FDK REFERENCE MANUAL
AGILENT ACQIRIS
ANALYZERS / STREAMER ANALYZERS

Models covered:

U1080A

AC240/AC210

SC240/SC210

Agilent Technologies

Manual Part Number

U1092-90010

Edition

C-RevA, November 2007

You can download the latest version of this manual from http://www.agilent.com/ by clicking on Manuals in the Technical Support section and then entering a model number. You can also visit our web site at
http://www.agilent.com/find/acqiris. At Agilent we appreciate and encourage customer input. If you have a suggestion related to the content of this manual or the presentation of information, please contact your local Agilent Acqiris product line representative or the dedicated Agilent Acqiris Technical Support (ACQIRIS_SUPPORT@agilent.com).

Acqiris Product Line Information

USA (845) 782-6544

Asia - Pacific 61 3 9210 2890

Europe 41 (22) 884 32 90

CONTENTS

1. Introduction

1.1 Message to the User

Congratulations on having purchased an Agilent Technologies Acqiris data conversion product. Acqiris Analyzers and Stream Analyzers are high-speed data acquisition modules designed for capturing high frequency electronic signals. To get the most out of the products we recommend that you read the accompanying product User Manual, the Programmer's Guide, the Programmer's Reference Manual, and this Firmware Development Kit (FDK) Reference Manual carefully. We trust that the product you have purchased as well as the accompanying software will meet with your expectations and provide you with a high quality solution to your data conversion applications.

1.2 Using this Manual

This guide assumes you are familiar with the operation of a personal computer (PC) running a Windows 95/98/2000/NT4/XP or other supported operating system. In addition you ought to be familiar with the fundamentals of the programming environment that you will be using to control your Acqiris product. It also assumes you have a good understanding of Field Programmable Gate Array (FPGA) use and basic understanding of the principles of data acquisition using either a waveform digitizer or a digital oscilloscope.

The **User Manual** that you also have received (or have access to) has important and detailed instructions concerning your Acqiris product. You should consult it first. You will find the following chapters there:

Chapter 1 ***OUT OF THE BOX,*** describes what to do when you first receive your new Acqiris product. Special attention should be paid to sections on safety, packaging, and product handling. Before installing your product please ensure that your system configuration matches or exceeds the requirements specified.

Chapter 2 ***INSTALLATION***, covers all elements of installation and performance verification. Before attempting to use your Acqiris product for actual measurements we strongly recommend that you read all sections of this chapter.

Chapter 3 ***PRODUCT DESCRIPTION***, provides a full description of all the functional elements of your product.

Chapter 4 ***FIRMWARE***, describes the major elements of firmware supplied, as standard or as an option.

Chapter 5 ***RUNNING THE AcqirisANALYZERS APPLICATION***, describes the operation of this basic application which allows you to exercise the capabilities of the analyzer.

Chapter 6 ***PROGRAMMING THE FIRMWARE***, first describes programming aspects that are common to all applications. The second part contains sections that are applicable to specific firmware applications. They are marked as such.

The **Programmer's Guide** is divided into 4 separate sections.

Chapter 1 ***INTRODUCTION,*** describes what can be found where in the documentation and how to use it.

Chapter 2 ***PROGRAMMING ENVIRONMENTS & GETTING STARTED***, provides a description for programming applications using a variety of software products and development environments.

Chapter 3 ***PROGRAMMING AN ACQIRIS DIGITIZER***, provides information on using the device driver functions to operate an Acqiris digitizer.

The **Programmer's Reference manual** is divided into 2 sections.

Chapter 1 ***INTRODUCTION,*** describes what can be found where in the documentation and how to use it.

Chapter 2 ***DEVICE DRIVER FUNCTION REFERENCE***, contains a full device driver function reference. This documents the traditional Application Program Interface (API) as it can be used in the following environments:

LabWindowsCVI, Visual C++, LabVIEW, MATLAB, Visual Basic, Visual Basic .NET.

