clock (readback)	
4	qsfp_sio_intr			XO Interrupt	
31:5	-			Unused	

Table 31. SIO XO I2C register

PPS Trigger Arm register (MR_PRF_PPS, Base+0x15)
Arm PPS trigger waits until a PPS event to assert the PPS trigger. When disarmed it waits for the next PPS event to deassert the trigger.

Bits	Field	R/W	Description	Modules
0	arm_pps_trig	R/W	Arm PPS trigger (Default=0, not armed)	
31:1	-	R/W		

Packetizer Registers (WB Device 3)

These are the Velocia packetizer control registers. These are NOT associated with the VITA packetizers.

WB Base	WB Address	Register	Simulation Define	R/W	Description
0x300	0x00	pkt_data_ch_en	MR_PKT_DATA_CH_EN	R/W	Velocia packetizer data channel enables
	0x01	aux_hdr2	MR_PKT_AUX_HDR	R/W	Second header word for Velocia packets
	0x02	alert_pkt_hdr	MR_PKT_ALRT_HDR	R/W	Alert Velocia packet PID and size
	0x03	data_pkt_hdr	MR_PKT_DATA_HDR	R/W	Data Velocia packet PID and size
	others				
	0x22	force_pkt_size	MR_PKT_FRC_CH_SIZE	R/W	Force packet size
	0x23	Timer	MR_PKT_TIMER	R/W	Timeout timer

Table 32. Velocia Packetizer Component Registers

Packetizer Data Channel Enables (MR_PKT_DATA_CH_EN, Base+0x0)
This register enables each Velocia packetizer data channel.

Bits	Field	R/W	Default	Description	Modules
num_data_pkt_ch-1 : 0	pkt_data_ch_en	R/W		Packetizer data channel enables.	
others	-				

Table 33. Velocia Packetizer Data Channel Enable Register

Auxiliary header word (MR_PKT_AUX_HDR, Base+0x1)
This register defines the second word in the Velocia packets header.

Bits	Field	R/W	Default	Description
31 : 0	aux_hdr2	R/W	0	Auxiliary second header word for Velocia packets.

Table 34. Velocia Packetizer Auxiliary Header Register

Alert Velocia Packet Header (MR_PKT_ALRT_HDR, Base+0x2)
This register defines the alert Velocia packet header.

Bits	Field	R/W	Default	Description
23 : 0	alert_pkt_size	R/W	0x28	Alert packet size
31 : 24	alert_pd_addr	R/W	0xff	Alert packet Peripheral ID
others	-			

Table 35. Velocia Packetizer Alert Header Register

Data Velocia Packet Header (MR_PKT_DATA_HDR, Base+0x3 ..x"2+num_data_pkt_ch")
These registers, one for each data packet channel, define the Velocia header.

Offset	Description	Modules
00x03	ADC data	
00x04	Test LoopBack PID	

Table 36. Incoming PID Map

Bits	Field	R/W	Default	Description
23 : 0	ch_pkt_size(i)	R/W		Maximum packet size for ith data Velocia channel.
31 : 24	pd_addr(i)	R/W		Peripheral ID for ith data Velocia channel .
others	-			

Table 37. Velocia Packetizer Data Header Register

Force Velocia Packet Size (MR_PKT_FRC_CH_SIZE, Base+0x22)
This register forces the maximum size for each Velocia packetizer data channel.

Bits	Field	R/W	Default	Description	Modules
num_data_pkt_ch-1 : 0	force_pkt_size	R/W		Force maximum packet size for each data channel when set. Otherwise, if the data available is less than the max size, a smaller packet is constructed.	
others	-				

Table 38. Force Velocia Packet Size Per Channel Register

Timer (MR_PKT_TIMER, Base+0x23)
This register sets a timer inside packetizer to avoid sending too many small packets and potentially slowing down the host processing. By default the timer is set to 1ms, but may be programmed by software.

Bits	Field	R/W	Default	Description	Modules
21:0	timer	R/W	250000 clock cycles (at 250MHz default = 1ms).	Packetizer timer.	
others	-				

Table 39. Packetizer Timer

P16 DIO Registers

The single ended Digital I/O registers are concatenated as follows: positive on even bits, negative on odd bits, ie. p16_dio_p[6:0] are on even bits, p16_dio_n[6:0] on odd bits.

For example:

Bits[31:6]	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
…	p16_dio_n[2]	p16_dio_p[2]	p16_dio_n[1]	p16_dio_p[1]	p16_dio_n[0]	p16_dio_p[0]

P16 DIO register (WB Device 6)

These are the registers for P16 DIO bus.

WB Base	WB Address	Register	Simulation Define	R/W	Description
0x600	0x00		MR_DIO_DOUT	R/W	Lower 32 bit word of the P16 DIO bus. The entire DIO bus is updated when writing to this register.
	0x01	Reserved			
	0x02		MR_DIO_OE	R/W	Output enable of the lower 32 bit word of the P16 DIO bus

Table 40. P16 DIO register

P15 Aurora 0 Registers (WB Device 18)

These are the registers for Aurora port 0.

Base	WB Address	Register	Simulation Define	R/W	Description
0x1100	0x00		MR_AU1_TEST_CTRL	R/W	
	0x01		MR_AU1_CTRL_STAT	R/W	
	0x02		MR_AU1_CMD_WR	R/W	
	0x03		MR_AU1_CMD_RD	R/W	
	others				

Table 41. P15 Aurora Port 1 Component Registers

P15 Aurora Port 1 Test Control (MR_AU1_TEST_CTRL, Base+0x00)
These are controls and status for P15 Aurora port 0.

Bits	Field	R/W	Default	Description	Modules
0	tx_test_gen_en	R/W	0 = off	Enable test generator for transmission.	
1	rx_test_chk_en	R/W	0 = off	Enable test generator for receive.	
15:2					
31:16	test_errors	R		Test error count	

Table 42. P15 Aurora 0 Test Control Register

P15 Aurora Port 1 Control/Status (MR_AU1_CTRL_STAT, Base+0x01)
These are controls and status for P15 Aurora port 0.

Bits	Field	R/W	Default	Description	Modules
0	Gtxreset_n	R/W	0 = reset	MGT reset, active low.	
1	power_down	R/W	1 = on	MGT power down. (Turned off by default)	
2	run	R/W	0 = off	Aurora interface run.	
5:3	loopback	R/W	000	000 = Disable loopback 001 = Parallel 010 = Serial	
6	error_clr	R/W	0 = off	Clear port error.	
7	tx_channel_en	R/W	0 = off	Enable transmit channel.	
8	rx_channel_en	R/W	0 = off	Enable receive channel.	
22:9	-				
23	hard_error	R		Hard error. Link lost due to serious disruption.	
24	soft_error	R		Soft error such as bit error.	

25	frame_error	R		Frame error from Aurora.	
29:26	lane_up	R		Number of lane the Aurora port is using.	
30	channel_up	R		The Aurora channel is active.	
31	pll_locked	R		PLL for MGT is locked.	

Table 43. P15 Aurora 0 Control/Status Register

P15 Aurora Port 1 Sub-channel Write Port (MR_AU1_CMD_WR, Base+0x02)
This is the sub-channel write port for P15 Aurora 0.

Bits	Field	R/W	Default	Description	Modules
23:0	usr_cmd_wr_data	R/W	0	Command write data.	
29:24	usr_cmd_wr_addr	R/W	0	Command write address.	
30	usr_cmd_wr_rdn	R/W	0=write	Command read/write control. 0= write, 1 = read.	
31	cmd_ch_rdy	R		Command sub-channel is ready.	

Table 44. P15 Aurora 0 Sub-channel Write Register

P15 Aurora Port 1 Sub-channel Read Port (MR_AU1_CMD_RD, Base+0x03)
This is the sub-channel read port for P15 Aurora 0.

Bits	Field	R/W	Default	Description	Modules
23:0	usr_cmd_rd_data	R		Command read data.	
29:24	usr_cmd_rd_addr	R		Command read address.	
30	-				
31	usr_cmd_rd_vld	R		Command sub-channel is read data is valid.	

Table 45. P15 Aurora 0 Sub-channel Read Register

P15 Aurora 1 Registers (WB Device 19)

These are the registers for Aurora port 1.

Base	WB Address	Register	Simulation Define	R/W	Description
0x1100	0x00		MR_AU1_TEST_CTRL	R/W	
	0x01		MR_AU1_CTRL_STAT	R/W	
	0x02		MR_AU1_CMD_WR	R/W	
	0x03		MR_AU1_CMD_RD	R/W	
	others				

Table 46. P15 Aurora Port 1 Component Registers

P15 Aurora Port 1 Test Control (MR_AU1_TEST_CTRL, Base+0x00)
These are controls and status for P15 Aurora port 1.

Bits	Field	R/W	Default	Description	Modules
0	tx_test_gen_en	R/W	0 = off	Enable test generator for transmission.	
1	rx_test_chk_en	R/W	0 = off	Enable test generator for receive.	
15:2					
31:16	test_errors	R		Test error count	

Table 47. P15 Aurora 1 Test Control Register

P15 Aurora Port 1 Control/Status (MR_AU1_CTRL_STAT, Base+0x01)
These are controls and status for P15 Aurora port 1.

Bits	Field	R/W	Default	Description	Modules
0	Gtxreset_n	R/W	0 = reset	MGT reset, active low.	
1	power_down	R/W	1 = on	MGT power down. (Turned off by default)	
2	run	R/W	0 = off	Aurora interface run.	
5:3	loopback	R/W	000	000 = Disable loopback 001 = Parallel 010 = Serial	
6	error_clr	R/W	0 = off	Clear port error.	
7	tx_channel_en	R/W	0 = off	Enable transmit channel.	
8	rx_channel_en	R/W	0 = off	Enable receive channel.	
22:9	-				
23	hard_error	R		Hard error. Link lost due to serious disruption.	
24	soft_error	R		Soft error such as bit error.	
25	frame_error	R		Frame error from Aurora.	
29:26	lane_up	R		Number of lane the Aurora port is using.	
30	channel_up	R		The Aurora channel is active.	
31	pll_locked	R		PLL for MGT is locked.	

Table 48. P15 Aurora 1 Control/Status Register

P15 Aurora Port 1 Sub-channel Write Port (MR_AU1_CMD_WR, Base+0x02)
This is the sub-channel write port for P15 Aurora 1.

Bits	Field	R/W	Default	Description	Modules
23:0	usr_cmd_wr_data	R/W	0	Command write data.	
29:24	usr_cmd_wr_addr	R/W	0	Command write address.	
30	usr_cmd_wr_rdn	R/W	0=write	Command read/write control. 0= write, 1 = read.	
31	cmd_ch_rdy	R		Command sub-channel is ready.	

Table 49. P15 Aurora 1 Sub-channel Write Register

P15 Aurora Port 1 Sub-channel Read Port (MR_AU1_CMD_RD, Base+0x03)
This is the sub-channel read port for P15 Aurora 1.

Bits	Field	R/W	Default	Description	Modules
23:0	usr_cmd_rd_data	R		Command read data.	
29:24	usr_cmd_rd_addr	R		Command read address.	
30	-				
31	usr_cmd_rd_vld	R		Command sub-channel is read data is valid.	

Table 50. P15 Aurora 1 Sub-channel Read Register

Application logic Interface registers (WB Device 21)

This is the registers for Application logic Interface.

WB Base	WB Address	Register	Simulation Define	R/W	Description
0x1500	0x00		MR_APP_RUN	R/W	

Table 51. Application logic interface registers

Application run register (MR_APP_RUN, Base+0x00)
Application run register.

Bits	Field	R/W	Default	Description	Modules
0	run	R/W	-	Run	
31:1	-	-		-	

Table 52. Application run Register

FMC Memory Map

FMC Registers

The memory map for the FMC components on the Wishbone Bus is shown below. Some components are common to all the FMC modules, whereas others are card specific.

WB Base Address	WB Component	Simulation Define	Description	Module
0xc00	FMC	MR_FMC_IF	FMC interface registers	Cardsharp
0xd00	FMC	MR_FMC_LA_DIO	FMC LA DIO register	Cardsharp
0xe00	FMC	MR_FMC_HA_DIO	FMC HA DIO register	Cardsharp
0xf00	FMC	MR_FMC_HB_DIO	FMC HB DIO register	Cardsharp
0x1000	FMC	MR_FMC_RIO0	Aurora core 0 within FMC interface	Cardsharp
0x1100	FMC	MR_FMC_RIO1	Aurora core 1 within FMC interface	Cardsharp
0x1200	FMC	MR_FMC_RIO2	Aurora core 2 within FMC interface	Cardsharp

Table 53. FMC Memory Map

FMC status and configuration Registers (WB Device 12)

These are the registers for FMC interface.

WB Base	WB Address	Register	Simulation Define	R/W	Description
0xc00	0x00		MR_FMC_CTRL	R/W	FMC control and status
	0x01		MR_FMC_I2C_IF	R/W	FMC I2C interface
	0x02		MR_FMC_ID	R	FMC ID
	0x03		MR_FMC_CLK_CFG	R/W	FMC BIDIR clock control and status
	0x04		MR_FMC_CLK0_M2C_STS	R	FMC clock 0 m2c status
	0x05		MR_FMC_CLK1_M2C_STS	R	FMC clock 1 m2c status
	0x06		MR_FMC_CLK2_M2C_STS	R	FMC clock 2 m2c status
	0x07		MR_FMC_CLK3_M2C_STS	R	FMC clock 3 m2c status
	0x08		MR_FMC_CLK2_C2M_STS	R	FMC clock 2 c2m status
	0x09		MR_FMC_CLK3_C2M_STS	R	FMC clock 3 c2m status

Table 54. FMC status and configuration registers

FMC control register (MR_FMC_CTRL, Base+0x00)
These are FMC control and status bits.

Bits	Field	R/W	Default	Description	Modules
0	fmc_present_n	R	-	FMC present (active low)	Cardsharp
1	fmc_vadj_en_n_o	W	1	Enable FMC VADJ (active low)	Cardsharp
1	fmc_vadj_en_n_i	R	1	Actual status of FMC VADJ Enable pin (active low)	Cardsharp
2	fmc_vadj_forced	R		FMC VADJ is forced by the hardware	Cardsharp
3	fmc_vadj_pwr_gd	R		FMC VADJ power good status	Cardsharp
6:4	fmc_vadj_lvl	R		Selected VADJ level 000 => 1.2V 001 => 1.35V 010 => 1.5V 011 => 1.8V 100 => 2.5V 111 - 101 => N/A (spare)	Cardsharp
15:7	-				
16	fmc_pg_m2c_n	R	-	FMC power good M2C (active low)	Cardsharp
17	fmc_pg_c2m_n	R/W	1	FMC power good C2M (active low)	Cardsharp
31:18	-				

Table 55. FMC control register

FMC I2C interface (MR_FMC_I2C_IF, Base+0x01)
This is the I2C interface to the FMC.

Bits	Field	R/W	Default	Description	Modules
0	fmc_sdo	R/W	-	FMC I2C data out	Cardsharp
1	fmc_scl	R/W	-	FMC I2C clock	Cardsharp
2	fmc_sdi	R	-	FMC I2C data in	Cardsharp
3	-	R		FMC I2C clock readback	Cardsharp
31:4	-				

Table 56. FMC I2C interface

FMC ID (MR_FMC_ID, Base+0x2)
This register has the FMC ID.

Bits	Field	R/W	Default	Description	Modules
31:0	fmc_id	R		FMC ID	Cardsharp

Table 57. FMC ID

FMC BIDIR Clock control and status (MR_FMC_CLK_CFG, Base+0x3)
This register has the FMC BIDIR clock control and status bits.

Bits	Field	R/W	Default	Description	Modules
0	fmc_clk_dir	R		FMC BIDIR clock direction (0=M2C, 1=C2M)	Cardsharp
1	fmc_clk_2_3_en	R/W	0	FMC clock 2 & 3 driver enable	Cardsharp
2	fmc_clk2_sel	R/W	0	FMC clock 2 source select 0=DIFF, 1=SE	Cardsharp
3	fmc_clk3_sel	R/W	0	FMC clock 3 source select 0=DIFF, 1=SE	Cardsharp
31:4	-				

Table 58. FMC BIDIR Clock Register

FMC Clock0 m2c status (MR_FMC_CLK0_M2C_STS, Base+0x4)

Bits	Field	R/W	Default	Description	Modules
9:0	fmc_clk0_m2c_freq	R		FMC clock0 M2C frequency in MHz	Cardsharp
31:10	-				

Table 59. FMC Clock0 m2c Register

FMC Clock1 m2c status (MR_FMC_CLK1_M2C_STS, Base+0x5)

Bits	Field	R/W	Default	Description	Modules
9:0	fmc_clk1_m2c_freq	R		FMC clock1 M2C frequency in MHz	Cardsharp
31:10	-				

Table 60. FMC Clock1 m2c Register

FMC Clock2 m2c status (MR_FMC_CLK2_M2C_STS, Base+0x6)

Bits	Field	R/W	Default	Description	Modules
9:0	fmc_clk2_m2c_freq	R		FMC clock2 M2C frequency in MHz	Cardsharp
31:10	-				

Table 61. FMC Clock2 m2c Register

FMC Clock3 m2c status (MR_FMC_CLK3_M2C_STS, Base+0x7)

Bits	Field	R/W	Default	Description	Modules
9:0	fmc_clk3_m2c_freq	R		FMC clock3 M2C frequency in MHz	Cardsharp
31:10	-				

Table 62. FMC Clock3 m2c Register

FMC Clock2 c2m status (MR_FMC_CLK2_C2M_STS, Base+0x8)

Bits	Field	R/W	Default	Description	Modules
9:0	fmc_clk2_c2m_freq	R		FMC clock2 C2M frequency in MHz	Cardsharp
31:10	-				

Table 63. FMC Clock2 c2m Register

FMC Clock3 c2m status (MR_FMC_CLK3_C2M_STS, Base+0x9)

Bits	Field	R/W	Default	Description	Modules
9:0	fmc_clk3_c2m_freq	R		FMC clock3 C2M frequency in MHz	Cardsharp
31:10	-				

Table 64. FMC Clock3 c2m Register

FMC DIO Registers

The single ended Digital I/O registers are concatenated as follows: positive on even bits, negative on odd bits, ie. fmc_la_p[33:0] are on even bits, fmc_la_n[33:0] on odd bits.

For example:

Bits[31:6]	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
…	fmc_la_n[2]	fmc_la_p[2]	fmc_la_n[1]	fmc_la_p[1]	fmc_la_n[0]	fmc_la_p[0]

FMC LA DIO register (WB Device 13)

These are the registers for FMC LA DIO bus.

WB Base	WB Address	Register	Simulation Define	R/W	Description
0xd00	0x00		MR_FMC_LA_DOUT_L	R/W	Lower 32 bit word of the FMC LA DIO bus. The entire DIO bus is updated when writing to this register.
	0x01		MR_FMC_LA_DOUT_H	R/W	Middle 32 bit word of the FMC LA DIO bus
	0x02		MR_FMC_LA_DOUT_V	R/W	Upper 2 bit word of the FMC LA DIO bus
	0x04		MR_FMC_LA_OE_L	R/W	Output enable of the lower 32 bit word of the FMC LA DIO bus
	0x05		MR_FMC_LA_OE_H	R/W	Output enable of the middle 32 bit word of the FMC LA DIO bus
	0x06		MR_FMC_LA_OE_V	R/W	Output enable of the upper 2 bit word of the FMC LA DIO bus

Table 65. FMC LA DIO register

FMC HA DIO register (WB Device 14)

These are the registers for FMC HA DIO bus.

WB Base	WB Address	Register	Simulation Define	R/W	Description
0xe00	0x00		MR_FMC_HA_DOUT_L	R/W	Low 32 bit word FMC HA DIO bus. The entire DIO bus is updated when writing to this register.
	0x01		MR_FMC_HA_DOUT_H	R/W	High 32 bit word FMC HA DIO bus
	0x02		MR_FMC_HA_OE_L	R/W	Output enable of the low 32-bit word FMC HA DIO bus
	0x03		MR_FMC_HA_OE_H	R/W	Output enable of the high 16-bit word FMC HA DIO bus

Table 66. FMC HA DIO register

FMC HB DIO register (WB Device 15)

These are the registers for FMC HB DIO bus.

WB Base	WB Address	Register	Simulation Define	R/W	Description
0xf00	0x00		MR_FMC_HB_DOUT_L	R/W	Low 32 bit word FMC HB DIO bus. The entire DIO bus is updated when writing to this register.
	0x01		MR_FMC_HB_DOUT_H	R/W	High 32 bit word FMC HB DIO bus
	0x02		MR_FMC_HB_OE_L	R/W	Output enable of the low 32-bit word FMC HB DIO bus
	0x03		MR_FMC_HB_OE_H	R/W	Output enable of the high 12-bit word FMC HB DIO bus

Table 67. FMC HB DIO register

FMC Aurora 0 Registers (WB Device 16)

These are the registers for Aurora port 0.

WB Base	WB Address	Register	Simulation Define	R/W	Description
0x1000	0x00		MR_FMC_RIO0_TEST_CTRL	R/W	
	0x01		MR_FMC_RIO0_CTRL_STAT	R/W	
	0x02		MR_FMC_RIO0_CMD_WR	R/W	
	0x03		MR_FMC_RIO0_CMD_RD	R/W	
	others	-			

Table 68. FMC Aurora Port 0 Component Registers

FMC Aurora Port 0 Test Control (MR_FMC_RIO0_TEST_CTRL, Base+0x00)
These are controls and status for Aurora port 0.

Bits	Field	R/W	Default	Description	Modules
0	tx_test_gen_en	R/W	0 = off	Enable test generator for transmission.	
1	rx_test_chk_en	R/W	0 = off	Enable test generator for receive.	
15:2	-				
31:16	test_errors	R		Test error count	

Table 69. FMC Aurora 0 Test Control Register

FMC Aurora Port 0 Control/Status (MR_FMC_RIO0_CTRL_STAT, Base+0x01)
These are controls and status for Aurora port 0.

Bits	Field	R/W	Default	Description	Modules
0	Gtxreset_n	R/W	0 = reset	MGT reset, active low.	
1	power_down	R/W	1 = on	MGT power down. (Turned off by default)	
2	run	R/W	0 = off	Aurora interface run.	
5:3	loopback	R/W	000	000 = Disable loopback 001 = Parallel 010 = Serial	
6	error_clr	R/W	0 = off	Clear port error.	
7	tx_channel_en	R/W	0 = off	Enable transmit channel.	
8	rx_channel_en	R/W	0 = off	Enable receive channel.	
22:9	-				
23	hard_error	R		Hard error. Link lost due to serious disruption.	
24	soft_error	R		Soft error such as bit error.	

25	frame_error	R		Frame error from Aurora.	
29:26	lane_up	R		Number of lane the Aurora port is using.	
30	channel_up	R		The Aurora channel is active.	
31	pll_locked	R		PLL for MGT is locked.	

Table 70. FMC Aurora 0 Control/Status Register

FMC Aurora Port 0 Sub-channel Write Port (MR_FMC_RIO0_CMD_WR, Base+0x02)
This is the sub-channel write port for Aurora 0.

Bits	Field	R/W	Default	Description	Modules
23:0	usr_cmd_wr_data	R/W	0	Command write data.	
29:24	usr_cmd_wr_addr	R/W	0	Command write address.	
30	usr_cmd_wr_rdn	R/W	0=write	Command read/write control. 0= write, 1 = read.	
31	cmd_ch_rdy	R		Command sub-channel is ready.	

Table 71. FMC Aurora 0 Sub-channel Write Register

FMC Aurora Port 0 Sub-channel Read Port (MR_FMC_RIO0_CMD_RD, Base+0x03)
This is the sub-channel read port for Aurora 0.

Bits	Field	R/W	Default	Description	Modules
23:0	usr_cmd_rd_data	R		Command read data.	
29:24	usr_cmd_rd_addr	R		Command read address.	
30	-				
31	usr_cmd_rd_vld	R		Command sub-channel is read data is valid.	

Table 72. FMC Aurora 0 Sub-channel Read Register

FMC Aurora 1 Registers (WB Device 17)

These are the registers for Aurora port 1.

B Base	WB Address	Register	Simulation Define	R/W	Description
0x1100	0x00		MR_FMC_RIO1_TEST_CTRL	R/W	
	0x01		MR_FMC_RIO1_CTRL_STAT	R/W	
	0x02		MR_FMC_RIO1_CMD_WR	R/W	
	0x03		MR_FMC_RIO1_CMD_RD	R/W	
	others	-			

Table 73. FMC Aurora Port 1 Component Registers

FMC Aurora Port 1 Test Control (MR_FMC_RIO1_TEST_CTRL, Base+0x00)
These are controls and status for Aurora port 0.

Bits	Field	R/W	Default	Description	Modules
0	tx_test_gen_en	R/W	0 = off	Enable test generator for transmission.	
1	rx_test_chk_en	R/W	0 = off	Enable test generator for receive.	
15:2	-				
31:16	test_errors	R		Test error count	

Table 74. FMC Aurora 1 Test Control Register

FMC Aurora Port 1 Control/Status (MR_FMC_RIO1_CTRL_STAT, Base+0x01)
These are controls and status for Aurora port 0.