This **FDK Reference manual** is a central document for anyone attempting to implement new functionality in the Data Processing Unit of an Analyzer or Streamer Analyzer. It is platform dependent and contains all information leading to a complete custom FPGA firmware design. It is recommended that you read it in its entirety before starting to design. It is divided into 9 separate sections.

Chapter 1	***INTRODUCTION,*** describes what can be found where in this documentation and how to use it.
Chapter 2	***INSTALLATION***, describes the installation of the FDK and the configuration of the computer.
Chapter 3	***FDK OVERVIEW***, provides basic information on the FDK environment.
Chapter 4	***OVERVIEW OF THE DATA PROCESSING*** Unit, gives details on the hardware functions associated with the FPGA.
Chapter 5	***BASE DESIGN***, describes the Base Design examples.
Chapter 6	***FDK CORE LIBRARY***, describes the available cores.
Chapter 7	*VHDL TEST BENCH*, describes the test bench and the available script commands.
Chapter 8	***DESIGN FLOW***, provides information on the supported design flow.
Chapter 9	***VHDL LIBRARIES***, provides information about the delivered vhdl libraries.

1.3 Conventions Used in This Manual:

This icon to the left of text warns that an important point must be observed.

WARNING	Denotes a warning, which advises you of precautions to take, to avoid being electrically shocked.
CAUTION	Denotes a caution, which advises you of precautions to take to avoid electrical, mechanical, or operational damages.
NOTE	Denotes a note, which alerts you to important information.
Italic	text denotes a warning, caution, or note.
Bold Italic	text is used to emphasize an important point in the text or a note
`mono`	text is used for sections of code, programming examples, and operating system commands.
B,KB,MB,GB	is for Byte, KiloByte = 1024 bytes, MegaByte = 1024*1024 bytes, GigaByte = 1024*1024*1024 bytes
b,Kb,Mb	is for bit with multipliers as above.
`Triggered`	Denotes a VHDL object. It could be a single- or multi-bit signal or a component or a library name.
`0xn..n`	Denotes a hexadecimal value.

1.4 Warning Regarding Medical Use

The Analyzer and Streamer Analyzer cards are not designed with components and testing procedures that would ensure a level of reliability suitable for use in treatment and diagnosis of humans. Applications of these cards involving medical or clinical treatment can create a potential for accidental injury caused by product failure, or by errors on the part of the user. These cards are ***not*** intended to be a substitute for any form of established process or equipment used to monitor or safeguard human health and safety in medical treatment.

WARNING: ***The modules discussed in this manual have not been designed for making direct measurements on the human body. Users who connect an Acqiris module to a human body do so at their own risk.***

1.5 Warranty

Please refer to the appropriate User Manual.

1.6 Warranty and Repair Return Procedure, Assistance, and Support

Please refer to the appropriate User Manual.

1.7 System Requirements

Please refer to the appropriate User Manual.

2. Installation

2.1 Preliminary Remarks

The FDK installation does not install any design tools. It is recommended that you install design tools prior to the FDK installation.

2.2 Installation Types

On a workstation that will be used for firmware development only – no tests or development with actual modules –, you should select the **AcqirisFDK** only installation. If the workstation will also be used for tests and/or development with actual modules, then it is recommended to choose a **Full** installation.

To work with actual modules, the standard Acqiris Software must be installed. This will be automatically verified according to your installation choices, and the Acqiris Software 3.0 installer will be run in a simple, silent mode, as part of the FDK installation if necessary. If you want to control the installation of the Acqiris drivers and software development environment, you should manually run the Acqiris Software installer prior to the FDK installation. You may also want to use a later version of the Acqiris Software. Please refer to the *User Manual - Family of 8-bit Digitizers* for detailed instructions for the installation of the standard Acqiris Software. Note that the standard demo application *AcqirisLive* does not support the Analyzer Mode for AC210/SC210 nor AC240/SC240.

2.3 FDK Installation

To install the FDK, insert the CD-ROM in the computer drive, and select **Install FDK** from the autoplay window. If the FDK window does not start automatically, run AcqirisFDK_ACSC2x0Setup.exe from the Setup folder on the CD-ROM. After the installer starts, follow the instructions carefully.

If you are upgrading to a more recent version, it is recommended that you specify a different Installation Folder, or uninstall the previous version before running AcqirisFDK_ACSC2x0Setup.exe.

Upon completion of the installer, you may need to reboot your computer.

When the installation is completed, all of the files needed for developing a new firmware (except for the Xilinx compiled libraries) will be under the FDKDesign subfolder of the Installation Folder.

HdlDesigner users should set their project mapping to the file:
Installation Folder/HdlDesigner/Mapping/ac240_fdk.hdp (or the appropriate model name).

2.4 Checking Your Installation

After the installation, we recommend that you verify the entire flow with one of the Acqiris base designs in the developer's library. The base designs are described in chapter 5, ***BASE DESIGN***.

You should simulate, synthesize, and "Place and Route" the base design. Then, generate a new .bit file and use it instead of the existing .bit file (AC240.bit / AC210.bit / SC240.bit / SC210.bit). Run the application and use the continuous acquisition mode to verify the operation with different acquisition settings (sample rate, channel combination).

The base designs of the library ac240_developer_lib are entirely implemented and delivered with the simulation, synthesis, and "Place and Route" working directories. Comparing the initial log files to those you have generated will increase your confidence that all went right.

2.5 Recommendations on Beginning a New Design

Once you have verified the installation (see above) you may start from one of the base designs modifying it to become your design.

Initially, you may want to leave the core **`user_block_example`** and its connection in place so that you can use ***AcqirisAnalyzers*** to verify the data stream and the correct operation of the module with your firmware.

The core **`user_block_example`** uses less than 1% of the FPGA gates or registers and only 8 blocks of RAM. If necessary it can simply be removed.

NOTE: Developers should modify only their own library or the files in the library ac240_developer_lib. Original files in the library ac240_fdk should not be altered.

3. FDK Overview

The Agilent Acqiris models AC240/210 Analyzers and SC240/210 Stream Analyzers are 6U compactPCI digitizers with on-board data processing in the form of a large field-programmable gate array (FPGA).

Since FPGAs are reprogrammable, these products offer the possibility of designing customer-specific computing algorithms. And, because FPGAs contain a large number of computing elements, such algorithms can be made extremely powerful, with a computing power many times that of today's high-end personal computers.

The firmware design kit (FDK) described in this manual covers everything that is needed to develop a custom application on the AC2x0 Analyzers and the SC2x0 Stream Analyzers. The following sections give a short overview of the main components of the FDK.

3.1 The FPGA Cores

The FDK is built on a set of standard cores instantiated in the FPGA. In general, each core is an interface to a system of the DPU. Among these are:

- the Local Bus interface, to connect with the PC
- the external DDR memory interface
- the interface to the digitizer data input stream

3.2 Base Designs

These are complete FPGA programs with functionality implemented to demonstrate the usage of the available cores. The AcqirisAnalyzers program will, by default, load and run the Base Design. Its source code is intentionally left open, making it the best starting point for any new design.

To make the developer's work easier, there are several base designs. There are two base designs without external memory support, a single channel version for the AC210 and a dual channel version for the AC240. There is one base design for an AC240 with support for the (optional) external memory, and one base design for an SC240 with support for the Rocket IO, implementing data streaming with the sfpdp protocol.

3.3 Reference Designs

Reference Designs are real, complete, complex applications. The source code for the reference designs is not available. There are several reference designs described in the AC2x0 or SC2x0 User Manuals.