Bits	Field	R/W	Default	Description	Modules
0	Gtxreset_n	R/W	0 = reset	MGT reset, active low.	
1	power_down	R/W	1 = on	MGT power down. (Turned off by default)	
2	run	R/W	0 = off	Aurora interface run.	
5:3	loopback	R/W	000	000 = Disable loopback 001 = Parallel 010 = Serial	
6	error_clr	R/W	0 = off	Clear port error.	
7	tx_channel_en	R/W	0 = off	Enable transmit channel.	
8	rx_channel_en	R/W	0 = off	Enable receive channel.	
22:9	-				
23	hard_error	R		Hard error. Link lost due to serious disruption.	
24	soft_error	R		Soft error such as bit error.	

25	frame_error	R		Frame error from Aurora.	
29:26	lane_up	R		Number of lane the Aurora port is using.	
30	channel_up	R		The Aurora channel is active.	
31	pll_locked	R		PLL for MGT is locked.	

Table 75. FMC Aurora 1 Control/Status Register

FMC Aurora Port 1 Sub-channel Write Port (MR_FMC_RIO1_CMD_WR, Base+0x02)
This is the sub-channel write port for Aurora 0.

Bits	Field	R/W	Default	Description	Modules
23:0	usr_cmd_wr_data	R/W	0	Command write data.	
29:24	usr_cmd_wr_addr	R/W	0	Command write address.	
30	usr_cmd_wr_rdn	R/W	0=write	Command read/write control. 0= write, 1 = read.	
31	cmd_ch_rdy	R		Command sub-channel is ready.	

Table 76. FMC Aurora 1 Sub-channel Write Register

FMC Aurora Port 1 Sub-channel Read Port (MR_FMC_RIO1_CMD_RD, Base+0x03)
This is the sub-channel read port for Aurora 0.

Bits	Field	R/W	Default	Description	Modules
23:0	usr_cmd_rd_data	R		Command read data.	
29:24	usr_cmd_rd_addr	R		Command read address.	
30	-				
31	usr_cmd_rd_vld	R		Command sub-channel is read data is valid.	

Table 77. FMC Aurora 1 Sub-channel Read Register

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
03/30/16	1.1	Added Alerts registers to Peripheral Registers.

Table 78. Revision History

FMC Memory Map

FMC Registers

The memory map for the FMC components on the Wishbone Bus is shown below. Some components are common to all the FMC modules, whereas others are card specific.

WB Base Address	WB Component	Simulation Define	Description	Module
0xc00	FMC	MR_FMC_IF	FMC interface registers	VPX, PEX, SBC
0xd00	FMC	MR_FMC_LA_DIO	FMC LA DIO register	VPX, PEX, DIO, SBC
	FMC	MR_FMC_AFE_CMN /MR_FMC_CMN	FMC AFE common registers	ADC20, FMC500, FMC1000, FMCServo
0xe00	FMC	MR_FMC_HA_DIO	FMC HA DIO register	VPX, PEX, DIO, SBC
	FMC	MR_FMC_ADC	FMC ADC registers	ADC20, FMC500, FMC1000, FMCServo
0xf00	FMC	MR_FMC_HB_DIO	FMC HB DIO register	VPX, PEX, DIO, SBC
	FMC	MR_FMC_DAC	FMC DAC registers	DAC40, FMC500, FMC1000, FMCServo
0x1000	FMC	MR_FMC_RIO0	Aurora core 0 within FMC interface	VPX, PEX, SBC, FMC_SFP, FMC_QSFP
0x1100	FMC	MR_FMC_RIO1	Aurora core 1 within FMC interface	VPX, PEX, SBC, FMC_QSFP
0x1200	FMC	MR_FMC_RIO2	Aurora core 2 within FMC interface	PEX, SBC
0x1300	FMC2	MR_FMC2_IF	FMC2 interface registers	SBC
0x1400	FMC2	MR_FMC2_LA_DIO	FMC2 LA DIO register	SBC
	FMC2	MR_FMC2_CMN	FMC2 AFE common registers	FMC310, FMC500
0x1500	FMC2	MR_FMC2_ADC	FMC ADC registers	FMC310, FMC500

Table 79. FMC Memory Map

FMC status and configuration Registers (WB Device 12)

These are the registers for FMC interface.

WB Base	WB Address	Register	Simulation Define	R/W	Description
0xc00	0x00		MR_FMC_CTRL	R/W	FMC control and status
	0x01		MR_FMC_I2C_IF	R/W	FMC I2C interface
	0x02		MR_FMC_ID	R	FMC ID
	0x03		MR_FMC_CLK_CFG	R/W	FMC BIDIR clock control and status
	0x04		MR_FMC_CLK0_M2C_STS	R	FMC clock 0 m2c status
	0x05		MR_FMC_CLK1_M2C_STS	R	FMC clock 1 m2c status
	0x06		MR_FMC_CLK2_M2C_STS	R	FMC clock 2 m2c status
	0x07		MR_FMC_CLK3_M2C_STS	R	FMC clock 3 m2c status
	0x08		MR_FMC_CLK2_C2M_STS	R	FMC clock 2 c2m status
	0x09		MR_FMC_CLK3_C2M_STS	R	FMC clock 3 c2m status

Table 80. FMC status and configuration registers

FMC control register (MR_FMC_CTRL, Base+0x00)
These are FMC control and status bits.

Bits	Field	R/W	Default	Description	Modules
0	fmc_present_n	R	-	FMC present (active low)	VPX, PEX, SBC
1	fmc_vadj_en_n_o	W	1	Enable FMC VADJ (active low)	VPX, PEX, SBC
1	fmc_vadj_en_n_i	R	1	Actual status of FMC VADJ Enable pin (active low)	VPX, PEX, SBC
2	fmc_vadj_forced	R		FMC VADJ is forced by the hardware	PEX, SBC
3	fmc_vadj_pwr_gd	R		FMC VADJ power good status	PEX, SBC
6:4	fmc_vadj_lvl	R		Selected VADJ level 000 => 1.2V 001 => 1.35V 010 => 1.5V 011 => 1.8V 100 => 2.5V 111 - 101 => N/A (spare)	PEX, SBC
15:7	-				
16	fmc_pg_m2c_n	R	-	FMC power good M2C (active low)	VPX, PEX, SBC
17	fmc_pg_c2m_n	R/W	1	FMC power good C2M (active low)	VPX, PEX, SBC
31:18	-				

Table 81. FMC control register

FMC VADJ setting procedure for PEX-COP boards:

1- Read fmc_present_n register. If set (= 0), skip to line 4.

2- Read fmc_vadj_lvl and compare it with the required VADJ value read from the FMC module. If the two values match, skip to line 4.

3- Set fmc_vadj_en_n_o to '1' and report a mismatch between the carrier VADJ and the required FMC VADJ value and exit.

4- Set fmc_vadj_en_n_o to '0' to turn on the VADJ power supply, wait for 100ms for the actual power supply enable pin to assert, and read the fmc_vadj_en_n_i to make sure its not forced off by external devices. If forced off, report it, set fmc_vadj_en_n_o to '1', and exit.

5- Poll fmc_vadj_pwr_gd register till it sets and report a VADJ power good status message. Timeout if it doesn't set within 2 seconds, set fmc_vadj_en_n_o to '1', and report a faulty VADJ power supply.

FMC I2C interface (MR_FMC_I2C_IF, Base+0x01)
This is the I2C interface to the FMC.

Bits	Field	R/W	Default	Description	Modules
0	fmc_sdo	R/W	-	FMC I2C data out	VPX, PEX, SBC
1	fmc_scl	R/W	-	FMC I2C clock	VPX, PEX, SBC
2	fmc_sdi	R	-	FMC I2C data in	VPX, PEX, SBC
3	-	R		FMC I2C clock readback	VPX, PEX, SBC
31:4	-				

Table 82. FMC I2C interface

FMC ID (MR_FMC_ID, Base+0x2)
This register has the FMC ID.

Bits	Field	R/W	Default	Description	Modules
31:0	fmc_id	R		FMC ID	VPX, PEX, SBC

Table 83. FMC ID

FMC BIDIR Clock control and status (MR_FMC_CLK_CFG, Base+0x3)
This register has the FMC BIDIR clock control and status bits.

Bits	Field	R/W	Default	Description	Modules
0	fmc_clk_dir	R		FMC BIDIR clock direction (0=M2C, 1=C2M)	PEX, SBC
1	fmc_clk_2_3_en	R/W	0	FMC clock 2 & 3 driver enable	PEX
2	fmc_clk2_sel	R/W	0	FMC clock 2 source select 0=DIFF, 1=SE	PEX
3	fmc_clk3_sel	R/W	0	FMC clock 3 source select 0=DIFF, 1=SE	PEX
31:4	-				

Table 84. FMC BIDIR Clock Register

FMC Clock0 m2c status (MR_FMC_CLK0_M2C_STS, Base+0x4)

Bits	Field	R/W	Default	Description	Modules
9:0	fmc_clk0_m2c_freq	R		FMC clock0 M2C frequency in MHz	PEX, SBC
31:10	-				

Table 85. FMC Clock0 m2c Register

FMC Clock1 m2c status (MR_FMC_CLK1_M2C_STS, Base+0x5)

Bits	Field	R/W	Default	Description	Modules
9:0	fmc_clk1_m2c_freq	R		FMC clock1 M2C frequency in MHz	PEX, SBC
31:10	-				

Table 86. FMC Clock1 m2c Register

FMC Clock2 m2c status (MR_FMC_CLK2_M2C_STS, Base+0x6)

Bits	Field	R/W	Default	Description	Modules
9:0	fmc_clk2_m2c_freq	R		FMC clock2 M2C frequency in MHz	PEX, SBC
31:10	-				

Table 87. FMC Clock2 m2c Register

FMC Clock3 m2c status (MR_FMC_CLK3_M2C_STS, Base+0x7)

Bits	Field	R/W	Default	Description	Modules
9:0	fmc_clk3_m2c_freq	R		FMC clock3 M2C frequency in MHz	PEX, SBC
31:10	-				

Table 88. FMC Clock3 m2c Register

FMC Clock2 c2m status (MR_FMC_CLK2_C2M_STS, Base+0x8)

Bits	Field	R/W	Default	Description	Modules
9:0	fmc_clk2_c2m_freq	R		FMC clock2 C2M frequency in MHz	PEX
31:10	-				

Table 89. FMC Clock2 c2m Register

FMC Clock3 c2m status (MR_FMC_CLK3_C2M_STS, Base+0x9)

Bits	Field	R/W	Default	Description	Modules
9:0	fmc_clk3_c2m_freq	R		FMC clock3 C2M frequency in MHz	PEX
31:10	-				

Table 90. FMC Clock3 c2m Register

FMC DIO Registers

The single ended Digital I/O registers are concatenated as follows: positive on even bits, negative on odd bits, ie. fmc_la_p[33:0] are on even bits, fmc_la_n[33:0] on odd bits.

For example:

Bits[31:6]	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
…	fmc_la_n[2]	fmc_la_p[2]	fmc_la_n[1]	fmc_la_p[1]	fmc_la_n[0]	fmc_la_p[0]

FMC LA DIO register (WB Device 13)

These are the registers for FMC LA DIO bus.

WB Base	WB Address	Register	Simulation Define	R/W	Description
0xd00	0x00		MR_FMC_LA_DOUT_L	R/W	Lower 32 bit word of the FMC LA DIO bus. The entire DIO bus is updated when writing to this register.
	0x01		MR_FMC_LA_DOUT_H	R/W	Middle 32 bit word of the FMC LA DIO bus
	0x02		MR_FMC_LA_DOUT_V	R/W	Upper 2 bit word of the FMC LA DIO bus
	0x04		MR_FMC_LA_OE_L	R/W	Output enable of the lower 32 bit word of the FMC LA DIO bus
	0x05		MR_FMC_LA_OE_H	R/W	Output enable of the middle 32 bit word of the FMC LA DIO bus
	0x06		MR_FMC_LA_OE_V	R/W	Output enable of the upper 2 bit word of the FMC LA DIO bus

Table 91. FMC LA DIO register

FMC HA DIO register (WB Device 14)

These are the registers for FMC HA DIO bus.

WB Base	WB Address	Register	Simulation Define	R/W	Description
0xe00	0x00		MR_FMC_HA_DOUT_L	R/W	Low 32 bit word FMC HA DIO bus. The entire DIO bus is updated when writing to this register.
	0x01		MR_FMC_HA_DOUT_H	R/W	High 32 bit word FMC HA DIO bus
	0x02		MR_FMC_HA_OE_L	R/W	Output enable of the low 32-bit word FMC HA DIO bus
	0x03		MR_FMC_HA_OE_H	R/W	Output enable of the high 16-bit word FMC HA DIO bus

Table 92. FMC HA DIO register

FMC HB DIO register (WB Device 15)

These are the registers for FMC HB DIO bus.

WB Base	WB Address	Register	Simulation Define	R/W	Description
0xf00	0x00		MR_FMC_HB_DOUT_L	R/W	Low 32 bit word FMC HB DIO bus. The entire DIO bus is updated when writing to this register.
	0x01		MR_FMC_HB_DOUT_H	R/W	High 32 bit word FMC HB DIO bus
	0x02		MR_FMC_HB_OE_L	R/W	Output enable of the low 32-bit word FMC HB DIO bus
	0x03		MR_FMC_HB_OE_H	R/W	Output enable of the high 12-bit word FMC HB DIO bus

Table 93. FMC HB DIO register

FMC AFE

FMC AFE Common Registers (WB Device 13)
These are the FMC common Analog Front End control and configuration registers.

WB Base	WB Address	Register	Simulation Define	R/W	Description	Modules
Clock Registers						
0xD00	0x0		MR_FMC_PLL_CTRL	R/W		ADC20, FMC250, FMC110, FMC310, FMC500, FMC1000
	0x1		MR_FMC_PLL_SPI	R/W		ADC20, FMC110, FMC310, FMC500, FMC1000
			MR_FMC_PLL_UW			FMC250
	0x2		MR_FMC_VCXO	R/W		FMC500
	0x3		MR_FMC_CLK_CTRL	R/W		
	0x4 - 0x5					
AFE Common Registers						
	0x6		MR_FMC_TEST_CTRL	R/W		ADC20, FMC250, FMC310, FMC500, FMC1000
	0x7		MR_FMC_SW_TRIG	R/W		ADC20, FMC250, FMC310, FMC500, FMC1000, FMCServo
	0x8		MR_FMC_EXT_SYNC_CFG	R/W		ADC20, FMC250, FMC310, FMC500, FMCServo
			MR_FMC_EXT_TRIG_SEL			
	0x9		MR_FMC_EXT_CLK_CFG	R/W		FMC1000
	0xA		MR_FMC_EXT_SYNC_CFG	R/W		FMC1000
	0xB-0xF					

Table 94. FMC AFE Common Registers

PLL Control (MR_FMC_PLL_CTRL, Base+0x0)
This register has the PLL controls and status.

Bits	Field	R/W	Default	Description	Modules
0	pll_pwr_down_n	R/W	0	PLL power down. PLL is from 0.5 to 2W when operating. Allow 5 min warm-up time when device is powered up for best performance. 0 = power off, 1 = power on	ADC20
1	pll_reset	R/W	0	PLL reset	FMC310, FMC500, FMC1000
2	pll_mode	R/W	0	PLL configuration mode. '0' = SPI configuration '1' = load from default registers	
3	fpga_pll_clkin_stoppe	R		FPGA PLL input clock stopped	FMC110

	d				
4	pll_lock	R		PLL lock indicator, '1' = locked.	ADC20
4	fpga_pll_lock	R		FPGA PLL locked	FMC110
5	fpga_pll_rst	R/W	0	FPGA PLL reset (active high)	FMC110
5	pll_clk_sel(0)	R/W	'0'	0 = PRI, 1 = SEC	
	pll_ref_sel	R/W	'0'	0 = output of pll_clk_sel mux 1 = 10 MHz oscillator	FMCServo
6	pll_clk_sel(1)	R/W			
	pll_clk_sel	R/W	'0'	0 = clk2_bidir 1 = ext_clk	FMCServo
7	pll_sync	R/W	0	pll_sync	ADC20, FMC250, FMC500
7	pll_sync1	R/W		pll_sync	FMCServo
8	pll_status_ho	R		PLL programmable status pin	FMC250
8	pll_sync2	R/W		pll_sync	FMCServo
9	pll_status_ld	R		PLL programmable status pin	FMC250
10	pll_status_clkin0	R		PLL programmable status pin	FMC250
11	pll_status_clkin1	R		PLL programmable status pin	FMC250
12	pll_gpo	R		PLL general purpose output	FMC310, FMC500, FMC1000
13	pll_status_ld1	R		PLL programmable status pin	FMC310, FMC500, FMC1000
13	pll_status	R/W		PLL status pin	FMCServo
14	pll_status_ld2	R		PLL programmable status pin	FMC310, FMC500, FMC1000
15	pll_clkin_sel0_o	W		PLL clkin selector (bit 0)	FMC310, FMC1000
15	pll_clkin_sel0_i	R		PLL programmable status pin	FMC310
15	pll_clkin_sel0	R/W		PLL programmable status pin	FMC500
16	pll_clkin_sel1_o	W		PLL clkin selector (bit 1)	FMC310, FMC1000
16	pll_clkin_sel1_i	R		PLL programmable status pin	FMC310
16	pll_clkin_sel1	R/W		PLL programmable status pin	FMC500
17	pll_clkin_sel0_dir	R/W		PLL clkin_sel0 direction (1=out, 0=in)	FMC310, FMC1000
18	pll_clkin_sel1_dir	R/W		PLL clkin_sel1 direction (1=out, 0=in)	FMC310, FMC1000
29:19	-			unused	
30	pll_uw/spi_rdy	R		PLL uw/spi ready	ADC20, FMC250, FMC110, FMC310, FMC500, FMC1000, FMCServo
31	pll_uw/spi_rdata_valid	R		PLL uw/spi data valid	ADC20, FMC250, FMC110, FMC310, FMC500, FMC1000, FMCServo

Table 95. FMC PLL Control Register

PLL SPI Data (MR_FMC_PLL_SPI, Base+0x1)
This is the interface to the PLL SPI port.

Bits	Field	R/W	Default	Description	Modules
3:0	pll_spi_addr	R/W	0x0	PLL SPI address.	ADC20
31:4	pll_spi_wdata(W) pll_spi_rdata(R)	R/W	0x000 0000	PLL SPI data. The pll_spi_rdy bit must be checked before writing to this register. Reads from this register return the SPI read data from a read request.	ADC20

Table 96. FMC PLL SPI Data Register

PLL uWire Data (MR_FMC_PLL_UW, Base+0x1)
This is the interface to the PLL SPI port for FMC SPI.

Bits	Field	R/W	Default	Description	Modules
4:0	pll_uw_addr	R/W	0x0	PLL uWire address	FMC250
31:5	pll_uw_wdata(W) pll_uw_rdata(R)	R/W	0x000 0000	PLL SPI data. The pll_spi_rdy bit must be checked before writing to this register. Reads from this register return the SPI read data from a read request.	FMC250

Table 97. FMC PLL SPI Data Register

PLL SPI Data (MR_FMC_PLL_SPI, Base+0x1)
This is the interface to the PLL SPI port.

Bits	Field	R/W	Default	Description	Modules
9:0	pll_spi_addr	R/W	0x0	PLL SPI address.	FMC110, FMCServo
10	pll_sel	R/W		PLL select	FMCServo
11	unused				
12	pll_spi_rd_wrn	R/W		SPI read/write select 1=read/0=write	FMC110, FMCServo
15:13	unused				
23:16	pll_spi_wdata	R/W		SPI write data	FMC110, FMCServo
31:24	pll_spi_rdata	R		SPI read data	FMC110, FMCServo

Table 98. FMC PLL uWire/SPI Data Register

PLL SPI Data (MR_FMC_PLL_SPI, Base+0x1)
This is the interface to the PLL SPI port.

Bits	Field	R/W	Default	Description	Modules
12:0	pll_spi_addr	R/W	0x0	PLL SPI address.	FMC310, FMC500, FMC1000
14:13	unused				
15	pll_spi_rd_wrn	R/W		SPI read/write select 1=read/0=write	FMC310, FMC500, FMC1000
23:16	pll_spi_wdata	R/W		SPI write data	FMC310, FMC500,

					FMC1000
31:24	pll_spi_rdata	R		SPI read data	FMC310, FMC500, FMC1000

Table 99. FMC PLL SPI Data Register

VCXO Control (MR_FMC_VCXO, Base+0x2)
This is the interface to the VCXO controls.

Bits	Field	R/W	Default	Description	Modules
3:0					
4	vcxo_pwr_en	R/W		VCXO power enable	FMC250, FMC500
7:5					
8	vcxo_pwr_gd	R		VCXO power good	FMC500
31:9					

Table 100. FMC VCXO control Register

CPLD Status (MR_FMC_CPLD_STAT, Base+0x4)
This register has the CPLD SPI status.

Bits	Field	R/W	Default	Description	Modules
29:0	-			unused	
30	cpld_spi_rdy	R		CPLD spi ready	FMC110
31	cpld_spi_rdata_valid	R		CPLD spi data valid	FMC110

Table 101. FMC CPLD Status Register

CPLD SPI Data (MR_FMC_CPLD_SPI, Base+0x5)
This is the interface to the CPLD SPI port.

Bits	Field	R/W	Default	Description	Modules
1:0	cpld_spi_addr	R/W	0x0	CPLD SPI address.	FMC110
11:2	unused				
12	cpld_spi_rd_wrn	R/W		CPLD SPI read/write select 1=read/0=write	FMC110
15:13	unused				
23:16	cpld_spi_wdata	R/W		CPLD SPI write data	FMC110
31:24	cpld_spi_rdata	R	0x000 0000	CPLD SPI read data	FMC110

Table 102. FMC CPLD SPI Data Register

FMC Test Controls (MR_FMC_TEST_CTRL, Base+0x6)
This register sets the test mode for different components in the FMC.

Bits	Field	R/W	Default	Description	Modules
0	adc_test_en	R/W	0	Enable ADC test generator. 0 = off, 1 = on	ADC20, FMC250,

					FMC110, FMC310, FMC500, FMC1000, FMCServo
3:1	-				
4	adc_test_mode	R/W	0	ADC test mode: 0 = unpaced sawtooth, 1 = paced sawtooth	ADC20, FMC250, FMC110, FMC310, FMC500, FMC1000, FMCServo
15:5	-				
16	dac_test_en	R/W	0	Enable DAC test generator. 0 = off, 1 = on	FMC110, FMC250, FMC500, FMC1000, FMCServo
19:17	-				
22:20	dac_test_mode	R/W	0	DAC test mode: 0 = ramp, 1 = sine, 2 = dac test pattern, 3 = zeros, 4 = max positive, 5 = max negative, 6 = alternating 1's and 0's, 7 = alternating two 1's and two 0's	FMC110, FMC250, FMC500, FMC1000, FMCServo
31:23	-				

Table 103. FMC Test Controls Register

FMC Software Trigger Controls (MR_FMC_SW_TRIG, Base+0x7)
This register enables software triggering for different components in the FMC.

Bits	Field	R/W	Default	Description	Modules
0	adc_sw_trig	R/W	0	ADC software trigger. 0 = off, 1 = on	ADC20, FMC250, FMC110, FMC310, FMC500, FMC1000, FMCServo
15:1	-				
16	dac_sw_trig	R/W	0	DAC software trigger. 0 = off, 1 = on	FMC110, FMC250, FMC500, FMC1000, FMCServo
31:17	-				

Table 104. FMC Software Trigger Controls Register

FMC External Sync Select (MR_FMC_EXT_SYNC_CFG, Base+0x8)

Bits	Field	R/W	Default	Description	Modules
0	ext_sync_sel	R/W	0	External sync select (0=front panel, 1=FMC)	ADC20, FMC250, FMC310, FMC500, FMCServo
31:1	unused				

Table 105. FMC External Sync Select Register

FMC External Trigger Select (MR_FMC_EXT_TRIG_SEL, Base+0x8)

Bits	Field	R/W	Default	Description	Modules

0	ext_trig_sel	R/W	0	External trigegr select (0=clk3_bidir, 1=front panel)	FMCServo
31:1	unused				

Table 106. FMC External Sync Select Register

FMC External Clock Configuration (MR_FMC_EXT_CLK_CFG, Base+0x9)

Bits	Field	R/W	Default	Description	Modules
11:0	clk_mux_cfg_data	R/W	0	Clock mux configuration data	FMC1000
30:12	unused				
31	clk_mux_cfg_rdy	R		Clock mux configuration ready	FMC1000

Table 107. FMC External Clock Configuration Register

FMC External Sync Configuration (MR_FMC_EXT_SYNC_CFG, Base+0xA)

Bits	Field	R/W	Default	Description	Modules
11:0	sy_mux_cfg_data	R/W	0	sy_mux configuration data	FMC1000
30:12	unused				
31	sy_mux_cfg_rdy	R		sy_mux configuration ready	FMC1000

Table 108. FMC External Sync Configuration Register

FMC ADC Registers (WB Device 14)
These are the registers for the FMC ADC control and configuration.