3.4 The FPGA Design Tools

Two flows are currently supported. The standard Acqiris VHDL flow based on Mentor and Xilinx tools (HdlDesigner, ModelSim, Precision Synthesis, and ISE), and the XST flow based on the Xilinx proprietary synthesizer. For large designs or timing critical designs, Agilent Acqiris recommends using the standard flow based on Precision Synthesis.

Agilent may add new tools in the future, depending on customer requirements.

3.5 The Acqiris Test Bench

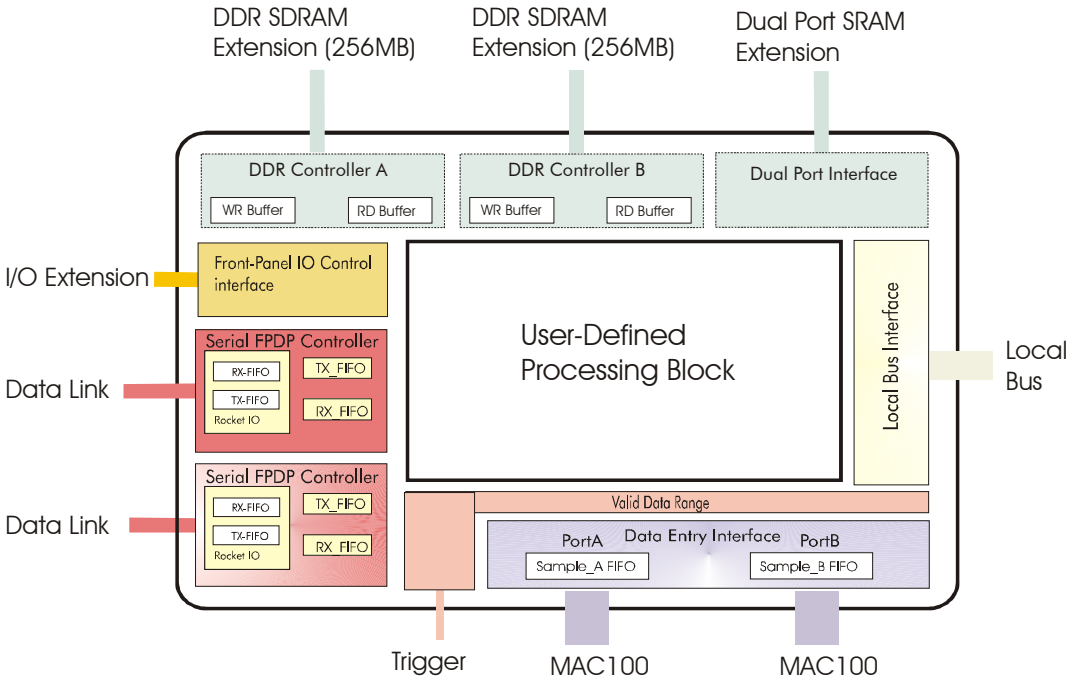
In order to simplify the functional verification by simulation, Agilent Acqiris supplies a complete VHDL test system. It is based on a set of high-level commands read from a set of test control text files. There are commands for clock generation, for reading/writing the FPGA (either with single or burst mode access), and for the generation of data input streams (which could also be read from a text file).

Developers do not have to deal with complex signal generation. This is done automatically by the Acqiris Test Bench.

3.6 The .bit File Header

It is often useful to clearly identify a .bit file, or to know what capabilities the loaded .bit file implements. Agilent Acqiris has defined a header with multiple fields for name, revision, and other comments. Some of the values can be read with a driver function. The header is inserted at the front of the .bit file with the FPGALook utility which can also be used to edit or read the .bit file header.