WB Base	WB Address	Register	Simulation Define	R/W	Description	Modules
ADC Common Registers						
0xE00	0x0		MR_FMC_ADC_EN1	R/W		ADC20, FMC250, FMC110, FMC310, FMC500, FMC1000, FMCServo
	0x1		MR_FMC_ADC_EN2	R/W		
	0x2		MR_FMC_ADC_PDN1	R/W		FMC310, FMC500, FMC1000
	0x3		MR_FMC_ADC_PDN2	R/W		
	0x4		MR_FMC_ADC_TRGR	R/W		ADC20, FMC250, FMC110, FMC310, FMC500, FMC1000, FMCServo
	0x5		MR_FMC_ADC_DECI	R/W		ADC20, FMC250, FMC110, FMC310, FMC500, FMC1000, FMCServo
	0x6		MR_FMC_ADC_PRI_TRGR	R/W		ADC20, FMC250, FMC110, FMC310, FMC500, FMC1000, FMCServo
	0x7		MR_FMC_ADC_PRI	R/W		ADC20, FMC250, FMC110, FMC310, FMC500, FMC1000, FMCServo
	0x8		MR_FMC_ADC_PRI_PARAM	R/W		ADC20, FMC250, FMC110, FMC310, FMC500, FMC1000, FMCServo
	0x9		MR_FMC_ADC_PRI_WIDTH	R/W		ADC20, FMC250, FMC110, FMC310, FMC500, FMC1000, FMCServo
	0xA - 0xF	-				
ADC Specific Registers						
	0x10		MR_FMC_ADC_SPI_EN	R/W		ADC20, FMC110, FMC310
			MR_FMC_ADC0_SPI_CTRL	R/W		FMC250
	0x11		MR_FMC_ADC_SPI_CTRL	R/W		ADC20, FMC110, FMC310, FMC500, FMC1000
			MR_FMC_ADC0_SPI_STAT	R/W		FMC250
	0x12		MR_FMC_ADC_SPI_STAT	R/W		ADC20, FMC110, FMC310, FMC500, FMC1000
			MR_FMC_ADC1_SPI_CTRL	R/W		FMC250
	0x13		MR_FMC_ADC_CAL	R/W		ADC20
			MR_FMC_ADC1_SPI_STAT	R/W		FMC250
	0x14		MR_FMC_VGA	R/W		ADC20
			MR_FMC_ADC_PHY_CAL	R/W		FMC250, FMC500, FMC1000
	0x15		MR_FMC_ADC_CAL_STS	R/W		FMC500
	0x16	-				
	0x17	-				
	0x18		MR_FMC_ADC_PHY_CAL	R/W		FMC110

WB Base	WB Address	Register	Simulation Define	R/W	Description	Modules
	0x19		MR_FMC_ADC_AMP_CFG	R/W		FMC110, FMC1000
	0x1A		MR_FMC_ADC_CTRL	R/W		FMCServo
	0x1B		MR_FMC_ADC_FIFO_CTRL	R		FMCServo
	0x1C		MR_FMC_ADC_FIFO_THRS	R/W		FMCServo
	0x1D		MR_FMC_ADC_FIFO_DATA	R		FMCServo
	0x1E		MR_FMC_ADC_GAIN_CTRL	R/W		FMCServo
	0x1F	-				
ADC VITA Packet Configuration and Timestamping						
	0x20		MR_FMC_ADC_TS_LD	R/W		ADC20, FMC250, FMC110, FMC310, FMC500, FMC1000
	0x21		MR_FMC_ADC_TS_CTRL	R/W		ADC20, FMC250, FMC110, FMC310, FMC500, FMC1000
	0x22		MR_FMC_ADC_VITA_CTRL	R/W		ADC20, FMC250, FMC110, FMC310, FMC500
	0x23 - 0x2F	-				
ADC VITA Frame Sizes and Stream IDs						
	0x30 - 0x34		MR_FMC_ADC_VFRAME	R/W		ADC20, FMC250, FMC110, FMC310, FMC500, FMC1000
	0x35 - 0x3F	-				
	0x40 - 0x44		MR_FMC_ADC_SID	R/W		ADC20, FMC250, FMC110, FMC310, FMC500, FMC1000
	0x45 - 0x4F	-				
ADC Gain Registers						
	0x50 - 0x63		MR_FMC_ADC_GAIN	R/W		ADC20, FMC250, FMC110, FMC500, FMC1000, FMCServo
	0x64 - 0x8F	-				
ADC Offset Registers						
	0x90 - 0xA3		MR_FMC_ADC_OFST	R/W		ADC20, FMC250, FMC110, FMC500, FMC1000, FMCServo
	0xA4 - 0xCF					

Table 109. FMC ADC Component Registers

ADC Low Channel Enables (MR_FMC_ADC_EN1, Base+0x0)
This is the lower A/D 32 channel enable register.

Bits	Field	R/W	Default	Description	Modules
31:0	adc_ch_en	R/W	0x0000	A/D channel enables.	ADC20 = 20, FMC250=2, FMC110 = 2, FMC310 = 4, FMC500 = 2, FMC1000 = 2, FMCServo = 8

Table 110. FMC Low ADC Channel Enables Register

ADC High Channel Enables (MR_FMC_ADC_EN2, Base+0x1)
This is the higher A/D 32 channel enable register.

Bits	Field	R/W	Default	Description	Modules
31:0	adc_ch_en	R/W	0x0000	A/D channel enables.	

Table 111. FMC High ADC Channel Enables Register

ADC Low Power Enables (MR_FMC_ADC_PDN1, Base+0x2)
This is the lower A/D 32 power enable register.

Bits	Field	R/W	Default	Description	Modules
31:0	adc_pwr_en	R/W	0x0000	A/D power enables.	FMC250=2, FMC310 = 4, FMC500 = 2, FMC1000 = 2

Table 112. FMC Low ADC Power Enables Register

ADC High Power Enables (MR_FMC_ADC_PDN2, Base+0x3)
This is the higher A/D 32 power enable register.

Bits	Field	R/W	Default	Description	Modules
31:0	adc_pwr_en	R/W	0x0000	A/D power enables.	

Table 113. FMC High ADC Power Enables Register

FMC ADC Trigger Controls (MR_FMC_ADC_TRGR, Base+0x4)
This register configures the A/D trigger modes.

Bits	Field	R/W	Default	Description	Modules
23:0	adc_window_size	R/W	0	ADC trigger window size. This is the number of points, after decimation, that make up one data window.	All
28:24					
31:29	adc_trigger_mode	R/W	“000”	ADC trigger modes. Bit29 = rising edge (1) or level (0) Bit30 = framed (1) or unframed (0) Bit31 = external (1) or software (0)	All

Table 114. FMC ADC Trigger Controls Register

ADC Decimation (MR_FMC_ADC_DECI, Base+0x5)
This register specifies the decimation ratio for the A/D triggering.

Bits	Field	R/W	Default	Description	Modules
11:0	adc_decimation	R/W	0	Decimation count for A/D samples. Decimation keeps 1 point every N specified by this field.	All
31:12	-				

Table 115. FMC ADC Decimation Ratio Register

ADC PRI trigger enable register (MR_FMC_ADC_PRI_TRGR, Base+0x6)
This register is the ADC PRI trigger control register

Bits	Field	R/W	Default	Description	Modules
0	en_pri_trig	R/W		Enable PRI trigger mode	All
1	stop_pri	R/W		Stop PRI triggering	All
2	en_num_pri	R/W		enable finite number of PRI frames	All
3	retrig_num_pri	R/W		re-arm after number of PRI frames	All
7:4	-				
8	pri_busy	R		PRI mode is running when this bit is 1	All
15:9	-				
31:16	num_pri	R/W		number of PRI frames	All

Table 116. FMC ADC PRI trigger enable Register

ADC PRI interval configuration register (MR_FMC_ADC_PRI, Base+0x7)
This register is used to set the pulse repetition interval

Bits	Field	R/W	Default	Description	Modules
31:0	pri	R/W		Pulse repetition interval	All

Table 117. FMC ADC PRI interval configuration Register

ADC PRI trigger parameters register (MR_FMC_ADC_PRI_PARAM, Base+0x8)
This register configures the ADC PRI trigger parameters

Bits	Field	R/W	Default	Description	Modules
23:0	trig_cycle_delay	R/W		Delay between trigger and sof	All
31:24	-				

Table 118. FMC ADC PRI trigger parameters Register

ADC PRI capture window configuration register (MR_FMC_ADC_PRI_WIDTH, Base+0x9)
This register configures the ADC PRI trigger parameters FIFO. Writing to this register generates a write strobe to the ADC PRI parameters FIFO which causes the width and cycle delay parameters to be written to that FIFO.

Bits	Field	R/W	Default	Description	Modules
23:0	trig_width	R/W		trigger width	All
31:24	-				

Table 119. FMC ADC PRI capture window configuration Register

FMC ADC device SPI enable register (MR_FMC_ADC_SPI_EN, Base+0x10)
This register is used to enable the ADC device that receives the SPI command.

Bits	Field	R/W	Default	Description	Modules
31:0	ad_spi_dev_en	R/W		ADC device SPI enable (1 bit per device). When a certain device's enable bit is set, then that device will receive a SPI command when the SPI registers are accessed.	ADC20 = 5, FMC110 = 2, FMC310 = 2

Table 120. FMC ADC device SPI enable Register

FMC ADC0 SPI control register (MR_FMC_ADC0_SPI_CTRL, Base+0x10)
This is the FMC250 ADC0 SPI control register.

Bits	Field	R/W	Default	Description	Modules
7:0	ad0_spi_wdata	R/W		ADC0 SPI write data	FMC250
15:8	-			Unused	
23:16	ad0_spi_addr	R/W		ADC0 SPI address	FMC250
27:24	-			Unused	
28	ad0_spi_rd_wrn	R/W		ADC0 SPI read/write enable	FMC250
31:29	-			Unused	

Table 121. FMC250 ADC0 device SPI control register

FMC ADC SPI Control (MR_FMC_ADC_SPI_CTRL, Base+0x11)
This is the A/D devices SPI port writes.

Bits	Field	R/W	Default	Description	Modules
7:0	adc_spi_wdata	R/W	0x00	ADC SPI write data	ADC20, FMC110, FMC310, FMC500, FMC1000
15:8	-				
27:16	adc_spi_addr	R/W	0	ADC SPI address	ADC20, FMC110, FMC310
30:16	adc_spi_addr	R/W	0	ADC SPI address	FMC500, FMC1000
28	adc_spi_rd_wrn	R/W	0	ADC SPI read/write bit. 0= write, 1 = read	ADC20, FMC110, FMC310
31	adc_spi_rd_wrn	R/W	0	ADC SPI read/write bit. 0= write, 1 = read	FMC500, FMC1000
31:29	-				

Table 122. FMC ADC SPI Control Register

FMC ADC0 SPI status register (MR_FMC_ADC0_SPI_STAT, Base+0x11)
This is the FMC250 ADC0 SPI status register.

Bits	Field	R/W	Default	Description	Modules
7:0	ad0_spi_rdata	R	0x00	ADC0 SPI read data	FMC250
29:8	-			Unused	
30	ad0_spi_rdy	R		ADC0 SPI ready	FMC250
31	ad0_spi_rdata_valid	R		ADC0 SPI data valid	FMC250

Table 123. FMC250 ADC SPI Stauts Register

FMC ADC SPI Status (MR_FMC_ADC_SPI_STAT, Base+0x12)
This is the A/D devices SPI port reads.

Bits	Field	R/W	Default	Description	Modules

7:0	adc_spi_rdata	R		ADC SPI read data	ADC20, FMC110, FMC310, FMC500, FMC1000
29:8	-				
30	adc_spi_rdy	R		ADC SPI is ready for use.	ADC20, FMC110, FMC310, FMC500, FMC1000
31	adc_spi_rdata_valid	R		ADC SPI read data is valid.	ADC20, FMC110, FMC310, FMC500, FMC1000

Table 124. FMC ADC SPI Status Register

FMC ADC0 SPI control register (MR_FMC_ADC1_SPI_CTRL, Base+0x12)
This FMC250 ADC1 SPI control register.

Bits	Field	R/W	Default	Description	Modules
7:0	ad1_spi_wdata	R/W		ADC1 SPI write data	FMC250
15:8	-			Unused	
23:16	ad1_spi_addr	R/W		ADC1 SPI address	FMC250
27:24	-			Unused	
28	ad1_spi_rd_wrn	R/W		ADC1 SPI read/write enable	FMC250
31:29	-			Unused	

Table 125. FMC250 ADC device SPI control register

FMC ADC calibration Control and Status (MR_FMC_ADC_CAL, Base+0x13)
This is the status register for the ADC calibration.

Bits	Field	R/W	Default	Description	Modules
0	cal_start	R/W		Start ADC calibration (should be toggled after programming the ADC registers)	ADC20
15:1					
20:16	cal_done	R		ADC device calibration done	ADC20
31:21	-				

Table 126. FMC ADC calibration control and status register

FMC ADC1 SPI status register (MR_FMC_ADC1_SPI_STAT, Base+0x13)
This is the FMC250 ADC1 SPI status register.

Bits	Field	R/W	Default	Description	Modules
7:0	ad1_spi_rdata	R	0x00	ADC1 SPI read data	FMC250
29:8	-			Unused	
30	ad1_spi_rdy	R		ADC1 SPI ready	FMC250
31	ad1_spi_rdata_valid	R		ADC1 SPI data valid	FMC250

Table 127. FMC250 ADC SPI Stauts Register

FMC VGA Controls (MR_FMC_VGA, Base+0x14)
This is the interface to the VGA. The VGA I2C port is a bit-banged interface through this register.

Bits	Field	R/W	Default	Description	Modules
0	vga_sdo	R/W	0	VGA I2C data output bit.	ADC20
1	vga_scl	R/W	0	VGA I2C clock output bit.	ADC20
2	vga_sdi	R	-	VGA I2C data input bit.	ADC20
3	vga_scl	R	-	VGA I2C port clock readback.	ADC20
7:4	-				
8	vga_ldac_n	R/W	0	Load VGA DAC latch enable (active low).	ADC20
31:9	-				

Table 128. FMC VGA Controls Register

FMC ADC PHY Calibration register (MR_FMC_ADC_PHY_CAL, Base+0x14)
This is the adc phy calibration register.

Bits	Field	R/W	Default	Description	Modules
1:0	adc_rst	R/W		ADC reset	FMC250
8	adc_pwr_gd	R		ADC power good	FMC500, FMC1000
13:12	adc_phy_cal_start	R/W		Start ADC PHY calibration (write 1 then 0)	FMC500
14	adc_lat_cal_start	R/W		Enable latency calibration	FMC500
15	adc_en_data_fmt	R/W		Enable data format module	FMC500
17:16	adc_cal_done	R		ADC calibration status	FMC250
17:16	adc_phy_cal_done	R		ADC PHY calibration status	FMC500
18	adc_lat_cal_done	R		ADC latency calibration done	FMC500
20	adc_clk_stopped	R		ADC clock stopped	FMC250
21	adc_clk_locked	R		ADC clock locked	FMC250
31:18	-			Unused	

Table 129. FMC ADC PHY Calibration register

FMC ADC PHY Calibration status register (MR_FMC_ADC_CAL_STS, Base+0x15)
This is the adc phy calibration status register.

Bits	Field	R/W	Default	Description	Modules
14:0	adc_cal1_sts	R		ADC level1 calibration status	FMC500
30:16	adc_cal2_sts	R		ADC level2 calibration status	FMC500

Table 130. FMC ADC PHY Calibration status register

FMC ADC PHY Calibration register (MR_FMC_ADC_PHY_CAL, Base+0x18)
This is the FMC110 adc phy calibration register.

Bits	Field	R/W	Default	Description	Modules
0	adc_phy_init	R/W		Initialize ADC PHY of the selected ADC channel	FMC110
1:7	-			Unused	
8	sel_adc_ch	R/W		Select ADC channel to forward calibration control and status	FMC110
11:9	-			Unused	
12	skip_adc_phy_cal	R/W		Skip ADC PHY calibration	FMC110
13	-			Unused	
26:14	adc_eye_aligned	R		ADC data eye is aligned	FMC110
27	adc_prbs_locked	R		local PRBS is locked to ADC bit0	FMC110
28	adc_prbs_aligned	R		ADC PRBS data sequence is aligned	FMC110
29	adc_phy_rdy	R		ADC PHY is calibrated and ready	FMC110
30	adc_clka_stopped	R		ADC output clock stopped	FMC110
31	adc_clka_locked	R		ADC output clock locked	FMC110

Table 131. FMC110 ADC PHY Calibration register

FMC ADC Amplitude Configuration (MR_FMC_ADC_AMP_CFG, Base+0x19)

Bits	Field	R/W	Default	Description	Modules
1:0	adc_multx16_en	R/W	0	Multiply ADC output by 16	FMC110
0	adc_multx4_en	R/W	0	Multiply ADC output by 4	FMC1000
31:2					

Table 132. FMC ADC Amplitude Configuration Register

ADC Control (MR_FMC_ADC_CTRL, Base+0x1A)
ADC control register

Bits	Field	R/W	Default	Description	Modules
0	-				
1	adc_stby	R/W		ADC standby	FMCServo
2	adc_asleep	R/W		ADC sleep	FMCServo
31:3	-				

Table 133. FMC ADC Control Register

FMC Servo ADC Fifo Control (MR_FMC_ADC_FIFO_CTRL, Base+0x1B)

Bits	Field	R/W	Default	Description	Modules
10:0	adc_ofifo_data_count	R		ADC ofifo data count	FMCServo

27:11	-				
28	adc_ofifo_prog_empty	R		ADC ofifo almost empty	FMCServo
29	adc_ofifo_empty	R		ADC ofifo empty	FMCServo
30	adc_ofifo_prog_full	R		ADC ofifo almost full	FMCServo
31	adc_ofifo_full			ADC ofifo full	FMCServo

Table 134. FMC Servo ADC FIFO Control

FMC Servo ADC FIFO Threshold (MR_FMC_ADC_FIFO_THRS, Base+0x1C)

Bits	Field	R/W	Default	Description	Modules
9:0	adc_ofifo_empty_thresh	R/W		ADC ofifo almost empty threshold	FMCServo
15:10	-				
25:16	adc_ofifo_full_thresh	R/W		ADC ofifo almost full threshold	FMCServo
31:26	-				

Table 135. FMC Servo ADC FIFO Threshold

FMC Servo ADC FIFO Data (MR_FMC_ADC_FIFO_DATA, Base+0x1D)

Bits	Field	R/W	Default	Description	Modules
15:0	adc_data	R		ADC ofifo data	FMCServo
30:16	-				
31	Valid	R		ADC data valid when this bit is 1	FMCServo

Table 136. FMC Servo ADC FIFO Data

FMC Servo Gain Amp Control (MR_FMC_ADC_GAIN_CTRL, Base+0x1E)

Bits	Field	R/W	Default	Description	Modules
2:0	gain_amp_sel	R/W		Gain amp sel	FMCServo
3	gain_amp_wr	R/W		Gain amp wr	FMCServo
5:4	gain_amp_setting	R/W		Gain amp setting	FMCServo
31:6	-				

Table 137. FMC Servo Gain Amp Control

FMC ADC Timestamp Load (MR_FMC_ADC_TS_LD, Base+0x20)
This is the VITA packet timestamp load.

Bits	Field	R/W	Default	Description	Modules
31:0	ts_initial	R/W	0	VITA timestamp load initial value	All

Table 138. FMC ADC Timestamp Load Register

FMC ADC Timestamp Control (MR_FMC_ADC_TS_CTRL, Base+0x21)
This is the VITA packet timestamp load and control.

Bits	Field	R/W	Default	Description	Modules
0	ts_arm	R/W	0	VITA timestamp arm. Set this bit to initiate timestamp counting on rising edge of PPS when in PPS mode.	All
1	ts_pps_mode	R/W	0	VITA timestamp PPS mode. 1 = pps mode 0 = internal timer	All
3:2	tsi	R/W	11	VITA timestamp Integer-seconds mode 00 = No Integer-seconds Timestamp field included (Not supported) 01 = UTC: seconds elapsed since January 1, 1970 GMT. 10 = GPS: seconds elapsed since January 6, 1980 GMT. 11 = Other: seconds elapsed since some documented start time.	All
5:4	tsf	R/W	01	VITA timestamp Fractional-seconds mode 00 = No Fractional-seconds Timestamp field included (Not supported) 01 = Sample count timestamp: fractional seconds since last integer-seconds event, counting in samples. 10 = Real time timestamp: counts in increments of 1 picosecond since last integer-seconds event. (Not supported) 11 = Free running count timestamp: No relation to the integer-seconds field. Counts in samples.	All
31:6	-				

Table 139. FMC ADC Timestamp Control Register

FMC ADC VITA-49 framer Control (MR_FMC_ADC_VITA_CTRL, Base+0x22)
This register controls turning on/off the ADC VITA-49 framers.

Bits	Field	R/W	Default	Description	Modules
0	disable_vita	R/W		Disable ADC VITA-49 framers	All
31:1	-				

Table 140. FMC ADC VITA-49 framer control register

FMC ADC VITA Frame Sizes (MR_FMC_ADCx_VFRAME, Base+0x30 +x)
This is the VITA packet frame size for each ADCx. Number of streams defined is shown for each module.

Bits	Field	R/W	Default	Description	Modules
15:0	adcx_frame_size	R/W	0x1000	VITA packet frame size, in 32-bit words.	ADC20 = 5, FMC250 = 2, FMC110 = 2, FMC310 = 4, FMC500 = 1, FMC1000 = 1
31:16	adcx_dest_id	R/W	1	Internal routing destination Id (1=PCIE).	All

Table 141. FMC ADC VITA Frame Size Register

FMC ADC VITA Stream IDs (MR_FMC_ADCx_SID, Base+0x40 +x)
This is the VITA Stream ID (SID) each ADCx. Number of streams defined is shown for each module.

Bits	Field	R/W	Default	Description	Modules
15:0	adcx_stream_id	R/W	0	VITA packet stream Id	ADC20 = 5, FMC250, FMC110 = 2, FMC310 = 4, FMC500 = 1, FMC1000 = 1
31:16	adcx_dest_id	R/W	1	Internal routing destination Id (1=PCIE).	FMC250

Table 142. FMC ADC VITA Stream ID Register

FMC ADC Gain Error Correction (MR_FMC_ADCx_GAIN, Base+0x50 +x)
This is the gain error correction factor for each ADC. Number of ADC is shown for each module.

Bits	Field	R/W	Default	Description	Modules
17:0	adcx_gain	R/W	0x10000	Gain coefficient for error correction (0x10000 = 1.00)	ADC20 = 20, FMC500 = 2, FMC1000 = 2, FMCServo = 8
11:0	adcx_gain	R/W	0x0000	Gain coefficient for error correction, check ADC datasheet for the correct format	FMC110
30:18	-				

Table 143. FMC ADC Gain Error Correction Register

FMC ADC Offset Error Correction (MR_FMC_ADCx_OFST, Base+0x90 +x)
This is the offset error correction factor for each ADC. Number of ADC is shown for each module.

Bits	Field	R/W	Default	Description	Modules
15:0	adcx_offset	R/W	0	Offset error correction	ADC20 = 20, FMC500 = 2, FMC1000 = 2, FMCServo = 8
8:0	adcx_offset	R/W	0x100	Offset error correction, check ADC datasheet for the correct format	FMC110
31:16	-				

Table 144. FMC ADC Offset Error Correction Register

FMC DAC Registers (WB Device 15)
These are the registers for the FMC DAC control and configuration.

WB Base	WB Address	Register	Simulation Define	R/W	Description	Modules
DAC Common Registers						
0xF00	0x0		MR_FMC_DAC_EN1	R/W		FMC110, FMC250, FMC1000, FMCServo
	0x1		MR_FMC_DAC_EN2	R/W		
	0x2		MR_FMC_DAC_PDN1	R/W		FMC250, FMC1000
	0x3		MR_FMC_DAC_PDN2	R/W		
	0x4		MR_FMC_DAC_TRGR	R/W		FMC110, FMC250, FMC1000, FMCServo
	0x5		MR_FMC_DAC_DECI	R/W		FMC110, FMC250, FMC1000, FMCServo
	0x6		MR_FMC_DAC_PRI_TRGR	R/W		FMC110, FMC250, FMC1000, FMCServo
	0x7		MR_FMC_DAC_PRI	R/W		FMC110, FMC250, FMC1000, FMCServo
	0x8		MR_FMC_DAC_PRI_PARAM	R/W		FMC110, FMC250, FMC1000, FMCServo
	0x9		MR_FMC_DAC_PRI_WIDTH	R/W		FMC110, FMC250, FMC1000, FMCServo
	0xA		MR_FMC_DAC_PINC	R/W		FMC110, FMC250, FMC1000, FMCServo
	0xB		MR_FMC_DAC_RST	R/W		FMC250, FMC1000
			MR_FMC_DAC_POFF			FMCServo
	0xC		MR_FMC_DAC_CTRL	R/W		FMCServo
	0xD - 0xF					
DAC Specific Registers						
	0x10		MR_FMC_DAC_SPI_EN	R/W		FMC110
	0x11		MR_FMC_DAC_SPI_CTRL	R/W		FMC110, FMC250, FMC1000
	0x12		MR_FMC_DAC_SPI_STAT	R/W		FMC110, FMC250, FMC1000
	0x13 - 0x19		-			
	0x1A		MR_FMC_DAC_FIFO_CTRL			FMCServo
	0x1B		MR_FMC_DAC_FIFO_THRS			FMCServo
	0x1C		MR_FMC_DAC_FIFO_DATA			FMCServo
	0x1D		MR_FMC_DAC_LDAC_DLY			FMCServo
	0x1E- 0x1F					
DAC VITA Packet Configuration and Stream IDs						
	0x20- 0x2F		MR_FMC_DAC_SID	R/W		FMC110, FMC250, FMC1000
DAC Gain Registers						
	0x30- 0x6F		MR_FMC_DAC_GAIN	R/W		FMC110, FMC1000, FMCServo
DAC Offset Registers						

WB Base	WB Address	Register	Simulation Define	R/W	Description	Modules
	0x70-0xAF		MR_FMC_DAC_OFST	R/W		FMC110,FMC1000, FMCServo

Table 145. FMC DAC Component Registers

DAC Low Channel Enables (MR_FMC_DAC_EN1, Base+0x0)
This is the lower D/A 32 channel enable register.

Bits	Field	R/W	Default	Description	Modules
31:0	dac_ch_en	R/W	0x0000	D/A channel enables.	FMC110, FMC250 = 2, FMC1000 = 2, FMCServo=8

Table 146. FMC Low DAC Channel Enables Register

DAC High Channel Enables (MR_FMC_DAC_EN2, Base+0x1)
This is the lower D/A 32 channel enable register.

Bits	Field	R/W	Default	Description	Modules
31:0	dac_ch_en	R/W	0x0000	D/A channel enables.	

Table 147. FMC High DAC Channel Enables Register

DAC Low Power Enables (MR_FMC_DAC_PDN1, Base+0x2)
This is the lower D/A 32 power enable register.

Bits	Field	R/W	Default	Description	Modules
31:0	dac_pwr_en	R/W	0x0000	D/A power enables.	FMC250=2, FMC1000 = 2

Table 148. FMC Low DAC Power Enable Register

DAC High Power Enables (MR_FMC_DAC_PDN2, Base+0x3)
This is the higher D/A 32 power enable register.

Bits	Field	R/W	Default	Description	Modules
31:0	dac_pwr_en	R/W	0x0000	D/A power enables.	

Table 149. FMC High DAC Power Enable Register

FMC DAC Trigger Controls (MR_FMC_DAC_TRGR, Base+0x4)
This register configures the D/A trigger modes.

Bits	Field	R/W	Default	Description	Modules
23:0	dac_window_size	R/W	0	DAC trigger window size. This is the number of points, after decimation, that make up one data window.	FMC110, FMC250, FMC1000, FMCServo
28:24					
31:29	dac_trigger_mode	R/W	“000”	DAC trigger modes. Bit29 = rising edge (1) or level (0) Bit30 = framed (1) or unframed (0) Bit31 = external (1) or software (0)	FMC110, FMC250, FMC1000, FMCServo

Table 150. FMC DAC Trigger Controls Register

DAC Decimation (MR_FMC_DAC_DECI, Base+0x5)
This register specifies the decimation ratio for the D/A triggering.

Bits	Field	R/W	Default	Description	Modules
11:0	dac_decimation	R/W	0	Decimation count for D/A samples. Decimation keeps 1 point every N specified by this field.	FMC110, FMC250, FMC1000, FMCServo
31:12	-				

Table 151. FMC DAC Decimation Ratio Register

DAC PRI trigger enable register (MR_FMC_DAC_PRI_TRGR, Base+0x6)
This register is the DAC PRI trigger control register

Bits	Field	R/W	Default	Description	Modules
0	en_pri_trig	R/W		Enable PRI trigger mode	FMC110, FMC250, FMC1000, FMCServo
1	stop_pri	R/W		Stop PRI triggering	FMC110, FMC250, FMC1000, FMCServo
2	en_num_pri	R/W		enable finite number of PRI frames	FMC110, FMC250, FMC1000, FMCServo
3	retrig_num_pri	R/W		re-arm after number of PRI frames	FMC1000FMC110, FMC250, FMCServo
7:4	-				
8	pri_busy	R		PRI mode is running when this bit is 1	FMC110, FMC250, FMC1000, FMCServo
15:9	-				
31:16	num_pri	R/W		number of PRI frames	FMC110, FMC250, FMC1000, FMCServo

Table 152. FMC DAC PRI trigger enable Register

DAC PRI interval configuration register (MR_FMC_DAC_PRI, Base+0x7)
This register is used to set the pulse repetition interval

Bits	Field	R/W	Default	Description	Modules
31:0	pri	R/W		Pulse repetition interval	FMC110, FMC250, FMC1000, FMCServo

Table 153. FMC DAC PRI interval configuration Register

DAC PRI trigger parameters register (MR_FMC_DAC_PRI_PARAM, Base+0x8)
This register configures the DAC PRI trigger parameters

Bits	Field	R/W	Default	Description	Modules
23:0	trig_cycle_delay	R/W		Delay between trigger and sof	FMC110, FMC250, FMC1000, FMCServo

31:24	-				

Table 154. FMC DAC PRI trigger parameters Register

DAC PRI capture window configuration register (MR_FMC_DAC_PRI_WIDTH, Base+0x9)
This register configures the DAC PRI trigger parameters FIFO. Writing to this register generates a write strobe to the DAC PRI parameters FIFO which causes the width and cycle delay parameters to be written to that FIFO.

Bits	Field	R/W	Default	Description	Modules
23:0	trig_width	R/W		trigger width	FMC110, FMC250, FMC1000, FMCServo
31:24	-				

Table 155. FMC DAC PRI capture window configuration Register

FMC DAC device phase increment register (MR_FMC_DAC_PINC, Base+0xA)
This register is used to set the DDS phase increment value.

Bits	Field	R/W	Default	Description	Modules
31:0	phase_inc	R/W		DDS phase increment	FMC110, FMC250, FMC1000, FMCServo

Table 156. FMC DAC phase increment Register

FMC DAC reset register (MR_FMC_DAC_RST, Base+0xB)
This register is used to reset the DAC.

Bits	Field	R/W	Default	Description	Modules
31:0	dac_reset	R/W		DAC reset	FMC110, FMC250

Table 157. FMC DAC reset Register

FMC DAC reset register (MR_FMC_DAC_RST, Base+0xB)
This register is used to reset the DAC.

Bits	Field	R/W	Default	Description	Modules
0	dac_reset	R/W		DAC reset	FMC1000
7:1	unused				
8	dac_per_gd	R		DAC power good	FMC1000
11:9	unused				
12	dac_alarm	R			FMC1000
31:13	unused				

Table 158. FMC DAC reset Register

FMC DAC control register (MR_FMC_DAC_CTRL, Base+0xB)
This register is used to control the DAC.

Bits	Field	R/W	Default	Description	Modules
0	-				

1	dac_clk_sel	R/W		DAC clock select	FMCServo
31:2	-				

Table 159. FMC DAC control Register

FMC DAC device SPI enable register (MR_FMC_DAC_SPI_EN, Base+0x10)
This register is used to enable the DAC device that receives the SPI command.

Bits	Field	R/W	Default	Description	Modules
31:0	dac_spi_dev_en	R/W		DAC device SPI enable (1 bit per device). When a certain device's enable bit is set, then that device will receive a SPI command when the SPI registers are accessed.	FMC110 = 2

Table 160. FMC DAC device SPI enable Register

FMC DAC SPI Control (MR_FMC_DAC_SPI_CTRL, Base+0x11)
This is the D/A devices SPI port writes.

Bits	Field	R/W	Default	Description	Modules
7:0	dac_spi_wdata	R/W	0x00	DAC SPI write data	FMC110, FMC250
15:8	-				
22:16	dac_spi_addr	R/W	0	DAC SPI address	FMC110, FMC250
27:23	-				
28	dac_spi_rd_wrn	R/W	0	DAC SPI read/write bit. 0= write, 1 = read	FMC110, FMC250
31:29	-				

Table 161. FMC DAC SPI Control Register

FMC DAC SPI Control (MR_FMC_DAC_SPI_CTRL, Base+0x11)
This is the D/A devices SPI port writes.

Bits	Field	R/W	Default	Description	Modules
15:0	dac_spi_wdata	R/W	0x00	DAC SPI write data	FMC1000
22:16	dac_spi_addr	R/W	0	DAC SPI address	FMC1000
27:23	-				
28	dac_spi_rd_wrn	R/W	0	DAC SPI read/write bit. 0= write, 1 = read	FMC1000
31:29	-				

Table 162. FMC DAC SPI Control Register

FMC DAC SPI Status (MR_FMC_DAC_SPI_STAT, Base+0x12)
This is the D/A devices SPI port reads.

Bits	Field	R/W	Default	Description	Modules
7:0	dac_spi_rdata	R		DAC SPI read data	FMC110, FMC250
29:8	-				
30	dac_spi_rdy	R		DAC SPI is ready for use.	FMC110, FMC250

31	dac_spi_rdata_valid	R		DAC SPI read data is valid.	FMC110, FMC250

Table 163. FMC DAC SPI Status Register

FMC DAC SPI Status (MR_FMC_DAC_SPI_STAT, Base+0x12)
This is the D/A devices SPI port reads.

Bits	Field	R/W	Default	Description	Modules
15:0	dac_spi_rdata	R		DAC SPI read data	FMC1000
29:16	-				
30	dac_spi_rdy	R		DAC SPI is ready for use.	FMC1000
31	dac_spi_rdata_valid	R		DAC SPI read data is valid.	FMC1000

Table 164. FMC DAC SPI Status Register

FMC Servo DAC Fifo Control (MR_FMC_DAC_FIFO_CTRL, Base+0x1A)

Bits	Field	R/W	Default	Description	Modules
9:0	dac_ofifo_data_count	R		DAC ofifo data count	FMCServo
27:10	-				
28	dac_ofifo_prog_empty	R		DAC ofifo almost empty	FMCServo
29	dac_ofifo_empty	R		DAC ofifo empty	FMCServo
30	dac_ofifo_prog_full	R		DAC ofifo almost full	FMCServo
31	dac_ofifo_full			DAC ofifo full	FMCServo

Table 165. FMC Servo DAC FIFO Control

FMC Servo DAC FIFO Threshold (MR_FMC_DAC_FIFO_THRS, Base+0x1B)

Bits	Field	R/W	Default	Description	Modules
9:0	dac_ofifo_empty_thresh	R/W		DAC ofifo almost empty threshold	FMCServo
15:10	-				
25:16	dac_ofifo_full_thresh	R/W		DAC ofifo almost full threshold	FMCServo
31:26	-				

Table 166. FMC Servo DAC FIFO Threshold

FMC Servo DAC FIFO Data (MR_FMC_DAC_FIFO_DATA, Base+0x1C)

Bits	Field	R/W	Default	Description	Modules
15:0	dac_data	R		DAC ofifo data	FMCServo
31:16	-				

Table 167. FMC Servo ADC FIFO Data

FMC Servo DAC trigger delay register (MR_FMC_DAC_LDAC_DLY, Base+0x1D)

Bits	Field	R/W	Default	Description	Modules
9:0	dac_ldac_delay	R/W		Count value adds delay of 1 us	FMCServo
31:5	-				

Table 168. FMC Servo DAC trigger delay

FMC DAC VITA Stream IDs (MR_FMC_DACx_SID, Base+0x20 +x)
This is the VITA Stream ID (SID) each DACx. Number of streams defined is shown for each module.

Bits	Field	R/W	Default	Description	Modules
15:0	dacx_stream_id	R/W	0	VITA packet stream Id	FMC110 = 2, FMC250 = 1, FMC1000 = 1
31:16	dacx_dest_id	R/W	1	Internal routing destination Id (1=PCIE).	FMC110 = 2

Table 169. FMC DAC VITA Stream ID Register

FMC DAC Gain Error Correction (MR_FMC_DACx_GAIN, Base+0x30 +x)
This is the gain error correction factor for each DAC. Number of DAC is shown for each module.

Bits	Field	R/W	Default	Description	Modules
17:0	dacx_gain	R/W	0x10000	Gain coefficient for error correction (0x10000 = 1.00)	FMC110 = 2, FMC1000 = 2, FMCServo=8
31:18	-				

Table 170. FMC DAC Gain Error Correction Register

FMC DAC Offset Error Correction (MR_FMC_DACx_OFST, Base+0x70 +x)
This is the offset error correction factor for each DAC. Number of DAC is shown for each module.

Bits	Field	R/W	Default	Description	Modules
15:0	dacx_offset	R/W	0	Offset error correction	FMC110 =2, FMC1000 = 2, FMCServo=8
31:16	-				

Table 171. FMC DAC Offset Error Correction Register

FMC Aurora 0 Registers (WB Device 16)

These are the registers for Aurora port 0.

WB Base	WB Address	Register	Simulation Define	R/W	Description
0x1000	0x00		MR_FMC_RIO0_TEST_CTRL	R/W	
	0x01		MR_FMC_RIO0_CTRL_STAT	R/W	
	0x02		MR_FMC_RIO0_CMD_WR	R/W	
	0x03		MR_FMC_RIO0_CMD_RD	R/W	
	others	-			

Table 172. FMC Aurora Port 0 Component Registers

FMC Aurora Port 0 Test Control (MR_FMC_RIO0_TEST_CTRL, Base+0x00)
These are controls and status for Aurora port 0.

Bits	Field	R/W	Default	Description	Modules
0	tx_test_gen_en	R/W	0 = off	Enable test generator for transmission.	VPX, PEX, SBC
1	rx_test_chk_en	R/W	0 = off	Enable test generator for receive.	VPX, PEX, SBC
15:2	-				
31:16	test_errors	R		Test error count	VPX, PEX, SBC

Table 173. FMC Aurora 0 Test Control Register

FMC Aurora Port 0 Control/Status (MR_FMC_RIO0_CTRL_STAT, Base+0x01)
These are controls and status for Aurora port 0.

Bits	Field	R/W	Default	Description	Modules
0	Gtxreset_n	R/W	0 = reset	MGT reset, active low.	VPX, PEX,SBC
1	power_down	R/W	1 = on	MGT power down. (Turned off by default)	VPX, PEX,SBC
2	run	R/W	0 = off	Aurora interface run.	VPX, PEX,SBC
5:3	loopback	R/W	000	000 = Disable loopback 001 = Parallel 010 = Serial	VPX, PEX,SBC
6	error_clr	R/W	0 = off	Clear port error.	VPX, PEX,SBC
7	tx_channel_en	R/W	0 = off	Enable transmit channel.	VPX, PEX,SBC
8	rx_channel_en	R/W	0 = off	Enable receive channel.	VPX, PEX,SBC
22:9	-				
23	hard_error	R		Hard error. Link lost due to serious disruption.	VPX, PEX,SBC
24	soft_error	R		Soft error such as bit error.	VPX, PEX,SBC
25	frame_error	R		Frame error from Aurora.	VPX, PEX,SBC

29:26	lane_up	R		Number of lane the Aurora port is using.	VPX, PEX,SBC
30	channel_up	R		The Aurora channel is active.	VPX, PEX,SBC
31	pll_locked	R		PLL for MGT is locked.	VPX, PEX,SBC

Table 174. FMC Aurora 0 Control/Status Register

FMC Aurora Port 0 Sub-channel Write Port (MR_FMC_RIO0_CMD_WR, Base+0x02)
This is the sub-channel write port for Aurora 0.

Bits	Field	R/W	Default	Description	Modules
23:0	usr_cmd_wr_data	R/W	0	Command write data.	VPX, PEX,SBC
29:24	usr_cmd_wr_addr	R/W	0	Command write address.	VPX, PEX,SBC
30	usr_cmd_wr_rdn	R/W	0=write	Command read/write control. 0= write, 1 = read.	VPX, PEX,SBC
31	cmd_ch_rdy	R		Command sub-channel is ready.	VPX, PEX,SBC

Table 175. FMC Aurora 0 Sub-channel Write Register

FMC Aurora Port 0 Sub-channel Read Port (MR_FMC_RIO0_CMD_RD, Base+0x03)
This is the sub-channel read port for Aurora 0.

Bits	Field	R/W	Default	Description	Modules
23:0	usr_cmd_rd_data	R		Command read data.	VPX, PEX,SBC
29:24	usr_cmd_rd_addr	R		Command read address.	VPX, PEX,SBC
30	-				
31	usr_cmd_rd_vld	R		Command sub-channel is read data is valid.	VPX, PEX,SBC

Table 176. FMC Aurora 0 Sub-channel Read Register

These registers are only available for FMC SFP+/QSFP+

FMC SFP/QSFP port control/status register (MR_FMC_RIO0_CTRL_SFP/MR_FMC_RIO0_CTRL_QSFP, Base+0x04)
This is the SFP port control status register.

Bits	Field	R/W	Default	Description	Modules
3:0	sfp_disable	R/W		Transmitter disable when asserted high	SFP+
0	qsfp_modesel_n	R/W		When “low”, the module responds to 2-wire serial communication. When “high”, the module shall not respond or acknowledge any 2-wire communication	QSFP+
3:1	-			Unused	QSFP+
11:4	sfp_ratesel	R/W		SFP rate select 2 bits for every connector	SFP+
4	qsfp_reset_n	R/W		A low level on this pin resets the qsfp module	QSFP+
11:3	-			Unused	QSFP+
15:12	sfp_los	R		Receiver loss of signal indication, when high indicates that the signal level is below the specified relevant standards.	SFP+

12	qsfp_lpmode	R/W		Setting this bit high sets the qsfp module in low power mode. When low, the qsfp module is in high power mode	QSFP+
15:13	-			Unused	QSFP+
19:16	sfp_txfault	R		Module transmitter fault, when high, indicates that the module transmitter has detected a fault condition.	SFP+
16	qsfp_int_n	R		When low, indicates possible module fault	QSFP+
19:17	-			Unused	QSFP+
23:20	sfp_detect	R		Module detected, when high indicates that a module is physically present in a host slot	SFP+
20	qsfp_modpres_n	R		A high on this bit indicates module absent	QSFP+
23:21	-			Unused	QSFP+
31:24	-			Unused	

Table 177. FMC SFP port control/status register

Rate select

Bits	Field	Description	Modules
sfp_ratesel(0)	Low	RX signalling rate less than or equal to 4.25 GBd	SFP+
	High	RX signalling rate greater than 4.25 GBd	SFP+
sfp_ratesel(1)	Low	TX signalling rate less than or equal to 4.25 GBd	SFP+
	High	TX signalling rate greater than 4.25 GBd	SFP+

Table 178. Rate select hardware control

FMC SFP/QSFP port I2C register (MR_FMC_RIO0_I2C_SFP/MR_FMC_RIO0_I2C_QSFP, Base+0x05)
This is the SFP/QSFP port I2C register.

Bits	Field	R/W	Default	Description	Modules
3:0	*_sdata_o	R/W		I2C data out	SFP+ = 4, QSFP+ = 1
7:4	*_sclk	R/W		I2C clock	SFP+ = 4, QSFP+ = 1
11:8	*_sdata_i	R		I2C data in	SFP+ = 4, QSFP+ = 1
15:12	*_sclk_i	R		I2C clock (readback)	SFP+ = 4, QSFP+ = 1
31:16	-			Unused	
* is sfp for SFP+ module and qsfp for QSFP+ module					

Table 179. FMC SFP port I2C register

FMC SIO XO I2C register (MR_FMC_RIO0_I2C_SIO, Base+0x06)
This is the SIO XO I2C register.

Bits	Field	R/W	Default	Description	Modules
0	*_sio_sdo	R/W		XO I2C data out	SFP+, QSFP+
1	*_sio_sck	R/W		XO I2C clock	SFP+, QSFP+
2	*_sio_sdi	R		XO I2C data in	SFP+, QSFP+
3	*_sclk_i	R		XO I2C clock (readback)	SFP+, QSFP+
4	*_sio_intr			XO Interrupt	SFP+, QSFP+
31:5	-			Unused	
* is sfp for SFP+ module and qsfp for QSFP+ module					

Table 180. FMC SIO XO I2C register

FMC Aurora 1 Registers (WB Device 17)

These are the registers for Aurora port 1.

B Base	WB Address	Register	Simulation Define	R/W	Description
0x1100	0x00		MR_FMC_RIO1_TEST_CTRL	R/W	
	0x01		MR_FMC_RIO1_CTRL_STAT	R/W	
	0x02		MR_FMC_RIO1_CMD_WR	R/W	
	0x03		MR_FMC_RIO1_CMD_RD	R/W	
	others	-			

Table 181. FMC Aurora Port 1 Component Registers

FMC Aurora Port 1 Test Control (MR_FMC_RIO1_TEST_CTRL, Base+0x00)
These are controls and status for Aurora port 0.

Bits	Field	R/W	Default	Description	Modules
0	tx_test_gen_en	R/W	0 = off	Enable test generator for transmission.	VPX, PEX, SBC
1	rx_test_chk_en	R/W	0 = off	Enable test generator for receive.	VPX, PEX,SBC
15:2	-				
31:16	test_errors	R		Test error count	VPX, PEX,SBC

Table 182. FMC Aurora 1 Test Control Register

FMC Aurora Port 1 Control/Status (MR_FMC_RIO1_CTRL_STAT, Base+0x01)
These are controls and status for Aurora port 0.

Bits	Field	R/W	Default	Description	Modules
0	Gtxreset_n	R/W	0 = reset	MGT reset, active low.	VPX, PEX,SBC
1	power_down	R/W	1 = on	MGT power down. (Turned off by default)	VPX, PEX,SBC
2	run	R/W	0 = off	Aurora interface run.	VPX, PEX,SBC
5:3	loopback	R/W	000	000 = Disable loopback 001 = Parallel 010 = Serial	VPX, PEX,SBC
6	error_clr	R/W	0 = off	Clear port error.	VPX, PEX,SBC
7	tx_channel_en	R/W	0 = off	Enable transmit channel.	VPX, PEX,SBC
8	rx_channel_en	R/W	0 = off	Enable receive channel.	VPX, PEX,SBC
22:9	-				
23	hard_error	R		Hard error. Link lost due to serious disruption.	VPX, PEX,SBC
24	soft_error	R		Soft error such as bit error.	VPX, PEX,SBC
25	frame_error	R		Frame error from Aurora.	VPX, PEX,SBC

29:26	lane_up	R		Number of lane the Aurora port is using.	VPX, PEX,SBC
30	channel_up	R		The Aurora channel is active.	VPX, PEX,SBC
31	pll_locked	R		PLL for MGT is locked.	VPX, PEX,SBC

Table 183. FMC Aurora 1 Control/Status Register

FMC Aurora Port 1 Sub-channel Write Port (MR_FMC_RIO1_CMD_WR, Base+0x02)
This is the sub-channel write port for Aurora 0.

Bits	Field	R/W	Default	Description	Modules
23:0	usr_cmd_wr_data	R/W	0	Command write data.	VPX, PEX,SBC
29:24	usr_cmd_wr_addr	R/W	0	Command write address.	VPX, PEX,SBC
30	usr_cmd_wr_rdn	R/W	0=write	Command read/write control. 0= write, 1 = read.	VPX, PEX,SBC
31	cmd_ch_rdy	R		Command sub-channel is ready.	VPX, PEX,SBC

Table 184. FMC Aurora 1 Sub-channel Write Register

FMC Aurora Port 1 Sub-channel Read Port (MR_FMC_RIO1_CMD_RD, Base+0x03)
This is the sub-channel read port for Aurora 0.

Bits	Field	R/W	Default	Description	Modules
23:0	usr_cmd_rd_data	R		Command read data.	VPX, PEX,SBC
29:24	usr_cmd_rd_addr	R		Command read address.	VPX, PEX,SBC
30	-				
31	usr_cmd_rd_vld	R		Command sub-channel is read data is valid.	VPX, PEX,SBC

Table 185. FMC Aurora 1 Sub-channel Read Register

FMC Aurora 2 Registers (WB Device 18)

These are the registers for Aurora port 2.

B Base	WB Address	Register	Simulation Define	R/W	Description
0x1200	0x00		MR_FMC_RIO2_TEST_CTRL	R/W	
	0x01		MR_FMC_RIO2_CTRL_STAT	R/W	
	0x02		MR_FMC_RIO2_CMD_WR	R/W	
	0x03		MR_FMC_RIO2_CMD_RD	R/W	
	others	-			

Table 186. FMC Aurora Port 1 Component Registers

FMC Aurora Port 2 Test Control (MR_FMC_RIO2_TEST_CTRL, Base+0x00)
These are controls and status for Aurora port 0.

Bits	Field	R/W	Default	Description	Modules
0	tx_test_gen_en	R/W	0 = off	Enable test generator for transmission.	PEX,SBC
1	rx_test_chk_en	R/W	0 = off	Enable test generator for receive.	PEX,SBC
15:2	-				
31:16	test_errors	R		Test error count	PEX,SBC

Table 187. FMC Aurora 2 Test Control Register

FMC Aurora Port 2 Control/Status (MR_FMC_RIO2_CTRL_STAT, Base+0x01)
These are controls and status for Aurora port 0.

Bits	Field	R/W	Default	Description	Modules
0	Gtxreset_n	R/W	0 = reset	MGT reset, active low.	PEX,SBC
1	power_down	R/W	1 = on	MGT power down. (Turned off by default)	PEX,SBC
2	run	R/W	0 = off	Aurora interface run.	PEX,SBC
5:3	loopback	R/W	000	000 = Disable loopback 001 = Parallel 010 = Serial	PEX,SBC
6	error_clr	R/W	0 = off	Clear port error.	PEX,SBC
7	tx_channel_en	R/W	0 = off	Enable transmit channel.	PEX,SBC
8	rx_channel_en	R/W	0 = off	Enable receive channel.	PEX,SBC
22:9	-				
23	hard_error	R		Hard error. Link lost due to serious disruption.	PEX,SBC
24	soft_error	R		Soft error such as bit error.	PEX,SBC
25	frame_error	R		Frame error from Aurora.	PEX,SBC

27:26	lane_up	R		Number of lane the Aurora port is using.	PEX,SBC
30	channel_up	R		The Aurora channel is active.	PEX,SBC
31	pll_locked	R		PLL for MGT is locked.	PEX,SBC

Table 188. FMC Aurora 2 Control/Status Register

FMC Aurora Port 2 Sub-channel Write Port (MR_FMC_RIO2_CMD_WR, Base+0x02)
This is the sub-channel write port for Aurora 0.