3.7 Overview of the FPGA Core Structure



Key Features

- **FPGA** – Xilinx XC2VP70-6 (FF1517 Package)
- **Local Bus Interface (LB) –** Interfaces the FPGA to the Local Bus through which the host PC can read from and write to the FPGA over the PCI bus. The transfer rate maximum is 132 MB/s.
- **Data Entry Interface –** Supplies the FPGA with acquisition data retrieved from one or two channels. The data rate depends on the timebase sample rate setting. The maximum rate is 1GB/s per channel.
- **Serial Front Panel Data Port (SFPDP) –** Provides an interface to external high-speed optical data link transceivers compliant with the Serial Front Panel Data Port protocol. This core should only be instantiated in firmware for SC Streamer Analyzers.
- **Dual Port Interface [Option] –** Provides an interface to the (optional) external dual port memory extension. Each port is 64 bits wide. The Dual Port Static Ram (DPSR) runs at up to 125 MHz, and provides a capacity of 128 Kwords or 1 MB.
- **DDR SDRAM memory controller [Option] –** Provides an interface to two (optional) external SDRAM extensions. Each port is 64 bits wide. Each Synchronous Dynamic Ram (SDR) block runs at up to 166 MHz, provides a capacity of 32 Mwords or 256 MB, and can achieve continuous transfer rates of ~ 2 GB/s.
- **Front Panel IO Control Interface** – Supports multiple digital connections to the front panel connectors. Digital signal type and direction are configurable within the FPGA. The interface also supports a 16-bit DAC for the generation of an analog output signal.

3.8 Accessing Registers and Memories

The user application communicates with the FPGA using the function **Acqrs_logicDeviceIO.** This function can directly address 128 registers of 32 bits. Much larger memories can be accessed through the implementation of an indirect addressing method.

Although applications can access all registers, the definition of registers 0 to 63 is reserved for Agilent Acqiris. The definition of the remaining 64 registers is entirely open.

Experience has shown that 2 * 64 registers is not sufficient. The Agilent Acqiris registers 0 to 3 are predefined to implement indirect addressing, thus extending the address space to 2^{32} addresses. The DMA data transfer uses burst readout and indirect addressing at a rate of up to 132 MB/s. Register use is further described in section 4.5.1 ***EXTENSION OF THE INTERNAL BUS ADDRESSING SPACE***.

4. Overview of the Data Processing Unit

4.1 Digitizer

The ACxx0 and SCxx0 are fully described in chapter 3 of their respective user manuals. It is recommended to read that before this document.

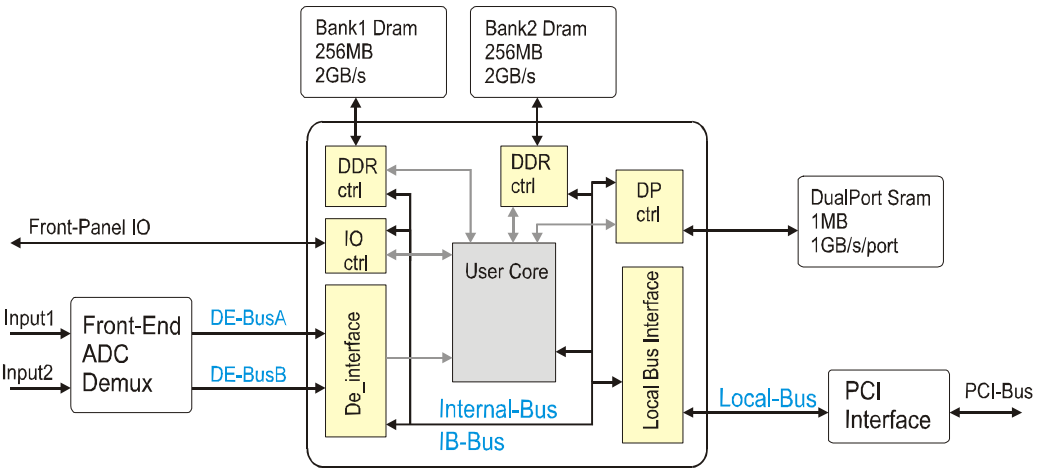
4.2 Data Processing Unit (DPU)

The DPU is the combination of a Virtex2P FPGA – XC2VP70 speed-6 – and (optional) external memories. There are two instances of a large dynamic Double-Data-Rate high-speed memory (DDRM) for applications needing more memory than available within the FPGA. The additional static Dual-Port memory (DPM) is useful for applications needing simple fast random access memory.