Bits	Field	R/W	Default	Description	Modules
23:0	usr_cmd_wr_data	R/W	0	Command write data.	PEX,SBC
29:24	usr_cmd_wr_addr	R/W	0	Command write address.	PEX,SBC
30	usr_cmd_wr_rdn	R/W	0=write	Command read/write control. 0= write, 1 = read.	PEX,SBC
31	cmd_ch_rdy	R		Command sub-channel is ready.	PEX,SBC

Table 189. FMC Aurora 2 Sub-channel Write Register

FMC Aurora Port 2 Sub-channel Read Port (MR_FMC_RIO2_CMD_RD, Base+0x03)
This is the sub-channel read port for Aurora 0.

Bits	Field	R/W	Default	Description	Modules
23:0	usr_cmd_rd_data	R		Command read data.	PEX,SBC
29:24	usr_cmd_rd_addr	R		Command read address.	PEX,SBC
30	-				
31	usr_cmd_rd_vld	R		Command sub-channel is read data is valid.	PEX,SBC

Table 190. FMC Aurora 1 Sub-channel Read Register

FMC2 status and configuration Registers (WB Device 19)

These are the registers for FMC2 (LPC) interface.

WB Base	WB Address	Register	Simulation Define	R/W	Description
0x1300	0x00		MR_FMC_IF_STAT	R/W	
	0x01		MR_FMC_IF_CFG	R/W	FMC I2C interface
	0x02		MR_FMC_ID	R	

Table 191. FMC2 status and configuration registers

FMC2 status register (MR_FMC2_IF_STAT, Base+0x00)
These are status bits for FMC2.

Bits	Field	R/W	Default	Description	Modules
0	fmc_prsnt_m2c_l	R	-	FMC present (active low)	SBC
1	fmc_vadj_en	R/W	1	Unused	SBC
2	fmc_vadj_pwr_gd	R		Reads '1'	SBC
3	fmc_vadj_forced	R		Reads '0'	SBC
10:4	fmc_vadj_sel	R/W		Unused	SBC
15:11	-				
16	fmc_pg_m2c	R	-	FMC power good M2C	SBC
17	fmc_pg_c2m	R/W	1	FMC power good C2M	SBC
31:18	-				

Table 192. FMC status register

FMC2 I2C interface (MR_FMC2_IF_CFG, Base+0x01)
This is the I2C interface to the FMC2.

Bits	Field	R/W	Default	Description	Modules
0	fmc_sdo	R/W	-	FMC I2C data out	SBC
1	fmc_scl	R/W	-	FMC I2C clock	SBC
2	fmc_sdi	R	-	FMC I2C data in	SBC
3	-	R		FMC I2C clock readback	SBC

Table 193. FMC I2C interface

FMC2 ID (MR_FMC2_ID, Base+0x2)
This register has the FMC2 ID.

Bits	Field	R/W	Default	Description	Modules
31:0	fmc_id	R		FMC ID = Reads '2'	SBC

Table 194. FMC2 ID RegisterFMC2 DIO Registers

FMC2 LA DIO register (WB Device 20)
These are the registers for FMC2 LA DIO bus.

WB Base	WB Address	Register	Simulation Define	R/W	Description
0x1400	0x00		MR_FMC2_LA_DOUT_L	R/W	Lower 32 bit word of the FMC LA DIO bus
	0x01		MR_FMC2_LA_DOUT_H	R/W	Upper 32 bit word of the FMC LA DIO bus
	0x02		MR_FMC2_LA_OE_L	R/W	Output enable of the lower 32 bit word of the FMC LA DIO bus
	0x03		MR_FMC2_LA_OE_H	R/W	Output enable of the upper 32 bit word of the FMC LA DIO bus

Table 195. FMC LA DIO register

FMC2 AFE

FMC2 AFE Common Registers (WB Device 20)
These are the FMC common Analog Front End control and configuration registers.

WB Base	WB Address	Register	Simulation Define	R/W	Description	Modules
Clock Registers						
0x1400	0x0		MR_FMC2_PLL_CTRL	R/W		FMC310, FMC500
	0x1		MR_FMC2_PLL_SPI	R/W		FMC310, FMC500
			MR_FMC2_PLL_UW			
	0x2		MR_FMC2_VCXO	R/W		FMC500
	0x3		MR_FMC2_CLK_CTRL	R/W		
	0x4 - 0x5					
AFE Common Registers						
	0x6		MR_FMC2_TEST_CTRL	R/W		FMC310, FMC500
	0x7		MR_FMC2_SW_TRIG	R/W		FMC310, FMC500
	0x8		MR_FMC2_EXT_SYNC_CFG	R/W		FMC310, FMC500
	0x9 - 0xF					

Table 196. FMC AFE Common Registers

PLL Control (MR_FMC2_PLL_CTRL, Base+0x0)
This register has the PLL controls and status.

Bits	Field	R/W	Default	Description	Modules
0	pll_pwr_down_n	R/W	0	PLL power down. PLL is from 0.5 to 2W when operating. Allow 5 min warm-up time when device is powered up for best performance. 0 = power off, 1 = power on	
1	pll_reset	R/W	0	PLL reset	FMC310, FMC500
2	pll_mode	R/W	0	PLL configuration mode. '0' = SPI configuration '1' = load from default registers	
3	fpga_pll_clkin_stopped	R		FPGA PLL input clock stopped	
4	pll_lock	R		PLL lock indicator, '1' = locked.	
4	fpga_pll_lock	R		FPGA PLL locked	
5	fpga_pll_rst	R/W	0	FPGA PLL reset (active high)	
5	pll_ref_sel/pll_clk_sel(0)	R/W	'0'	0 = PRI, 1 = SEC	

6	pll_clk_sel(1)	R/W			
7	pll_sync	R/W	0	pll_sync	
8	pll_status_ho	R		PLL programmable status pin	
9	pll_status_ld	R		PLL programmable status pin	
10	pll_status_clkin0	R		PLL programmable status pin	
11	pll_status_clkin1	R		PLL programmable status pin	
12	pll_gpo	R		PLL general purpose output	
13	pll_status_ld1	R		PLL programmable status pin	FMC310, FMC500
14	pll_status_ld2	R		PLL programmable status pin	FMC310, FMC500
15	pll_clkin_sel0_o	W		PLL clkin selector (bit 0)	FMC310,
15	pll_clkin_sel0_i	R		PLL programmable status pin	FMC310,
15	pll_clkin_sel0	R		PLL programmable status pin	FMC500
16	pll_clkin_sel1_o	W		PLL clkin selector (bit 1)	FMC310,
16	pll_clkin_sel1_i	R		PLL programmable status pin	FMC310,
16	pll_clkin_sel1	R		PLL programmable status pin	FMC500
17	pll_clkin_sel0_dir	R/W		PLL clkin_sel0 direction (1=out, 0=in)	FMC310,
18	pll_clkin_sel1_dir	R/W		PLL clkin_sel1 direction (1=out, 0=in)	FMC310,
29:19	-			unused	
30	pll_uw/spi_rdy	R		PLL uw/spi ready	FMC310, FMC500
31	pll_uw/spi_rdata_valid	R		PLL uw/spi data valid	FMC310, FMC500

Table 197. FMC PLL Control Register

PLL SPI Data (MR_FMC2_PLL_SPI, Base+0x1)
This is the interface to the PLL SPI port.

Bits	Field	R/W	Default	Description	Modules
12:0	pll_spi_addr	R/W		PLL SPI address.	FMC310, FMC500
14:13				Unused	
15	pll_spi_rd_wrn	R/W		PLL SPI 1=read/0=write access	FMC310, FMC500
23:16	pll_spi_wdata	R/W		PLL SPI write data	FMC310, FMC500
31:24	pll_spi_rdata	R		PLL SPI read data	FMC310, FMC500

Table 198. FMC PLL SPI Data Register

VCXO Control (MR_FMC2_VCXO, Base+0x2)
This is the interface to the VCXO controls.

Bits	Field	R/W	Default	Description	Modules
3:0					

4	vcxo_pwr_en	W		VCXO power enable	FMC500
7:5					
8	vcxo_pwr_gd	R		VCXO power good	FMC500
31:9					

Table 199. FMC VCXO control Register

FMC Test Controls (MR_FMC2_TEST_CTRL, Base+0x6)
This register sets the test mode for different components in the FMC.

Bits	Field	R/W	Default	Description	Modules
0	adc_test_en	R/W	0	Enable ADC test generator. 0 = off, 1 = on	FMC310, FMC500
3:1	-				
4	adc_test_mode	R/W	0	ADC test mode: 0 = unpaced sawtooth, 1 = paced sawtooth	FMC310, FMC500
15:5	-				
16	dac_test_en	R/W	0	Enable DAC test generator. 0 = off, 1 = on	FMC500
19:17	-				
22:20	dac_test_mode	R/W	0	DAC test mode: 0 = ramp, 1 = sine, 2 = dac test pattern, 3 = zeros, 4 = max positive, 5 = max negative, 6 = alternating 1's and 0's, 7 = alternating two 1's and two 0's	FMC500
31:23	-				

Table 200. FMC Test Controls Register

FMC Software Trigger Controls (MR_FMC2_SW_TRIG, Base+0x7)
This register enables software triggering for different components in the FMC.

Bits	Field	R/W	Default	Description	Modules
0	adc_sw_trig	R/W	0	ADC software trigger. 0 = off, 1 = on	FMC310, FMC500
15:1	-				
16	dac_sw_trig	R/W	0	DAC software trigger. 0 = off, 1 = on	FMC500
31:17	-				

Table 201. FMC Software Trigger Controls Register

FMC External Sync Select (MR_FMC2_EXT_SYNC_CFG, Base+0x8)
This register allows for selecting either JP1 or FMC external trigger.

Bits	Field	R/W	Default	Description	Modules
0	ext_sync_sel	R/W	0	External sync select (0=front panel, 1=FMC)	FMC310, FMC500
31:1	unused				

Table 202. FMC External Sync Select Register

FMC2 ADC Registers (WB Device 21)
These are the registers for the FMC ADC control and configuration.

WB Base	WB Address	Register	Simulation Define	R/W	Description	Modules
ADC Common Registers						
0x1500	0x0		MR_FMC2_ADC_EN1	R/W		FMC310, FMC500
	0x1		MR_FMC2_ADC_EN2	R/W		
	0x2		MR_FMC2_ADC_PDN1	R/W		FMC310, FMC500
	0x3		MR_FMC2_ADC_PDN2	R/W		
	0x4		MR_FMC2_ADC_TRGR	R/W		FMC310, FMC500
	0x5		MR_FMC2_ADC_DECI	R/W		FMC310, FMC500
	0x6		MR_FMC2_ADC_PRI_TRGR	R/W		FMC310, FMC500
	0x7		MR_FMC2_ADC_PRI	R/W		FMC310, FMC500
	0x8		MR_FMC2_ADC_PRI_PARAM	R/W		FMC310, FMC500
	0x9		MR_FMC2_ADC_PRI_WIDTH	R/W		FMC310, FMC500
	0xA - 0xF	-				
ADC Specific Registers						
	0x10		MR_FMC2_ADC_SPI_EN	R/W		FMC310,
			MR_FMC2_ADC0_SPI_CTRL	R/W		
	0x11		MR_FMC2_ADC_SPI_CTRL	R/W		FMC310, FMC500
			MR_FMC2_ADC0_SPI_STAT	R/W		
	0x12		MR_FMC2_ADC_SPI_STAT	R/W		FMC310, FMC500
			MR_FMC2_ADC1_SPI_CTRL	R/W		
	0x13		MR_FMC2_ADC_CAL	R/W		
			MR_FMC2_ADC1_SPI_STAT	R/W		
	0x14		MR_FMC2_VGA	R/W		
			MR_FMC2_ADC_PHY_CAL	R/W		FMC500
	0x15		MR_FMC2_ADC0_CAL_STS	R/W		
	0x16		MR_FMC2_ADC1_CAL_STS	R/W		
	0x17	-				
	0x18		MR_FMC2_ADC_PHY_CAL	R/W		
	0x19		MR_FMC2_ADC_AMP_CFG	R/W		
	0x1A- 0x1F	-				
ADC VITA Packet Configuration and Timestamping						
	0x20		MR_FMC2_ADC_TS_LD	R/W		FMC310, FMC500
	0x21		MR_FMC2_ADC_TS_CTRL	R/W		FMC310, FMC500
	0x22		MR_FMC2_ADC_VITA_CTRL	R/W		FMC310, FMC500
	0x23 - 0x2F	-				
ADC VITA Frame Sizes and Stream IDs						
	0x30 - 0x34		MR_FMC2_ADC_VFRAME	R/W		FMC310 = 4, FMC500 = 2
	0x35 - 0x3F	-				
	0x40 -		MR_FMC2_ADC_SID	R/W		FMC310 = 4, FMC500 = 2

WB Base	WB Address	Register	Simulation Define	R/W	Description	Modules
	0x44					
	0x45 - 0x4F	-				
ADC Gain Registers						
	0x50 - 0x63		MR_FMC2_ADC_GAIN	R/W		FMC310 = 4
	0x64 - 0x8F	-				
ADC Offset Registers						
	0x90 - 0xA3		MR_FMC2_ADC_OFST	R/W		FMC310 = 4
	0xA4 - 0xCF					

Table 203. FMC ADC Component Registers

ADC Low Channel Enables (MR_FMC2_ADC_EN1, Base+0x0)
This is the lower A/D 32 channel enable register.

Bits	Field	R/W	Default	Description	Modules
31:0	adc_ch_en	R/W	0x0000	A/D channel enables.	FMC310 = 4, FMC500 = 2

Table 204. FMC Low ADC Channel Enables Register

ADC High Channel Enables (MR_FMC2_ADC_EN2, Base+0x1)
This is the higher A/D 32 channel enable register.

Bits	Field	R/W	Default	Description	Modules
31:0	adc_ch_en	R/W	0x0000	A/D channel enables.	

Table 205. FMC High ADC Channel Enables Register

ADC Low Power Enables (MR_FMC2_ADC_PDN1, Base+0x2)
This is the lower A/D 32 power enable register.

Bits	Field	R/W	Default	Description	Modules
31:0	adc_pwr_en	R/W	0x0000	A/D power enables.	FMC310 = 4, FMC500 = 2

Table 206. FMC Low ADC Power Enables Register

ADC High Power Enables (MR_FMC2_ADC_PDN2, Base+0x3)
This is the higher A/D 32 power enable register.

Bits	Field	R/W	Default	Description	Modules
31:0	adc_pwr_en	R/W	0x0000	A/D power enables.	

Table 207. FMC High ADC Power Enables Register

FMC ADC Trigger Controls (MR_FMC2_ADC_TRGR, Base+0x4)
This register configures the A/D trigger modes.

Bits	Field	R/W	Default	Description	Modules
23:0	adc_window_size	R/W	0	ADC trigger window size. This is the number of points, after decimation, that make up one data window.	FMC310, FMC500
28:24					
31:29	adc_trigger_mode	R/W	"000"	ADC trigger modes. Bit29 = rising edge (1) or level (0) Bit30 = framed (1) or unframed (0) Bit31 = external (1) or software (0)	FMC310, FMC500

Table 208. FMC ADC Trigger Controls Register

ADC Decimation (MR_FMC2_ADC_DECI, Base+0x5)
This register specifies the decimation ratio for the A/D triggering.

Bits	Field	R/W	Default	Description	Modules
11:0	adc_decimation	R/W	0	Decimation count for A/D samples. Decimation keeps 1 point every N specified by this field.	FMC310, FMC500
31:12	-				

Table 209. FMC ADC Decimation Ratio Register

ADC PRI trigger enable register (MR_FMC2_ADC_PRI_TRGR, Base+0x6)
This register is the ADC PRI trigger control register

Bits	Field	R/W	Default	Description	Modules
0	en_pri_trig	R/W		Enable PRI trigger mode	FMC310, FMC500
1	stop_pri	R/W		Stop PRI triggering	FMC310, FMC500
2	en_num_pri	R/W		enable finite number of PRI frames	FMC310, FMC500
3	retrig_num_pri	R/W		re-arm after number of PRI frames	FMC310, FMC500
7:4	-				
8	pri_busy	R		PRI mode is running when this bit is 1	FMC310, FMC500
15:9	-				
31:16	num_pri	R/W		number of PRI frames	FMC310, FMC500

Table 210. FMC ADC PRI trigger enable Register

ADC PRI interval configuration register (MR_FMC2_ADC_PRI, Base+0x7)

Bits	Field	R/W	Default	Description	Modules
31:0	pri	R/W		Pulse repetition interval	FMC310, FMC500

Table 211. FMC ADC PRI interval configuration Register

ADC PRI trigger parameters register (MR_FMC2_ADC_PRI_PARAM, Base+0x8)

Bits	Field	R/W	Default	Description	Modules
23:0	trig_cycle_delay	R/W		Delay between trigger and sof	FMC310, FMC500
31:24	-				

Table 212. FMC ADC PRI trigger parameters Register

ADC PRI capture window configuration register (MR_FMC2_ADC_PRI_WIDTH, Base+0x9)
This register configures the ADC PRI trigger parameters FIFO. Writing to this register generates a write strobe to the ADC PRI parameters FIFO which causes the width and cycle delay parameters to be written to that FIFO.

Bits	Field	R/W	Default	Description	Modules
23:0	trig_width	R/W		trigger width	FMC310, FMC500
31:24	-				

Table 213. FMC ADC PRI capture window configuration Register

FMC ADC device SPI enable register (MR_FMC2_ADC_SPI_EN, Base+0x10)
This register is used to enable the ADC device that receives the SPI command.

Bits	Field	R/W	Default	Description	Modules
31:0	ad_spi_dev_en	R/W		ADC device SPI enable (1 bit per device). When a certain device's enable bit is set, then that device will receive a SPI command when the SPI registers are accessed.	FMC310 = 2

Table 214. FMC ADC device SPI enable Register

FMC ADC0 SPI control register (MR_FMC2_ADC0_SPI_CTRL, Base+0x10)

Bits	Field	R/W	Default	Description	Modules
7:0	ad0_spi_wdata	R/W		ADC0 SPI write data	
15:8	-			Unused	
23:16	ad0_spi_addr	R/W		ADC0 SPI address	
27:24	-			Unused	
28	ad0_spi_rd_wrn	R/W		ADC0 SPI read/write enable	
31:29	-			Unused	

Table 215. FMC250 ADC0 device SPI control register

FMC ADC SPI Control (MR_FMC2_ADC_SPI_CTRL, Base+0x11)
This is the A/D devices SPI port writes.

Bits	Field	R/W	Default	Description	Modules
7:0	adc_spi_wdata	R/W	0x00	ADC SPI write data	FMC310
15:8	-				
15:0	adc_spi_wdata	R/W		ADC SPI write data	FMC500
27:16	adc_spi_addr	R/W	0	ADC SPI address	FMC310
22:16	adc_spi_addr	R/W	0	ADC SPI address	FMC500
28	adc_spi_rd_wrn	R/W	0	ADC SPI read/write bit. 0= write, 1 = read	FMC310, FMC500
31:29	-				

Table 216. FMC ADC SPI Control Register

FMC ADC0 SPI status register (MR_FMC2_ADC0_SPI_STAT, Base+0x11)
This is the FMC250 ADC0 SPI status register.

Bits	Field	R/W	Default	Description	Modules
7:0	ad0_spi_rdata	R	0x00	ADC0 SPI read data	
29:8	-			Unused	
30	ad0_spi_rdy	R		ADC0 SPI ready	
31	ad0_spi_rdata_valid	R		ADC0 SPI data valid	

Table 217. FMC250 ADC SPI Stauts Register

FMC ADC SPI Status (MR_FMC2_ADC_SPI_STAT, Base+0x12)
This is the A/D devices SPI port reads.

Bits	Field	R/W	Default	Description	Modules
7:0	adc_spi_rdata	R		ADC SPI read data	FMC310
29:8	-				
15:0	adc_spi_rdata	R		ADC SPI read data	FMC500
30	adc_spi_rdy	R		ADC SPI is ready for use.	FMC310, FMC500
31	adc_spi_rdata_valid	R		ADC SPI read data is valid.	FMC310, FMC500

Table 218. FMC ADC SPI Status Register

FMC ADC0 SPI control register (MR_FMC2_ADC1_SPI_CTRL, Base+0x12)
This FMC250 ADC1 SPI control register.

Bits	Field	R/W	Default	Description	Modules
7:0	ad1_spi_wdata	R/W		ADC1 SPI write data	
15:8	-			Unused	
23:16	ad1_spi_addr	R/W		ADC1 SPI address	
27:24	-			Unused	
28	ad1_spi_rd_wrn	R/W		ADC1 SPI read/write enable	
31:29	-			Unused	

Table 219. FMC250 ADC device SPI control register

FMC ADC calibration Control and Status (MR_FMC2_ADC_CAL, Base+0x13)
This is the status register for the ADC calibration.

Bits	Field	R/W	Default	Description	Modules
0	cal_start	R/W		Start ADC calibration (should be toggled after programming the ADC registers)	
15:1					
20:16	cal_done	R		ADC device calibration done	
31:21	-				

Table 220. FMC ADC calibration control and status register

FMC ADC1 SPI status register (MR_FMC_ADC1_SPI_STAT, Base+0x13)
This is the FMC250 ADC1 SPI status register.

Bits	Field	R/W	Default	Description	Modules
7:0	ad1_spi_rdata	R	0x00	ADC1 SPI read data	FMC250
29:8	-			Unused	
30	ad1_spi_rdy	R		ADC1 SPI ready	FMC250
31	ad1_spi_rdata_valid	R		ADC1 SPI data valid	FMC250

Table 221. FMC250 ADC SPI Stauts Register

FMC VGA Controls (MR_FMC2_VGA, Base+0x14)
This is the interface to the VGA. The VGA I2C port is a bit-banged interface through this register.

Bits	Field	R/W	Default	Description	Modules
0	vga_sdo	R/W	0	VGA I2C data output bit.	
1	vga_scl	R/W	0	VGA I2C clock output bit.	
2	vga_sdi	R	-	VGA I2C data input bit.	
3	vga_scl	R	-	VGA I2C port clock readback.	
7:4	-				
8	vga_ldac_n	R/W	0	Load VGA DAC latch enable (active low).	
31:9	-				

Table 222. FMC VGA Controls Register

FMC ADC PHY Calibration register (MR_FMC2_ADC_PHY_CAL, Base+0x14)
This is the FMC250 adc phy calibration register.

Bits	Field	R/W	Default	Description	Modules
1:0	adc_rst	R/W		ADC reset	
0	adc_rst	R/W		ADC reset	FMC500
15:2	-			Unused	
3:1	-			Unused	FMC500
4	adc_cal	R/W		ADC calibrate	FMC500
7:5				Unused	FMC500
8	adc_pwr_gd	R		ADC power good	FMC500
17:9				Unused	FMC500
17:16	adc_cal_done	R		ADC calibration status	
19:18					
20	adc_clk_stopped	R		ADC clock stopped	
21	adc_clk_locked	R		ADC clock locked	
31:18	-			Unused	

Table 223. FMC250 ADC PHY Calibration register

FMC ADC PHY Calibration register (MR_FMC2_ADC_PHY_CAL, Base+0x18)
This is the FMC110 adc phy calibration register.

Bits	Field	R/W	Default	Description	Modules
0	adc_phy_init	R/W		Initialize ADC PHY of the selected ADC channel	
1:7	-			Unused	
8	sel_adc_ch	R/W		Select ADC channel to forward calibration control and status	
11:9	-			Unused	
12	skip_adc_phy_cal	R/W		Skip ADC PHY calibration	
13	-			Unused	
26:14	adc_eye_aligned	R		ADC data eye is aligned	
27	adc_prbs_locked	R		local PRBS is locked to ADC bit0	
28	adc_prbs_aligned	R		ADC PRBS data sequence is aligned	
29	adc_phy_rdy	R		ADC PHY is calibrated and ready	
30	adc_clka_stopped	R		ADC output clock stopped	
31	adc_clka_locked	R		ADC output clock locked	

Table 224. FMC110 ADC PHY Calibration register

FMC ADC Amplitude Configuration (MR_FMC2_ADC_AMP_CFG, Base+0x19)

Bits	Field	R/W	Default	Description	Modules
1:0	adc_multx16_en	R/W	0	Multiply ADC output by 16	
31:2					

Table 225. FMC ADC Amplitude Configuration Register

FMC ADC Timestamp Load (MR_FMC2_ADC_TS_LD, Base+0x20)
This is the VITA packet timestamp load.

Bits	Field	R/W	Default	Description	Modules
31:0	ts_initial	R/W	0	VITA timestamp load initial value	FMC310, FMC500

Table 226. FMC ADC Timestamp Load Register

FMC ADC Timestamp Control (MR_FMC2_ADC_TS_CTRL, Base+0x21)
This is the VITA packet timestamp load and control.

Bits	Field	R/W	Default	Description	Modules
0	ts_arm	R/W	0	VITA timestamp arm. Set this bit to initiate timestamp counting on rising edge of PPS when in PPS mode.	FMC310, FMC500
1	ts_pps_mode	R/W	0	VITA timestamp PPS mode. 1 = pps mode 0 = internal timer	FMC310, FMC500
3:2	tsi	R/W	11	VITA timestamp Integer-seconds mode	FMC310, FMC500

				00 = No Integer-seconds Timestamp field included (Not supported) 01 = UTC: seconds elapsed since January 1, 1970 GMT. 10 = GPS: seconds elapsed since January 6, 1980 GMT. 11 = Other: seconds elapsed since some documented start time.	
5:4	tsf	R/W	01	VITA timestamp Fractional-seconds mode 00 = No Fractional-seconds Timestamp field included (Not supported) 01 = Sample count timestamp: fractional seconds since last integer-seconds event, counting in samples. 10 = Real time timestamp: counts in increments of 1 picosecond since last integer-seconds event. (Not supported) 11 = Free running count timestamp: No relation to the integer-seconds field. Counts in samples.	FMC310, FMC500
31:6	-				

Table 227. FMC ADC Timestamp Control Register

FMC ADC VITA-49 framer Control (MR_FMC2_ADC_VITA_CTRL, Base+0x22)
This register controls turning on/off the ADC VITA-49 framers.

Bits	Field	R/W	Default	Description	Modules
0	disable_vita	R/W		Disable ADC VITA-49 framers	FMC310, FMC500
31:1	-				

Table 228. FMC ADC VITA-49 framer control register

FMC ADC VITA Frame Sizes (MR_FMC2_ADCx_VFRAME, Base+0x30 +x)
This is the VITA packet frame size for each ADCx. Number of streams defined is shown for each module.

Bits	Field	R/W	Default	Description	Modules
15:0	adcx_frame_size	R/W	0x1000	VITA packet frame size, in 32-bit words.	FMC310 = 4, FMC500 = 2
31:16	adcx_dest_id	R/W	1	Internal routing destination Id (1=PCIE).	

Table 229. FMC ADC VITA Frame Size Register

FMC ADC VITA Stream IDs (MR_FMC2_ADCx_SID, Base+0x40 +x)
This is the VITA Stream ID (SID) each ADCx. Number of streams defined is shown for each module.

Bits	Field	R/W	Default	Description	Modules
15:0	adcx_stream_id	R/W		VITA packet stream Id	FMC310 = 4, FMC500 = 2
31:16	adcx_dest_id	R/W	1	Internal routing destination Id (1=PCIE).	FMC310 = 4, FMC500 = 2

Table 230. FMC ADC VITA Stream ID Register

FMC ADC Gain Error Correction (MR_FMC2_ADCx_GAIN, Base+0x50 +x)
This is the gain error correction factor for each ADC. Number of ADC is shown for each module.

Bits	Field	R/W	Default	Description	Modules
17:0	adcx_gain	R/W	0x10000	Gain coefficient for error correction (0x10000 = 1.00)	FMC310 = 4
11:0	adcx_gain	R/W	0x0000	Gain coefficient for error correction, check ADC datasheet for the correct format	
30:18	-				

Table 231. FMC ADC Gain Error Correction Register

FMC ADC Offset Error Correction (MR_FMC2_ADCx_OFST, Base+0x90 +x)
This is the offset error correction factor for each ADC. Number of ADC is shown for each module.

Bits	Field	R/W	Default	Description	Modules
15:0	adcx_offset	R/W	0	Offset error correction	FMC310 = 4
8:0	adcx_offset	R/W	0x100	Offset error correction, check ADC datasheet for the correct format	
31:16	-				

Table 232. FMC ADC Offset Error Correction Register

K7 Logic Library

The logic library contains the logic components used on multiple K7 products. These library components are provided with VHDL source code.

ii_4ch_fifo_drainer

Source file: ii_4ch_fifo_drainer.vhd

Description:

This component is used to move data from up to four source channel FIFOs to other logic. When the enabled source channel FIFOs have data, as indicated by the empty and almost empty flags, the data flow state machine generates the control signals to read from the source FIFOs and write to the destination. The data is read continuously in a burst mode when all the enabled source channel FIFOs' SRC_AEMPTY flag is false and DEST_RDY is true. If at least one of the source channel FIFOs' SRC_AEMPTY flag is true but its SRC_EMPTY is false and DEST_RDY is true, then the data flow is one point every 8 clocks in a “drip” mode.

The component requires that all the source FIFOs provide the AEMPTY and EMPTY flags. The AEMPTY threshold must be > 8 points to allow for latencies. The destination must provide the DEST_RDY flag, indicating that it can accept at least 8 data points without overflow.

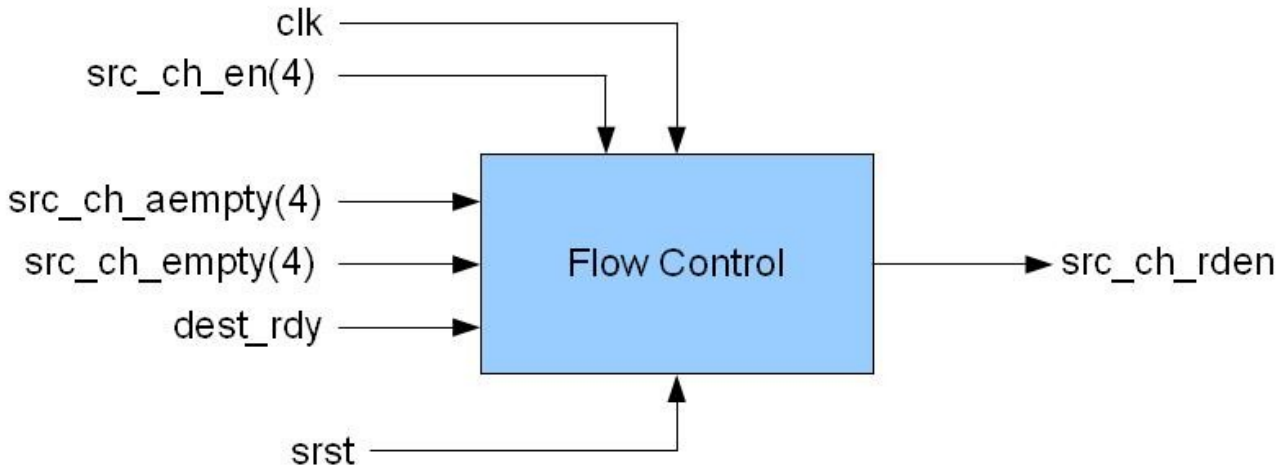


Figure 21. ii_4ch_fifo_drainer Component

The source usually has a data valid signal for each point read from the FIFO. This valid signal is used as a write enable to the destination logic.

Port	*Direction*	*Function*
srst	In	Synchronous active high reset
clk	In	Clock
src_ch_en	In	Source channel enable
src_ch_empty	In	Source channel FIFO is empty
src_ch_aempty	In	Source channel FIFO is almost empty (< 8 points)
dest_rdy	Out	Destination is ready (room for > 8 points)
src_ch_rden	In	Source channel FIFO read enable

Table 233. ii_4ch_fifo_drainer Component Ports

ii_ad9516_spi

Source file: ii_ad9516_spi.vhd

Description:

This component is an SPI port interface to the Analog Devices AD9516. The AD9516 device is configured and monitored over this SPI port. The maximum clock rate to this serial port is 25 MHz and it is configured as a 3 pin interface.

Each read/write operation is composed of 2 cycles: a 16-bit Instruction cycle followed by 8-bit data transfer cycle. Data is transferred MSB first. A strobe on spi_wr_strb initiates an SPI read or write transaction depending on the value of spi_rd_wrn.

For writes, a 16-bit instruction word is transmitted followed by an 8-bit data word. For reads, a 16-bit word instruction word is sent and an 8-bit word is received and is output on the spi_rdata byte. The spi_rdata_valid output indicates when the SPI read data is valid. The spi_rdy output indicates when the SPI port is ready for the next transfer.

Port	*Direction*	*Function*
srst	In	Synchronous active high reset
clk	In	Clock
spi_wr_strb	In	trigger an SPI transaction
spi_wdata	In	PLL SPI write data
spi_addr	In	PLL register address
spi_rd_wrn	In	PLL SPI 1=read/0=write access
spi_rdy	Out	PLL SPI port is ready
spi_rdata_valid	Out	SPI read data is valid
spi_rdata	Out	last SPI read data
spi_sclk	Out	PLL SPI clock
spi_cs_n	Out	PLL SPI enable, active low
spi_sdio	Inout	PLL SPI input/output data

Table 234. ii_ad9516_spi Component Ports

ii_alert_gen

Source file: ii_alert_gen.vhd

Description:

This component generates an alert strobe and latches the input alert bus for 4 cycles when any input alert occurs. This component is useful in generating alert triggers when multiple alerts on the same alert data bus assert and remain asserted. Otherwise, when an alert asserts on an alert line, then it will mask the detection of other alerts on the same bus since no trigger will be generated.

Generic	*Default*	*Function*
width	16	Number of alerts on the same alert data bus

Table 235. ii_alert_gen Generic Ports

Port	*Direction*	*Function*
srst	In	Synchronous active high reset
sys_clk	In	System clock
alert_din	In	alert data in
alert_strb	Out	alert strobe
alert_dout	Out	alert data out

Table 236. ii_alert_gen Component Ports

ii_alerts_top

Source files: ii_alerts_top.vhd, ii_alerts_regs.vhd, ii_alerts.vhd

Description:

The alert component is used to monitor critical system events, such as trigger active or overrange, and report these occurrences to the system. This is done by monitoring alert inputs and generating a packet to the host for each alert. In most cases, these packets are rare and tell the host software that an important event or error has occurred.

The alerts logic is comprised of 3 components: ii_alerts; responsible for generating an alert packet when any of its alert trigger inputs occurs, ii_alerts_regs; which is a wishbone slave that provides access to the registers that control the ii_alerts module, and finally ii_alerts_top; which is a wrapper around the two components mentioned earlier.

The number of alerts (num_alerts) is defined in the project package. Note that num_alerts MUST be a multiple of 4 number. The package should be modified for the number of alerts required.

The ii_alerts component monitors num_alerts input alerts and looks for rising edges on the enabled alerts. An enable for each alert is provided on the alert_enable inputs which correspond to the alerts on a bit-by-bit basis. When enabled, a rising edge indicates an alert is signaled and the logic then generates a packet indicating which alerts were triggered, the system time it was triggered and a status word for each alert. The status words can be anything of interest, the logic just puts these into the packet.

The system time is from a 32-bit counter clocked by the reference clock. This time stamp is included in each packet indicating when this alert occurred. The time stamp counter can be extended in software by enabling the time stamp rollover event so that an alert is generated to the system.

Alert Data Format

The packets are timestamped using a 32-bit counter running off the system clock, showing the system time that the alert occurred. Multiple alerts can be active for each alert packet as reported in the alerts signaled field of the packet.

Dword #	Description
0	Alerts Signaled : a '1' in a bit indicates that alert is active
1	Timestamp : a 32-bit system time clocked by a reference clock.
2.3	Reserved
3..num_alerts	Alert status words

Table 237. ii_alerts Packet Format

The array of alert data is the status word that is included in the alert packet for each enabled alert. This 32-bit word can be anything of interest and is included in the timestamped alert packet when any alert is triggered.

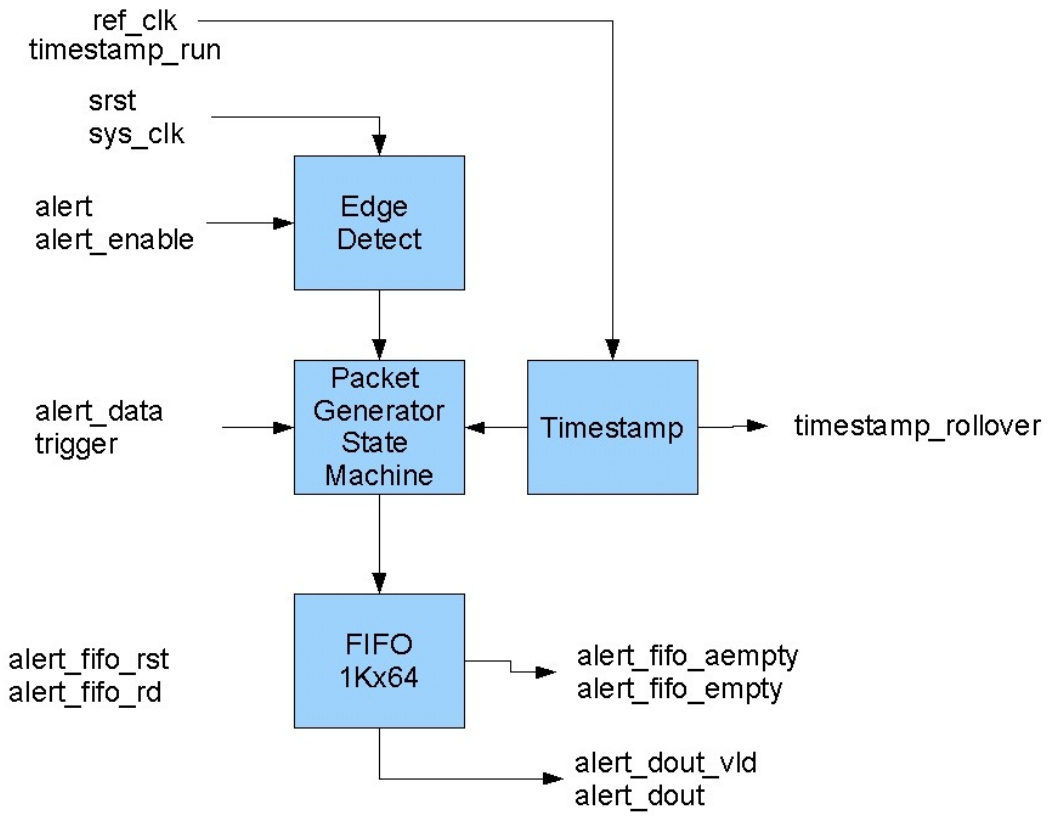


Figure 22. ii_alert Component

The alert component is usually followed by a packetizing component, such as ii_packetizer, to format the alerts for transmission to the host via the host interface such as PCI Express.

Maximum Alert Rates and Overflow Behavior

The alert log is intended for occasional use in the system, events that occur at rates expected to be no higher than 10 kHz. Since each alert generates a small data packet, higher rates require a larger data buffer. System performance may also be impacted by the large number of interrupts resulting from the packet stream.

If the FIFO fills the alert will remain pending until there is room in the FIFO for another alert packet. If the active alerts signal again when the FIFO is full however, only the first occurrence is signaled.

Generic	*Default*	*Function*
offset		Alerts Wishbone slave address offset

Table 238. ii_alerts_top Generic Ports

Port	*Direction*	*Function*
wb_rst_i	In	WB synchronous active high reset
wb_clk_i	In	WB clock
wb_adr_i	In	WB address in
wb_dat_i	In	WB data in
wb_we_i	In	WB write enable
wb_stb_i	In	WB strobe from master
wb_ack_o	Out	WB acknowledge out
wb_dat_o	Out	WB data out
srst	In	Synchronous reset
sys_clk	In	System clock
ref_clk	In	Reference clock used for timebase.
alert_data(num_alerts -1 ..0)(31..0)	In	Array of status words for the alert packet. The dimension is defined in k7_pkg for num_alerts.
alert(num_alerts -1..0)	In	Alert trigger inputs. The component monitors these signals for a rising edge.
trigger	In	Trigger
alert_sw_data	Out	Software alert status word
alert_sw_stb	Out	Software alert trigger
timestamp_rollover	Out	The timestamp counter rolled over. Used for software extending the counter.
alert_fifo_wrd_cnt	Out	Alert FIFO word count
alert_fifo_aempty	Out	Alert FIFO almost empty
alert_fifo_empty	Out	Alert FIFO empty
alert_fifo_rd	In	Alert FIFO read enable
alert_dout_vld	Out	Alert FIFO data output is valid
alert_dout(127..0)	Out	Alert FIFO data output

Table 239. ii_alerts_top Component Ports

ii_alerts_axis

Source files: ii_alerts_axis.vhd

Description:

This component is a variant of the ii_alerts_top component, that captures events in a similar way and instead sends a AXI4-Stream data with the alerts payload to the system. The main difference is the alerts_cnt input, changing to a programmable number of alerts being processed from 1 to 32. Only alerts under this number would be processed, the rest are ignored. In turn, the output AXI-Stream will vary in size according to the number of alerts and will delimit the stream using the TLAST signal on the last strobe of the stream.

This component streams alerts to the alerts processing engine built into the Multichannel DMA, which will trigger an interrupt to the software, which in turn will read the alerts payload. The alerts structure is similar to the component in ii_alerts.vhd except for its programmable variable size.

Dword #	Description
0	Alerts Signaled : a '1' in a bit indicates that alert is active
1	Timestamp : a 32-bit system time clocked by a reference clock.
2..num_alerts	Alert status words

Table 240. Alerts data structure

Port	Direction	Function
srst	In	Synchronous reset
sys_clk	In	System clock
ref_clk	In	Reference clock used for timebase.
alert_data(num_alerts -1 ..0)(31..0)	In	Array of status words for the alert packet. The dimension is defined in k7_pkg for num_alerts.
alert(31..0)	In	Alert trigger inputs. The component monitors these signals for a rising edge.
trigger	In	Trigger
timestamp_rollover	Out	The timestamp counter rolled over. Used for software extending the counter.
m_axis_tvalid	Out	AXI-Stream TVALID
m_axis_tready	In	AXI-Stream TREADY input
m_axis_tdata	Out	AXI-Stream TDATA
m_axis_tstrb	Out	AXI-Stream TSTRB
m_axis_tlast	Out	AXI-Stream TLAST

Table 241. ii_alerts_axis Component Ports

ii_bin2gray

Source file: ii_bin2gray.vhd

Description:

This component onverts the binary input into a grey coded output.

Generic	*Default*	*Function*
bw	8	Bit width

Table 242. ii_bin2gray Generic Ports

Port	*Direction*	*Function*
binary_i(bw-1:0)	In	Binary data in
gray_o(bw-1:0)	Out	Gray coded data out

Table 243. ii_bin2gray Component Ports

ii_cdce18005_spi

Source file: ii_cdce18005_spi.vhd

Description:

This component is an SPI port interface to the TI CDCE18005. The CDCD18005 device is configured and monitored over this SPI port. The maximum clock rate to the serial port is 20 MHz. For reads, a 28-bit word instruction word is sent with a 4-bit address and a 32-bit word is returned. For writes, a 32-bit word is transmitted consisting of a 4-bit address followed by 28-bits of data.

Port	*Direction*	*Function*
srst	In	Synchronous active high reset
clk	In	Clock
spi_wr_strb	In	trigger an SPI transaction
spi_wdata	In	PLL SPI write data
spi_addr	In	PLL register address
spi_rdy	Out	PLL SPI port is ready
spi_rdata_valid	Out	SPI read data is valid
spi_rdata	Out	last SPI read data
spi_sclk	Out	SPI clock
spi_le	Out	SPI load enable, active low
spi_mosi	Out	SPI master out slave in
spi_miso	In	SPI master in slave out

Table 244. ii_cdce18005_spi Component Ports

ii_cdce72010_spi

Source file: ii_cdce72010_spi.vhd

Description:

This component is an SPI port interface to the TI CDCE72010. The CDCD18005 device is configured and monitored over this SPI port. The maximum clock rate to the serial port is 20 MHz. For reads, a 28-bit word instruction word is sent with a 4-bit address and a 32-bit word is returned. For writes, a 32-bit word is transmitted consisting of a 4-bit address followed by 28-bits of data.

Port	*Direction*	*Function*
srst	In	Synchronous active high reset
clk	In	Clock
spi_wr_strb	In	trigger an SPI transaction
spi_wdata	In	PLL SPI write data
spi_addr	In	PLL register address
spi_rdy	Out	PLL SPI port is ready
spi_rdata_valid	Out	SPI read data is valid
spi_rdata	Out	last SPI read data
spi_sclk	Out	SPI clock
spi_le	Out	SPI load enable, active low
spi_mosi	Out	SPI master out slave in
spi_miso	In	SPI master in slave out

Table 245. ii_cdce72010_spi Component Ports

ii_circ_buffer

Source file: ii_circ_buffer.vhd

Description:

This component is a circular buffer that is used to synchronize signals between two clock domains. The two clocks should have the same frequency but can be of different phase.

Generic	*Default*	*Function*
WIDTH	1	Number of bits in data bus

Table 246. ii_circ_buffer Generic Ports

Port	*Direction*	*Function*
arst	In	Asynchronous active high reset
wclk	In	Write clock
wdata(WIDTH-1:0)	In	Write data
rclk	In	Read clock
rdata(WIDTH-1:0)	Out	Read data

Table 247. ii_ circ_buffer Component Ports

ii_crm

Source file: ii_crm.vhd

Description:

This component provides clocks and resets used in the K7 logic. The 200 MHz input clock is buffered and a MMCM is used to create the system clock. Resets to the logic are derived from the power on reset, board reset, and MMCM's lock so that the logic remains in reset until the clock is stable.

The resets to the system are controlled to allow the clock to stabilize before use and sequenced according to the requirements of the logic design and its dependencies. The hardware power-on-reset por_arst is the highest priority and resets the memory and all the logic. The run input is lowest priority, used for the backend reset data flow reset.

Step	Function	Result
1	Power-on-reset por_arst	All logic and memory clock MMCM is reset. All resets are asserted.
2	por_arst deasserted	Memory clock MMCM locks. Memory reset deasserts.
3	brd_arst asserts	System clock MMCM is reset.
4	brd_arst deasserted	System clock MMCM locks.
5	MMCM is Locked	clks_locked is set true. Wishbone reset deasserts.
4	app_rst is deasserted	Frontend reset deasserts.
5	run is asserted	Backend reset deasserts.

Table 248. ii_clock Reset Sequencing

As this sequence shows, the application logic will not come out of reset until the Power-on-reset and board reset are false, MMCM is locked, and app_rst is false. This allows the front end logic – PCI Express and Wishbone system control bus- to be active before the application logic for configuration.

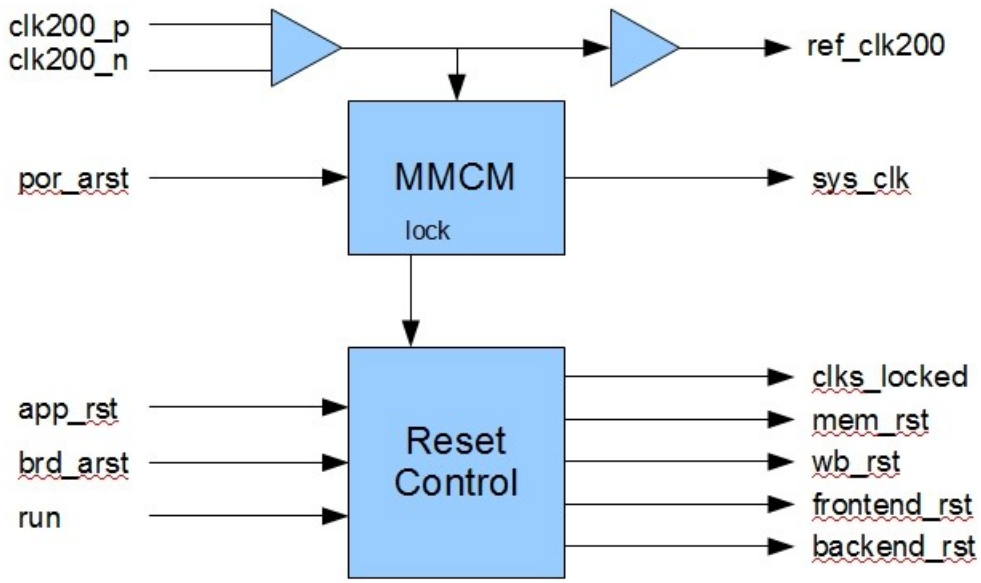


Figure 23. ii_crm Component

Generic	*Default*	*Function*
SYS_CLK_FREQ	250	System clock frequency in MHz

Table 249. ii_crm Generic Ports

Port	*Direction*	*Function*
por_arst	In	Asynchronous active high power on reset
brd_arst	In	Asynchronous active high board reset
clk200_p/n	In	200 MHz differential clock input pair
ref_clk200	Out	200MHz reference clock
sys_clk	Out	System clock output
mem_clk_div2	Out	Memory clock divided by 2
clks_locked	Out	MMCMs are locked
app_rst	In	Application reset request
run		Enable data flow
mem_rst	Out	Synchronous active high memory reset at mem_clk_div2 domain. (asserted only while power on and before memory clock lock)
wb_rst	Out	Wishbone system reset (asserted when clocks are not locked)
frontend_rst	Out	Synchronous active high reset (asserted while clocks are not locked and upon software reset request)
backend_rst	Out	Synchronous active high reset (asserted while clocks are not locked, upon software reset request, and while run is low)

Table 250. ii_crm Component Ports

ii_decimate_x2

Source file: ii_decimate_x2.vhd

Description:

This component generates an (scnt) sample output word by dropping the odd numbered samples in two (scnt) sample input data words in a frame to achieve a decimate by 2 function when not bypassed. In bypass mode, the input is passed to the output as is.