The PCI Interface connects the analyzer card to the host computer through the PCI bus. It translates the complex PCI transactions to the simpler ones of the Local Bus, which connects all major components of the cards, including the FPGA.

Within the FPGA, Agilent Acqiris has defined the Internal Bus (IB), a different data transfer protocol better adapted to an FPGA implementation. The User Core is controlled through the FPGA Internal Bus. The User Core has direct access to all memories and the IO control interface. The De_Interface simply feeds the User Core with acquisition data.

There is additional trigger capability not shown below.



The analog input signals are passed through signal-conditioning amplifiers, where the coupling, offset, and gain can be programmed. Each signal is sampled at up to 1 GS/s and converted to 8-bit values. They are multiplexed to blocks of 16 samples, at up to 62.5 MHz, and passed to the data processing unit.

In case of an AC240 or a SC240 module, the front panel input **INPUT1** corresponds to the data flow B (DE-BusB) and the front panel input **INPUT2** corresponds to the data flow A (DE-BusA). In case of an AC210 or a SC210, the front panel input **INPUT1** corresponds to the data flow A (DE-BusA).

In interleaved operation of the AC240 or SC240, a single signal (Input 1 or 2) is converted by both ADCs in a time-shifted manner, so as to effectively achieve twice the conversion rate of a single ADC. The data flow A has the odd samples (0,2,4,…) and the data flow B has the even samples (1,3,5,…)

4.2.1 The FPGA – XC2VP70-6

More information on the XC2VP70-6 can be found in the Xilinx documentation. Please refer to it at http://www.xilinx.com/. Its major characteristics are listed below.

Name	Qty	Description / Comment
Logic cells	66176	1 Logic cell has 1x (4 Input LUT + Flip-Flop + Carry Logic)
Block Ram	5.9 Mb	328 instances of 18 kb block ram
Multiplier	328	18x18-bit multiplier
DCM	8	Digital clock manager including frequency synthesis and phase shift.

Name	Qty	Description / Comment
		Frequency up to 420 MHz
Rocket IO	16	16 instances of 3.25 Gb/s serializer, 12 instances usable in the design
PowerPC	2	Currently not supported by Agilent
.bit file Size	3.2 MB	Size of the .bit file for FPGA configuration

4.2.2 DPU Clocking Resources

There are 3 different clock source types that can be used within the Data Processing Unit.

1. **CK33M** and **CK66M** are two clocks derived from the PCI clock fed by the CompactPCI backplane. **CK33M** is a 33 MHz clock whereas **CK66M** is a 66 MHz clock. They are always available (continuous clock).
2. **DECLKA** is driven by the data demultiplexer chip located on the bottom of the board (MACA). Its frequency is derived from the ADC Sampling clock (FS/16) and depends on the Acquisition mode. **DECLKB** is similar to **DECLKA** and is driven by the upper data multiplexer when using the second acquisition channel (available in the AC240 or SC240). Depending on the Acquisition mode these clocks may be stopped. When running, **DECLKA** and **DECLKB** always have the same frequency but they could be phase shifted depending on the acquisition settings (Interleaved acquisition).
3. **RefCKA** and **RefCKB** are driven by an external PLL and are intended to generate a clock reference for serial transmission in the SC analyzers. **RefCKA** and **RefCKB** always have the same frequency but may exhibit a small phase shift due to PCB routing delay. It should be noted that **RefCKA** is intended for top edge RocketIO instances whereas RefCKB is intended for bottom edge ones.

The others signals allocated to the clock pads are either dedicated to DCM feedback (**DMem_FB**, **DPA_CLKFB**, **DPB_CLKFB**) or used for low and stable input propagation delay (**TRIGA**).