Generic	*Default*	*Function*
scnt	8	sample count
sbw	8	sample bit width

Table 251. ii_crm Generic Ports

Port	*Direction*	*Function*
srst	In	Synchronous active high reset
clk	In	Clock
bypass	In	bypass decimation
din_frame	In	input data frame
din_rdy	In	input data ready
din(scnt*sbw-1 : 0)	In	input data (scnt samples)
dout_frame	Out	data out frame
dout_vld	Out	data out is valid
dout(scnt*sbw-1 : 0)	Out	data out (scnt samples)

Table 252. ii_decimate_x2 Component Ports

ii_deframer

Source file: ii_deframer.vhd

Description:

The deframer component parses incoming packets and routes them to the peripheral device number (PDN) embedded in the header. Data is pulled from the source FIFO, is stripped of its header, and written to a destination device. Each destination has a specific PDN as defined in the PD_ADDR array.

The header for each packet has the PDN and packet size that is used by the deframer for packet routing. The deframer state machine reads the packet header and then transfers the data payload to its destination. The deframer does not do anything with the data payload – it simply passes through whatever payload is attached to a packet header.

The packet size taken from the first header word is used to move the data points as available from the source FIFO to the destination. These data moves are controlled by the flow control signals SRC_EMPTY, SRC_AEMPTY, and DEST_RDY. DEST_RDY must be true when it can accept at least 8 points. The source flags are usually the output of a FIFO.

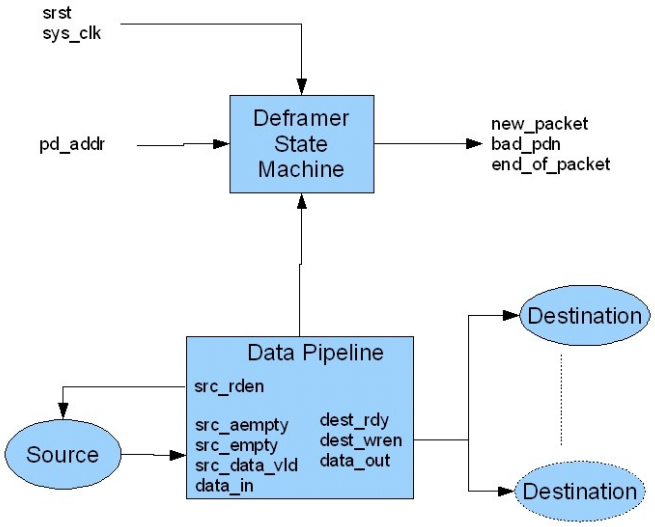


Figure 24. ii_deframer Component

The PDN definitions for device mapping are assigned to the array PD_ADDR. These addresses can be dynamic, provided that they are changed on packet boundaries.

Several signals are also available to monitor the packet flow: NEW_PACKET, BAD_PDN and END_OF_PACKET. These can be used by destination logic to know when a packet is complete for processing. The error signal BAD_PDN reports if an unknown PDN is received. If a bad PDN is detected, then the packet data is dumped.

Port	*Direction*	*Function*
srst	In	Synchronous active high reset
sys_clk	In	System clock
pd_addr	In	peripheral device numbers for decoding (defined in k7 packages)
new_packet	Out	Signals the beginning of a packet (for debug)
bad_pdn	Out	A bad PDN was detected, indicating a malformed packet
end_of_packet	Out	Signals the end of a packet (for debug)
src_aempty	In	Source is almost empty (< 8 points)
src_empty	In	Source is empty
src_rden	Out	Source read enable
src_data_vld	In	Source data is valid
data_in(127:0)	In	Data input bus (32 bits)
dest_rdy(num_pd_df-1:0)	In	Destinations are ready indicating that at least 8 points can be accepted.
dest_wren(num_pd_df-1:0)	Out	Destination write enables, one for each of the Peripheral Devices
data_out(127:0)	Out	Data bus output

Table 253. ii_deframer Component Ports

ii_destacker

Source file: ii_destacker.vhd

Description:

This component is used to split data words into two words. This is required on many K7 modules to split the 128-bit system words into two 64-bit words for the DACs. This is referred to as "destacking" the data.

The word width is specified by the generic OBW port. The OBW must be an even number. The input data path is 2x larger than the output data path.

Data flow is moderated by the DIN_RDY and DOUT_RDY flags indicating when the input and output data streams are ready to flow data. When both are ready, and a request is issued by the destination, then data is read (RDEN) from the source. The output data is received by the destination when valid (DOUT_VLD) is true.

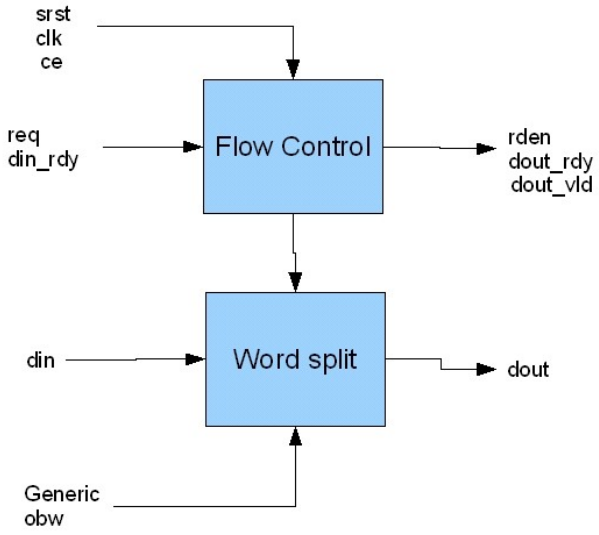


Figure 25. ii_destacker Component

Generic	*Default*	*Function*
obw	64	Data bus width, in bits

Table 254. ii_destacker Generic Ports

Port	*Direction*	*Function*
srst	In	Synchronous reset
clk	In	Clock
ce	In	Enable
req	In	Data request
rden	Out	Read enable
din_rdy	In	Input data is ready.
din(2*obw-1:0)	In	Data bus input
dout_rdy	Out	Data out is ready (available)
dout_vld	Out	Data out is valid
dout(obs-1:0)	Out	Data bus output

Table 255. ii_destacker Component Ports

ii_dio_top

Source files: ii_dio_top.vhd, ii_dio_regs.vhd, ii_dio.vhd

Description:

The DIO component provides a simple registered IO functionality for the K7 P16 DIO pins. The main purpose of this component is to provide either simple control IO or as an easily modifiable interface.

The DIO logic is comprised of 3 components: ii_dio; the basic digital I/O component that provides the interface between the DIO pins and a wishbone slave that configures, controls, and monitors these bits. ii_dio_regs; is a wishbone slave that provides access to registers that interface to the ii_dio module, and finally ii_dio_top; which is a wrapper around the two components mentioned earlier.

The direction of each bit on the DIO bus is controlled by a register.

Generic	*Default*	*Function*
width	8	DIO width
diff_en	FALSE	Enable differential mode DIO pins
addr_bits	2	DIO Wishbone slave address bits
offset		DIO Wishbone slave address offset

Table 256. ii_dio_top Generic Ports

Port	*Direction*	*Function*
wb_rst_i	In	WB synchronous active high reset
wb_clk_i	In	WB clock
wb_adr_i	In	WB address in
wb_dat_i	In	WB data in
wb_we_i	In	WB write enable
wb_stb_i	In	WB strobe from master
wb_ack_o	Out	WB acknowledge out
wb_dat_o	Out	WB data out
clk	In	System clock
dio_p(width-1:0)	InOut	Differential (P-side) DIO pins when diff_en generic is set true or even numbered single ended DIO pins when diff_en generic is set false
dio_n(width-1:0)	InOut	Differential (N-side) DIO pins when diff_en generic is set true or odd numbered single ended DIO pins when diff_en generic is set false

Table 257. ii_dio_top Component Ports

ii_drainer_destacker

Source file: ii_drainer_destacker.vhd

Description:

This component is used to move from a source FIFO to some destination logic with data split from 128-bit to 64-bit width. This component is usually used to move data from the DRAM memory to the output devices such as DACs.

When the source FIFO has data, as indicated by the empty and almost empty flags, the data flow state machine generates the control signals to read from source FIFO. The data is read continuously in a burst mode when the SRC_AEMPTY flag is false and DEST_RDY is true. If the source FIFO is almost empty, with a few points in in it, then the data flow is one point at a time in a "drip" mode.

The component requires that the source FIFO provide the AEMPTY and EMPTY flags. The AEMPTY must be > 8 points to allow for latencies. The destination must provide the DEST_RDY flag, indicating that it can accept at least 8 data points without overflow.

The source FIFO has a data valid signal for each point read from the FIFO that is used as a write enable to ii_drainer_destacker component. The data is split from a 128-bit word to two 64-bit words and written to the destination logic with DEST_WREN. Data is written to the destination logic when WR_EN is true, valid for each 64-bit word.

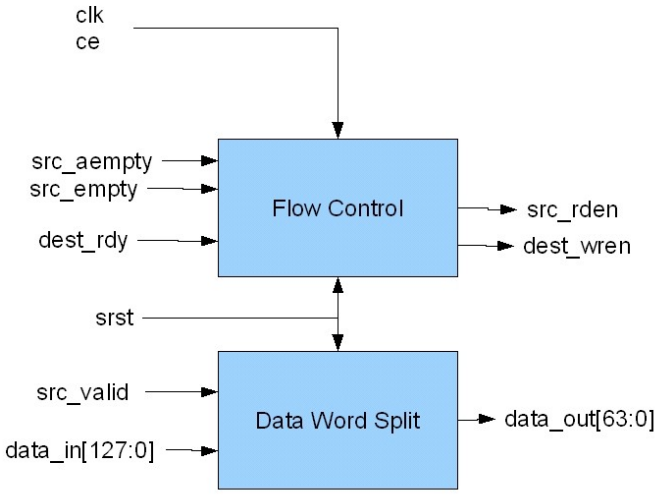


Figure 26. ii_drainer_destacker Component

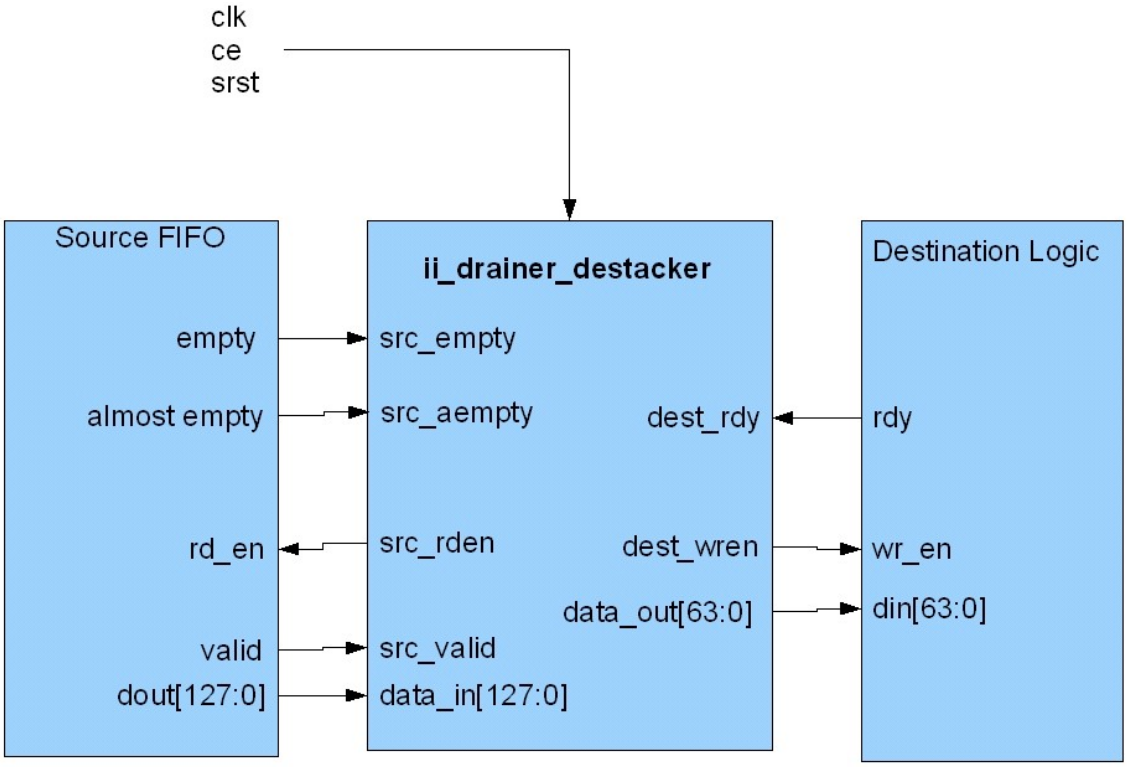


Figure 27. Using ii_drainer_destacker

Port	*Direction*	*Function*
srst	In	Synchronous active high reset
clk	In	Clock
ce	In	Enable
src_empty	In	Source FIFO is empty
src_aempty	In	Source FIFO is almost empty (<8 points)
dest_rdy	Out	Destination FIFO is ready (room for >8 points)
src_rden	Out	Source FIFO read enable
src_valid	Out	Source FIFO data valid
src_data[127:0]	In	Source FIFO data bus
dest_wren	Out	Destination write enable
dest_data[63:0]	Out	Destination data bus

Table 258. ii_drainer_destacker Component Ports

ii_ext_sync_iddr

Source file: ii_ext_sync_iddr.vhd

Description:

This component is used to capture the fast external sync signal using an IDDR component, and generate and latch a phase value corresponding to the clock cycle in which the external sync asserted. This component is used when ADC/DAC data synchronization with an external trigger is desired and when the ADC/DAC interface uses IDDR/ODDR components in its physical layer.

The captured external sync phase value is usually used to select between two sets of parallel samples captured on the interface to adjust with the external sync signal.

Port	*Direction*	*Function*
reset	In	Async active high reset
clk_bufio	In	IO clock
clk_bufr	In	BUFR (regional) clock
ext_sync_p/n	In	External sync (trigger) differential pair
trigger_en	In	Trigger enable (used to latch the phase)
ext_sync	Out	Detected external sync (in clk_bufr domain)
ext_sync_phase	Out	Calculated external sync phase

Table 259. ii_ext_sync_iddr Component Ports

ii_ext_sync_s1p4

Source file: ii_ext_sync_s1p4.vhd

Description:

This component is used to capture the fast external sync signal using an ISERDES component, and generate and latch a phase value corresponding to the clock cycle in which the external sync asserted. This component is used when ADC/DAC data synchronization with an external trigger is desired and when the ADC/DAC interface uses ISERDES/OSERDES components in its physical layer.

The captured external sync phase value is usually used to select between four sets of parallel samples captured on the interface to adjust with the external sync signal.

Port	*Direction*	*Function*
reset	In	Async active high reset
clk_bufio	In	IO clock
clk_bufr	In	BUFR (regional) clock
ext_sync_p/n	In	External sync (trigger) differential pair
trigger_en	In	Trigger enable (used to latch the phase)
idelay_rst	In	Software programmable S1P4 idelay reset
idelay	In	Software programmable S1P4 idelay value
ext_sync	Out	Detected external sync (in clk_bufr domain)
ext_sync_phase	Out	Calculated external sync phase

Table 260. ii_ext_sync_s1p4 Component Ports

ii_fifo_drainer

Source file: ii_fifo_drainer.vhd

Description:

This component is used to move data from a source FIFO to other logic. When the source FIFO has data as indicated by the empty and almost empty flags, the data flow state machine generates the control signals to read from source FIFO and write to the destination. The data is read continuously in a burst mode when the SRC_AEMPTY flag is false and DEST_RDY is true. If the source FIFO is almost empty, with a few points in in it, then the data flow is one point at a time in a "drip" mode.

The component requires that the source FIFO provide the AEMPTY and EMPTY flags. The AEMPTY must be > 8 points to allow for latencies. The destination must provide the DEST_RDY flag, indicating that it can accept at least 8 data points without overflow.

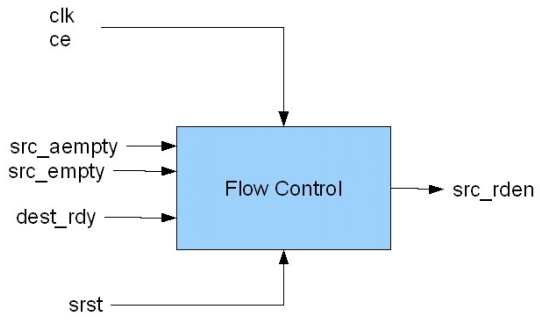


Figure 28. ii_fifo_drainer Component

The source usually has a data valid signal for each point read from the FIFO. This valid signal is used as a write enable to the destination logic.

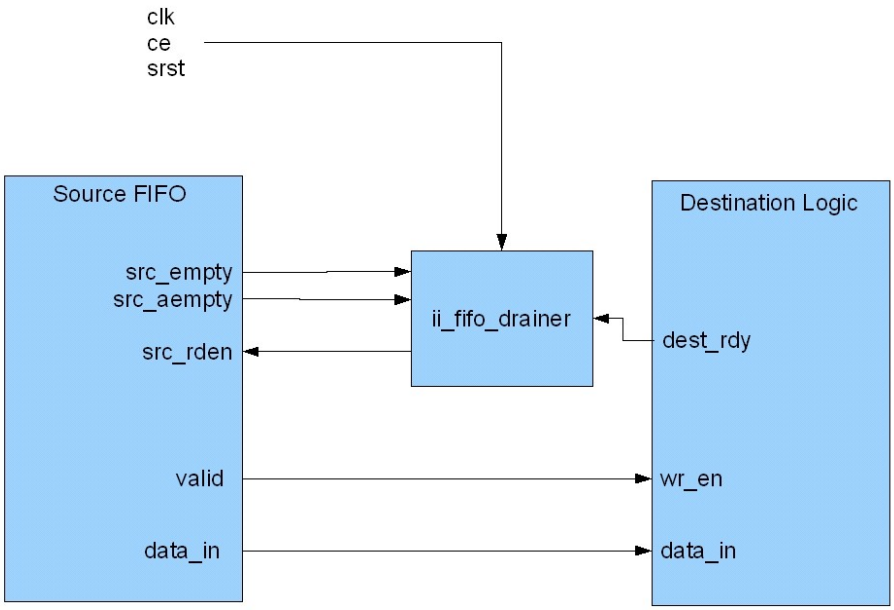


Figure 29. Using ii_fifo_drainer

Port	*Direction*	*Function*
clk	In	Clock
srst	In	Synchronous reset
ce	In	Enable
src_empty	In	Source FIFO is empty
src_aempty	In	Source FIFO is almost empty (<8 points)
dest_rdy	Out	Destination FIFO is ready (room for >8 points)
src_rden	In	Source FIFO read enable

Table 261. ii_fifo_drainer Component Ports

ii_flash_intf_top

Source files: ii_flash_intf_top.vhd, ii_flash_regs.vhd, ii_flash_spi.vhd

Description:

This component provides a link between the software and the on-board calibration ROM (serial flash memory) that is used to hold the analog front end calibration coefficients. The interface logic is comprised of 3 components: ii_flash_spi; responsile for generating the SPI interface signals to the SST25VF032B flash memory, ii_flash_regs; which is a wishbone slave that provides access to the registers that control the ii_flash_spi module, and finally ii_flash_intf_top; which is a wrapper around the two components mentioned earlier. The sequence of commands used to interface to this flash memory is included in the memory map section.

The software issues data, address, and opcode writes to initiates a serial transaction to the flash. The opcode is decoded and the number of cycles for opcode, address, data and dummy cycles and whether the transaction is a read or a write are determined based on it. The serial clock is 15.625MHz based on a system clock of 250MHz. This clock could be as fast as 80MHz for all commands (except read, which is 25MHz max), but in order to support all instructions up to a 400MHz system clock, it was intentionally limited. The serial clock is stopped between transactions to avoid analog noise interference. Note that the high speed read is not faster than the normal read command. All flash operations are fully supported.

Generic	*Default*	*Function*
offset		Flash interface Wishbone slave address offset

Table 262. ii_flash_intf_top Generic Ports

Port	*Direction*	*Function*
wb_rst_i	In	WB synchronous active high reset
wb_clk_i	In	WB clock
wb_adr_i	In	WB address in
wb_dat_i	In	WB data in
wb_we_i	In	WB write enable
wb_stb_i	In	WB strobe from master
wb_ack_o	Out	WB acknowledge out
wb_dat_o	Out	WB data out
srst	In	Synchronous reset
sys_clk	In	System clock
rom_sck	Out	Serial clock to flash
rom_cs_n	Out	Chip select to flash
rom_sdi	Out	Serial data in to flash
rom_sdo	In	Serial data out from flash
rom_hold_n	Out	Hold flash interface
rom_wp_n	Out	Flash write protect

Table 263. ii_flash_intf_top Component Ports

ii_gray2bin

Source file: ii_gray2bin.vhd

Description:

This component onverts the grey coded input into a binary output.

Generic	*Default*	*Function*
bw	8	Bit width

Table 264. ii_bin2gray Generic Ports

Port	*Direction*	*Function*
gray_i(bw-1:0)	In	Gray coded data in
binary_o(bw-1:0)	Out	Binary data out

Table 265. ii_gray2bin Component Ports

ii_offgain

Source file: ii_offgain.vhd

Description:

This component applies gain/offset error correction coefficients to the incoming data samples. The input data is multiplied by the gain coefficient first, and then an offset is added to it to compensate for analog errors. The gain factor is a 2's complement, 18-bit number ranging from +2 to -2 that allows for precise gain correction to the input data. The offset value is a 16-bit, 2's complement number that compensates for bias errors.

The error compensated output is

$$y = Gx + O$$

where x = input data, G = gain, O = offset

A gain of 1 is represented by 0x10000 and offset of 0 equal to 0.

The component uses a DSP48E block to perform the multiplication and addition followed by a signed number saturator to produce the output in the desired size.

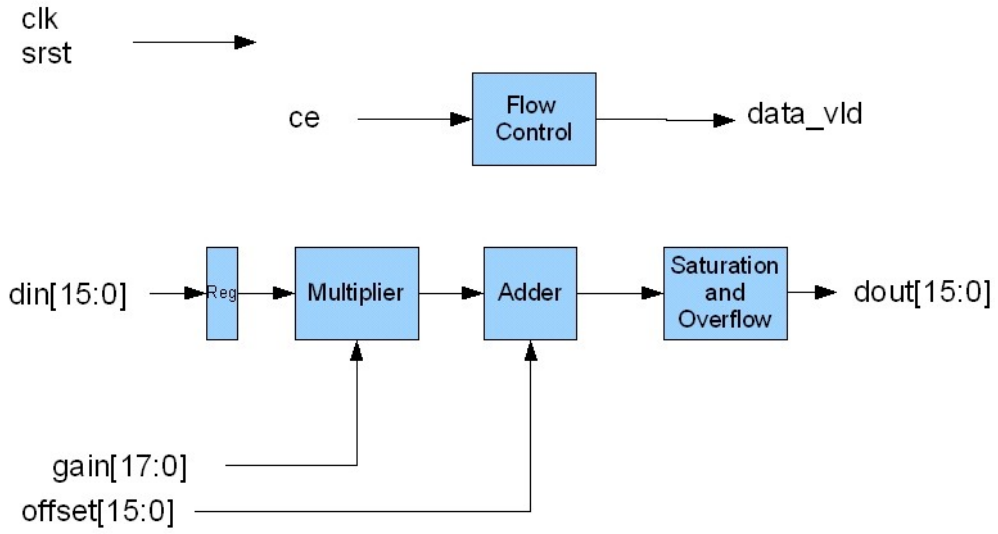


Figure 30. ii_offgain Component

Generic	*Default*	*Function*
obw	16	Output bit width

Table 266. ii_ offgain Generic Ports

Port	*Direction*	*Function*
srst	In	Synchronous reset
clk	In	Clock input
gain(17:0)	In	Gain factor, X”10000” = 1
offset(15:0)	In	Offset factor, 0 is offset of 0
ce	In	Enable
din(15:0)	In	Input data
data_vld	Out	Data is valid when true
dout(obw-1:0)	Out	Output data

Table 267. ii_offgain Component Ports

ii_packetizer_top

Source file: ii_packertizer_top.vhd, ii_packertizer_regs.vhd, ii_packertizer.vhd

Description:

The packetizer component places data streams into Velocia packets by attaching a header to a bundle of data. The primary use of these packets is to transfer data to the host using the Velocia packet system. Each data packet has a four double word header, 32-bits each, preceding the data. The packets are programmable in size and for their other routing information.

The packetizer logic is comprised of 3 components: ii_packetizer; the component that constructs Velocia packets from data received from various sources. ii_packetizer_regs; is a wishbone slave that provides access to registers that set the parameters for the ii_ packetizer module, and finally ii_packetizer_top; which is a wrapper around the two components mentioned earlier.

During operation, the packetizer scans the number of input channels in a round robin order and creates packets for the channels that are ready. Each channel has its packets built with the header information for that channel and the data payload attached to the header. The packet is transmitted as it is built to the destination; there is no data storage in the ii_packetizer component.

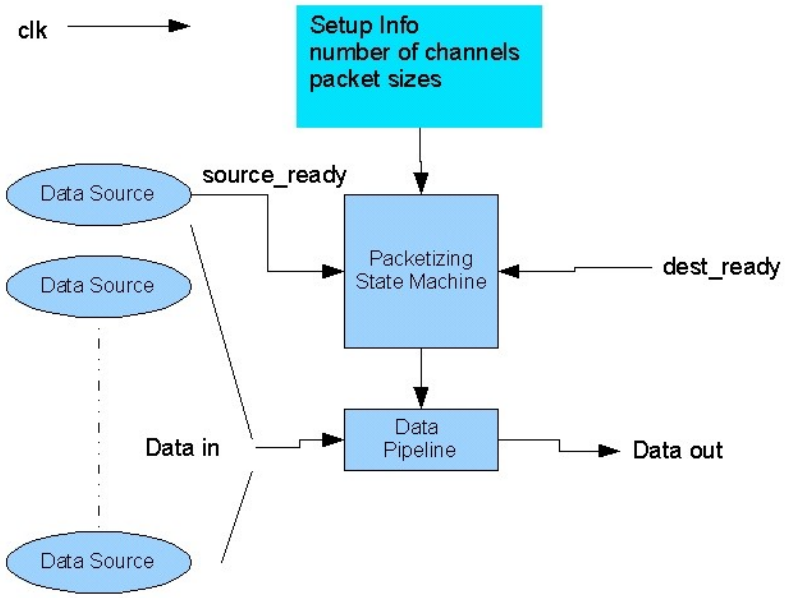


Figure 31. ii_packetizer Block Diagram

The component reads data from num_pkt_ch (defined in k7-pkg) of data sources for a max of ch_pkt_size points and gives out a packet with a header. If the amount of data available from a data source is less than the configured channel packet size programmed in the ch_pkt_size register for that channel, a packet is constructed with only the available size of data. The data width is 128; input and output are identical in size. The data sources provide data with valid after each src_rden(). Data destination must sink data continuously when dest_wren is true. The status of the source and destination devices is required by the src_data_cnt(), src_aempty(), src_empty(), and dest_rdy. No movement occurs if the destination does not have room.

Format of the packet is a four dword header, followed by a data payload. The header format is

bits[23:0] = packet size including header in dwords

bits[31:24] = peripheral device number

bits[63:32] = Auxiliary header (aux_hdr2)

bits[95:64] = Reserved (all zeros)

bits[127:96] = Reserved (all zeros)

dword = 32 bit word

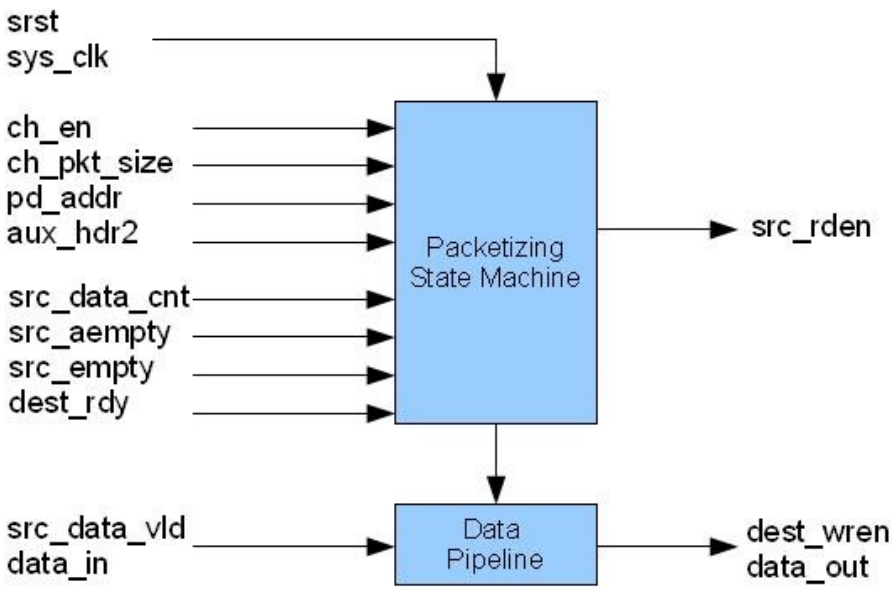


Figure 32. ii_packetizer Component

Generic	*Default*	*Function*
offset		Packetizer wishbone slave address offset

Table 268. ii_packetizer_top Generic Ports

Port	*Direction*	*Function*
wb_rst_i	In	WB synchronous active high reset
wb_clk_i	In	WB clock
wb_adr_i	In	WB address in
wb_dat_i	In	WB data in
wb_we_i	In	WB write enable
wb_stb_i	In	WB strobe from master
wb_ack_o	Out	WB acknowledge out
wb_dat_o	Out	WB data out
srst	In	Synchronous reset
sys_clk	In	System clock
src_data_cnt(num_pkt_ch-1:0)(21:0)	In	Source channel FIFO word count
src_aempty(num_pkt_ch-1:0)	In	Source channel FIFO almost empty
src_empty(num_pkt_ch-1:0)	In	Source channel FIFO empty
src_rden(num_pkt_ch-1:0)	Out	Source channel FIFO read enables
src_data_vld(num_pkt_ch-1:0)	In	Source channel data is valid
data_in(num_pkt_ch-1:0)(127:0)	In	Source channel data in array
dest_rdy	In	Destination is ready
dest_wren	Out	Destination write enable
data_out(127:0)	Out	Packetized data output bus

Table 269. ii_packetizer_top Component Ports

ii_regs_master

Source file: ii_regs_master.vhd

Description:

This component is a bridge between the PCIE register control interface and the application logic connected to Wishbone system bus. The wishbone master side is connected to all the slaves in the design. Crossing clock domains from/to PCIe clock to/from Wishbone clock is handled in this component.

Connecting to the Wishbone System Bus

When a new module with a Wishbone slave component in it is added to an K7 design, an offset address is assigned to that slave. When a Wishbone slave detects a transaction on the Wishbone system bus with an address offset matching its address offset, the Wishbone slave recognizes that the transaction is intended for it and it responds to it accordingly. It is the responsibility of the Wishbone slaves to signal the master when each transfer is complete. Transfers are limited to 16 cycles maximum. If longer cycles are required, then wait states should not be used. An alternate method such as a two-step write process should be considered if longer access times are required than 64 clocks. If the slave does not respond within 64 cycles then the master will terminate the access and return null data.

By having a Wishbone slave component in each module that needs to be configured or monitored by the software, that module can be decoupled from the overall system design for both code changes and physical layout requirements, leading to simplifying the design.

Application logic can connect to the Wishbone interface by decoding the Wishbone controls signals, use the read and write strobes or use the arrays of registers for IO. The array of registers for control and status are 32-bit each. If a register is not suitable, then read/write strobes are memory decodes for the slave memory region. Full access to the bus signals are also provided for logic devices that require multiple memory addresses or unusual decoding requirements.

Port	*Direction*	*Function*
rst	In	Active high reset on pcie_clk domain
pcie_clk	In	PCIe clock
ctrl_addr	In	PCIe control address
ctrl_din	In	PCIe control data in
ctrl_rd	In	PCIe control read strobe
ctrl_wr	In	PCIe control write strobe
ctrl_vld	Out	PCIe control data out valid
ctrl_dout	Out	PCIe control data out
wb_rst_i	In	Active high reset on system clock domain
wb_clk_i	In	system clock
wb_adr_o	Out	WB address out (to slaves)
wb_dat_o	Out	WB data out (to slaves)
wb_we_o	Out	WB write enable (to slaves)
wb_stb_o	Out	WB strobe (to slaves)
wb_cyc_o	Out	WB cycle=strobe (to slaves)
wb_ack_i	In	WB acknowledge in (from slaves)
wb_dat_i	In	WB data in (from slaves)

Table 270. ii_regs_master Component Ports

ii_stacker

Source file: ii_stacker.vhd

Description:

This component generates an obw-bit word output by stacking up n ibw-bit word input data points, where n = obw/ibn (ratio). This ratio must be an integer number.

The first input word goes to the low location of the output word and subsequent words go to the higher locations. ie. little endian. An output valid is generated after receiving every n input points.

Generic	*Default*	*Function*
ibw		Input bit width
obw		Output bit width

Table 271. ii_stacker Generic Ports

Port	*Direction*	*Function*
srst	In	Synchronous active high reset
clk	In	Clock
din_rdy	In	Input data is ready.
din(ibw-1:0)	In	Data bus input
dout_vld	Out	Data out is valid
dout(obw-1:0)	Out	Data bus output

Table 272. ii_stacker Component Ports

ii_timestamp

Source Files: ii_timestamp.vhd

Description:

The ii_timestamp component is a simple 32-bit counter used in conjunction with ii_alert to record the time of each alert.

The clock to the timestamp is usually either the sample clock or a reference clock so that the timestamp reports either the sample number or actual system time. This allows the system to parse the data stream and correlate alerts to the data samples. The CE (clock enable) input is used to gate the clock so that the counter increments only when CE and TIMESTAMP_RUN are true.

The output TIMESTAMP_TIMER is on CLK domain. When the counter rolls over from X"FFFFFFFF" to X"00000000", the TIMESTAMP_ROLLOVER output is true. TIMESTAMP_ROLLOVER is true for one CLK period and is also on the CLK domain. In K7 products, the TIMESTAMP_ROLLOVER is connected to an alert so that the software receives notification of the rollover. For a 100 MHz clock, this rollover occurs every 42.9 seconds.

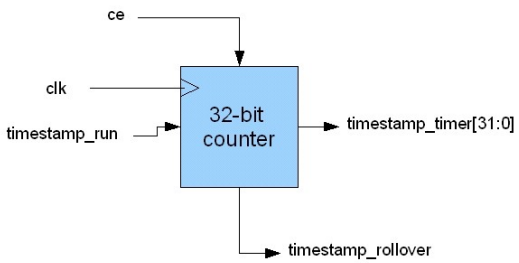


Figure 33. ii_timestamp Component

Port	*Direction*	*Function*
clk	In	clock
ce	In	Clock enable
timestamp_run	In	Time stamp run enable. The time stamp is reset to 0 when false.
timestamp_timer[31:0]	Out	Time stamp output.
timestamp_rollover	Out	The timestamp counter rolled over. Used for software extending the counter.

Table 273. ii_timestamp Component Ports

ii_trigger

Source file: ii_trigger.vhd

Description:

This component provides a trigger for data capture. Two trigger modes are supported : unframed and framed. In unframed mode, the trigger output is true whenever the selected trigger source is true. In framed mode, the trigger output is true after a rising edge on the selected trigger source until the frame count number of sample clocks are counted. A point is counted on each rising edge of the sample clock input. The trigger mode is selected with trig_mode input to be either framed or unframed, software or external trigger, positive edge or level.

The trigger sources are either external or software. The external trigger must be enabled to be used, however, the software trigger is always OR'd with the external trigger to allow it to be used anytime. In unframed mode, the mode can be positive edge or level. In the positive edge mode, the source trigger posedge enables the output, ignoring any subsequent changes. In Level mode, the output follows the source trigger.

Generic	*Default*	*Function*
SAMPLES_PER_CLK	1	Parallel samples per sample clock

Table 274. ii_trigger Generic Ports

Port	*Direction*	*Function*
reset	In	Asynchronous active high reset
clk	In	Clock
ce	In	Clock enable
ext_sync	In	External sync (trigger) input
sw_trig	In	Software trigger input
trig_mode	In	Select the trigger mode
		bit 0: 1 = posedge, 0 = level
		bit 1: 1 = framed, 0 = unframed
		bit 2: 1 = external, 0 = SW trigger
frame_size	In	Number of points in a frame count
decimation_coeff	In	Number of gaps between output triggers
decimation_en	Out	Set when (decimation_coeff /= 0)
trigger	Out	Trigger
trigger_en	Out	Trigger window (frame) is enabled

Table 275. ii_trigger Component Ports

ii_trigger_pri

Source file: ii_trigger_pri.vhd

Description:

This component when connected to the ii_trigger component periodically generates a series of trigger pulses of programmable width and delay. It only requires a single trigger (external or software) to start its operation in generating frames of programmable length. Up to 32 triggers can be generated in a frame.

This component generates single clock cycle wide train of trigger pulses at repeated intervals defined by the (pri) input parameter once a rising edge is detected on the (trig_in) input until a stop is requested by the software. A start of frame (sof) signal is generated every (pri) number of clock cycles. Each output trigger pulse within a frame is generated after (trig_cycle_del) number of clock cycles from (sof). For each trigger pulse output, its corresponding width parameter is also generated at the (dig_trig_width) port. This width parameter and trigger pulse are connected to ii_trigger's (frame_size) and (ext_sync) ports to generate trigger outputs of the desired width.

Since multiple and variable number of output triggers are needed per frame, their corresponding parameters (trig_cycle_del and trig_width) are loaded by the software and stored in a circular fifo. After a point is read from this fifo and used to generate a trigger pulse, the same point is written back into the fifo to be used during the next frame.

Port	*Direction*	*Function*
srst	In	Synchronous active high reset
clk	In	Clock
en_pri_trig	In	enable pri trigger mode
trig_in	In	external sync or sw trigger input
stop_pri	In	stop PRI triggering
pri_busy	Out	PRI trigger is running
pri	In	PRI (pulse repetition interval)
trig_fifo_rst	In	reset the pri trigger parameters' fifo
trig_fifo_wr	In	write to the pri trigger parameters' fifo
trig_cycle_del	In	delay between the trigger and sof
trig_width	In	trigger width
dig_trig_pls	Out	trigger pulse
dig_trig_idx	Out	trigger index
dig_trig_width	Out	trigger width
pri_idx	Out	pri index
pri_sof	Out	pri start-of-frame

Table 276. ii_trigger_pri Component Ports

ii_unsign_sat

Source file: ii_unsign_sat.vhd

Description:

This component saturates an unsigned input to obw bits. Saturation is done combinatorially, by comparing the MSBs of the input.

Generic	*Default*	*Function*
ibw		input bit width
obw		output bit width

Table 277. ii_unsign_sat Generic Ports

Port	*Direction*	*Function*
i	In	data in
o	Out	saturated data out

Table 278. ii_unsign_sat Component Ports

ii_vita_deframer

Source Files: ii_vita_deframer.vhd

Description:

This component reads data from a source FIFO in “drip” and “bleed” modes when a destination logic is ready and can accept data, it strips off the VITA-49 packet header and trailer, and produces the payload data with a valid per byte on its output.

For details on the VITA-49 packet header and trailer, please refer to the subpacketizer section.

Port	*Direction*	*Function*
srst	In	Synchronous active high reset
sys_clk	In	System clock
src_aempty	In	Source FIFO almost empty
src_empty	In	Source FIFO empty
src_rden	Out	Source FIFO read enables
src_vld	In	Source FIFO data is valid
src_din[127:0]	In	Source FIFO data
pkt_hdr_vld	Out	Packet header is valid on dst_out
pkt_pyld_size[13:0]	Out	Packet payload size in 128-bit words
pkt_idx[3:0]	Out	Packet index
pkt_tsi[1:0]	Out	Timestamp integer-seconds type
pkt_tsf[1:0]	Out	Timestamp fractional-seconds type
pkt_stream_id[31:0]	Out	Packet stream ID
pkt_ts_int_secs[31:0]	Out	Integer seconds in header
pkt_ts_frc_secs[63:0]	Out	Fractional time in header
dst_rdy	In	Destination is ready
dst_frame	Out	Destination write enable. (data frame)
dst_byte_vld[15:0]	Out	Destination per byte data valid
dst_dout[127:0]	Out	Destination data out

Table 279. ii_vita_deframer Component Ports

ii_vita_framer

Source Files: ii_vita_framer.vhd

Description:

This module generates a VITA-49 compliant packet. It stacks the source data to 128-bit wide and writes it to the source FIFO, then reads data back from the source FIFO once a frame worth of points are available and generates the VITA-49 packets that are written to a destination FIFO.

For details on the VITA-49 packet header and trailer, please refer to the subpacketizer section.

Generic	*Default*	*Function*
ibw	8	Input data bit width (8, 16, 32, 64, 128)

Table 280. ii_vita_framer Generic Ports

Port	*Direction*	*Function*
srst	In	Synchronous active high reset
sys_clk	In	System clock
fs_clk	In	Sample clock
frame_size	In	Frame size in words Word size specified by ge
stream_id	In	Unique stream identification (ie. PDN)
packet_type	In	VITA-49 packet type (4 bits)
ts_int_secs	In	Timestamp integer-seconds
ts_frc_secs	In	Timestamp fractional-seconds
tsi	In	Timestamp integer-seconds type
tsf	In	Timestamp fractional-seconds type
din_frame	In	frame data enable (on fs_clk)
din_vld	In	input data is valid
din	In	input data
src_fifo_afull	In	source FIFO is almost full
src_fifo_wren	Out	Write enable to source FIFO (stacked)
src_fifo_din	Out	Write data to source FIFO (stacked)
src_fifo_empty	In	Source FIFO empty flag
src_fifo_rden	Out	Read enable to source FIFO
src_fifo_vld	In	Data valid from source FIFO
src_fifo_dout	In	Data from source FIFO
dst_fifo_empty	Out	Destination FIFO empty flag
dst_fifo_aempty	Out	Destination FIFO almost empty flag
dst_fifo_rden	In	Destination FIFO read enable
dst_fifo_vld	Out	Destination FIFO valid data output
dst_fifo_dout	Out	Destination FIFO data output

Table 281. ii_vita_framer Component Ports

ii_vita_mover

Source Files: ii_vita_mover.vhd

Description:

This component routes VITA-49 format data packets, from one of num_src_ch source FIFOs into as many destination FIFOs as specified in the packet header. This component is used by the VITA router to serve one of the input channels at a time.

Generic	*Default*	*Function*
num_src_ch	4	Number of source channels
log2_num_src_ch	2	log2(num_src_ch)
num_dst_ch	3	Number of destination channels

Table 282. ii_vita_mover Generic Ports

Port	*Direction*	*Function*
srst	In	Synchronous active high reset
sys_clk	In	System clock
en_strb	In	Enable data mover strobe
src_ch_sel(log2_num_src_ch-1:0)	In	Source channel select
src_ch_hdr(127:0)	In	Source channel header
mvr_busy	Out	Mover is busy
dst_wip(num_dst_ch-1:0)	Out	Destination write in progress
src_rd_done(num_src_ch-1:0)	Out	Source channel read is done
src_aempty(num_src_ch-1:0)	In	Source channel FIFO almost empty
src_empty(num_src_ch-1:0)	In	Source channel FIFO empty
src_rden(num_src_ch-1:0)	Out	Source channel FIFO read enables
src_vld(num_src_ch-1:0)	In	Source channel FIFO data is valid
src_data(128*num_src_ch-1:0)	In	Source channel FIFO data
dst_rdy(num_dst_ch-1:0)	In	Destination FIFO ready
dst_wren	Out	Destination FIFO write
dst_data(127:0)	Out	Destination FIFO data

Table 283. ii_vita_mover Component Ports

ii_vita_router

Source Files: ii_vita_router.vhd

Description:

This component routes VITA-49 format data packets, from num_src_ch source FIFOs into up to num_dst_ch local distributed FIFOs as specified in the packet header. If more than one destination is selected in the packet header, then the packet is moved only when both destinations are available and ready for data. This mode is known as “multi-cast” packet.

Generic	*Default*	*Function*
num_src_ch	4	Number of source channels
num_dst_ch	3	Number of destination channels

Table 284. ii_vita_router Generic Ports

Port	*Direction*	*Function*
srst	In	Synchronous active high reset
sys_clk	In	System clock
src_aempty(num_src_ch-1:0)	In	Source channel FIFO almost empty
src_empty(num_src_ch-1:0)	In	Source channel FIFO empty
src_rden(num_src_ch-1:0)	Out	Source channel FIFO read enables
src_vld(num_src_ch-1:0)	In	Source channel FIFO data is valid
src_data(128*num_src_ch-1:0)	In	Source channel FIFO data
dst_rdy(num_dst_ch-1:0)	In	Destination channel ready
dst_wren(num_dst_ch-1:0)	Out	Destination channel write
dst_data(128*num_dst_ch-1:0)	Out	Destination channel data

Table 285. ii_vita_router Component Ports

ii_vita_ts

Source Files: ii_vita_ts.vhd

Description:

This module generates a VITA-49 compliant timestamp upon request. It sets the initial timestamp value by software, which then can be enabled through the 'arm' input, or by the PPS pulse input.

The integer-seconds timestamp counts in the system clock domain at 200MHz by keeping track of time as close as possible, or by the PPS pulse coming from a GPS input.

When the integer-seconds timestamp increments, it generates a pulseto the fractional-seconds counter (running on fs clock) to reset it.

Generic	*Default*	*Function*
G_SIM		True in simulation mode

Table 286. ii_vita_ts Generic Ports

Port	*Direction*	*Function*
srst	In	Synchronous active high reset
sys_clk	In	System clock
fs_clk	In	Sample clock
ts_initial	In	Initial timestamp integer-seconds value
ts_load	In	Load initial value
ts_arm	In	Start timestamp counter
pps_pls	In	PPS pulse input from ie. GPS (on sys_clk)
pps_mode	In	PPS mode enabled or internal seconds
tsf	In	TSF mode
ts_int_secs	Out	Timestamp integer-seconds value
ts_int_secs	Out	Timestamp fractional-seconds value

Table 287. ii_vita_ts Component Ports

ii_vita_velo_pad

Source Files: ii_vita_velo_pad.vhd

Description:

This component aligns VITA-49 format data packets into a Velocia packet, making sure an integer number of VITA packets fit in the requested Velocia packet size. If this is not possible, an extra VITA packet will be generated as a filler to complete the requested size. The component reads data from one data source. The data width is 128; input and output are identical in size. The data sources provide data with valid after each src_rden(). The input data is briefly stored in a 512 deep dual-quad-word FIFOwaiting to be drained by packetizer.

Port	*Direction*	*Function*
srst	In	Synchronous active high reset
sys_clk	In	System clock
ch_pkt_size[23:0]	In	Requested Velocia packet size in number of words.
force_pkt_size	In	Force Velocia size to be ch_pkt_size number of words.
bypass	Out	Bypass padding.
src_wrd_cnt[21:0]	In	Source channel word count
src_aempty	In	Source channel FIFO almost empty
src_empty	In	Source channel FIFO empty
src_rden	Out	Source channel FIFO read enable
src_vld	In	Source channel FIFO data is valid
src_data[127:0]	In	Source channel FIFO data
dst_wrd_cnt[21:0]	Out	Destination channel word count
dst_aempty	Out	Destination channel FIFO almost empty
dst_empty	Out	Destination channel FIFO empty
dst_rden	In	Destination channel FIFO read enable
dst_vld	Out	Destination channel FIFO data is valid
dst_data[127:0]	Out	Destination channel FIFO data

Table 288. ii_vita_velo_pad Component Ports

ii_vita2dma

Source file: ii_vita2dma.vhd

Description:

This component reads a VITA-49 stream from a source and converts it into a AXI4-Stream, optionally stripping its VITA headers & trailer. The StreamId field in the header is used to produce the AXI TDEST output used to route the stream to the proper DMA channel. Additionally, this field is used to decide whether to remove the VITA information.

Port	*Direction*	*Function*
srst	In	Synchronous active high reset
sys_clk	In	System clock
src_aempty	In	Source channel FIFO almost empty
src_empty	In	Source channel FIFO empty
src_rden	Out	Source channel FIFO read enable
src_vld	In	Source channel FIFO data is valid
src_din[127:0]	In	Source channel FIFO data
m_axis_tvalid	Out	AXI-Stream TVALID
m_axis_tready	In	AXI-Stream TREADY input
m_axis_tdest	Out	AXI-Stream TDEST carrying channel information
m_axis_tdata	Out	AXI-Stream TDATA
m_axis_tstrb	Out	AXI-Stream TSTRB
m_axis_tlast	Out	AXI-Stream TLAST

Table 289. ii_vita2dma Component Ports

ii_xdom_pulse

Source Files: ii_xdom_pulse.vhd

Description:

The ii_xdom_pulse component makes a pulse from one clock domain to another clock domain. The input pulse must be one SRC_CLK wide and the output pulse will be one DST_CLK wide.

Upon detecting an input pulse, a latch is toggled to convert it into a level change on the source clock doamin. The latched signal is then re-sampled on the destination clock domain and a pulse is generated whenever a level change is detected on the destination clock domain.

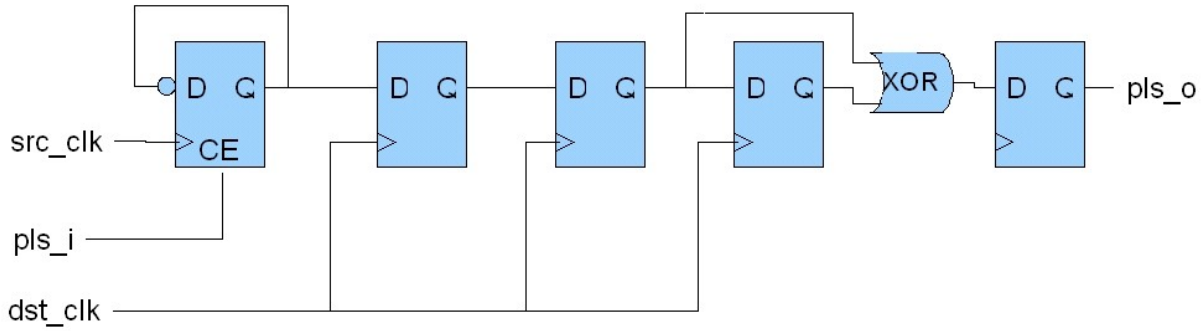


Figure 34. ii_xdom_pulse Component

Port	*Direction*	*Function*
src_clk	In	Source clock
pls_i	In	Input pulse on source clock domain
dest_clk	Out	Destination clock
pls_o	Out	Output pulse on destination clock domain

Table 290. ii_xdom_pulse Component Ports