The clock outputs to the SRAM and DRAM memories use standard IO pads and are phase locked to the internal clock driving the outputs (**DPA_CLK, DPB_CLK, DDRA_CK, DDRB_CK**).

The Data Processing Unit offers up to sixteen clocks pads (IBUFG_ primitives). Each clock pad can be grouped by 2 to provide a differential clock buffer (IBFUGDS_ primitives). The clock source allocation is frozen by the layout of the PCB board and is described in the next table.

The table below presents the various clock sources at the DPU pad level:

Clocks PAD	Allocation	Comments
GCLK0S	**DECLKA**	ADC Sampling Clock / 16 – Channel 0
GCLK1P	**DECLKA**	ADC Sampling Clock / 16 – Channel 0
GCLK2S	**RefCKA_p**	Programmable Reference Clocks for top edge RocketIO
GCLK3P	**RefCKA_n**	
GCLK4S	**TrigA_p**	Trigger Accepted
GCLK5P	**TrigA_n**	
GCLK6S	**--**	Not Used
GCLK7P	**DECLKB**	ADC Sampling Clock / 16 – Channel 1
GCLK0P	**RefCKB_p**	Programmable Reference Clocks for bottom edge RocketIO
GCLK1S	**RefCKB_n**	
GCLK2P	**DPA_CLKFB**	Clock Feedback for the SRAM Memory, port A
GCLK3S	**DPB_CLKFB**	Clock Feedback for the SRAM Memory, port B
GCLK4P	**DMem_FB_p**	Clock Feedback for the DRAM Memory, Bank A & B
GCLK5S	**DMem_FB_n**	
GCLK6P	**CK33M**	33 MHz Local Bus Clock
GCLK7S	**CK66M**	66 MHz Local Bus Clock

Firmware for the SC or AC Analyzers can use up to 14 different clock domains with some area restrictions.

Agilent Acqiris supplies several clock manager cores because a single solution does not cover enough applications. Two global clocks are distributed throughout the whole FPGA (**`lbclkg / sysclk`**). The User Core should only use these two clock domains. The other 14 possible clock domains are used by the Agilent Acqiris-supplied cores. For further details, please refer to chapter 6 ***FDK CORE LIBRARY***. Developers should not have to deal with clock management and generation unless very specific needs make it unavoidable.

4.3 Understanding DMA Transfers

Direct Memory Access (DMA) transfers are the fastest way of transmitting data from the FPGA to the host computer. They are supported in the context of the Indirect Read Access mode (to be described later in section 4.5 ***INTERNAL BUS***). After the overhead of some initialization, the operation can read data at each clock transition until the transfer is complete. A validation signal follows the data to indicate valid data; thus the target (i.e. the PCI interface) can control the data flow. A DMA transfer is usually split into several bursts because other operations on the PCI bus may interrupt it and because the destination memory in the computer is paged (4 kB/page under Windows). Each burst generates a complete Local Bus access cycle.

A DMA transfer is always initiated by the host computer. The Local Bus interface supports any sequence of bursts of any size. All designs should do likewise. The target circuitry within the FPGA knows neither the burst size nor the size of the entire DMA transfer. The FPGA must simply respond to requests from the PCI interface until termination of the transfer.

For most operating systems the first burst in a transfer is typically less than 4 kB, because it corresponds to a partial page in the computer. Subsequent bursts usually correspond to the page size of 4 kB and the last burst may be smaller again.

The software that initiates the DMA transfer must know the number of data to be transferred. If the quantity of data to be read is variable, there must be a mechanism for the software to be told the actual number of available data. The simplest solution is to count them within the FPGA and store the value in a register so that the software can read it prior to initializing the DMA transfer.

4.4 Local Bus

The Local Bus connects the FPGA to the PCI interface. The firmware designer does not need to understand the Local Bus in detail, but its signals are visible in the simulation. Thus, a short explanation is given here.

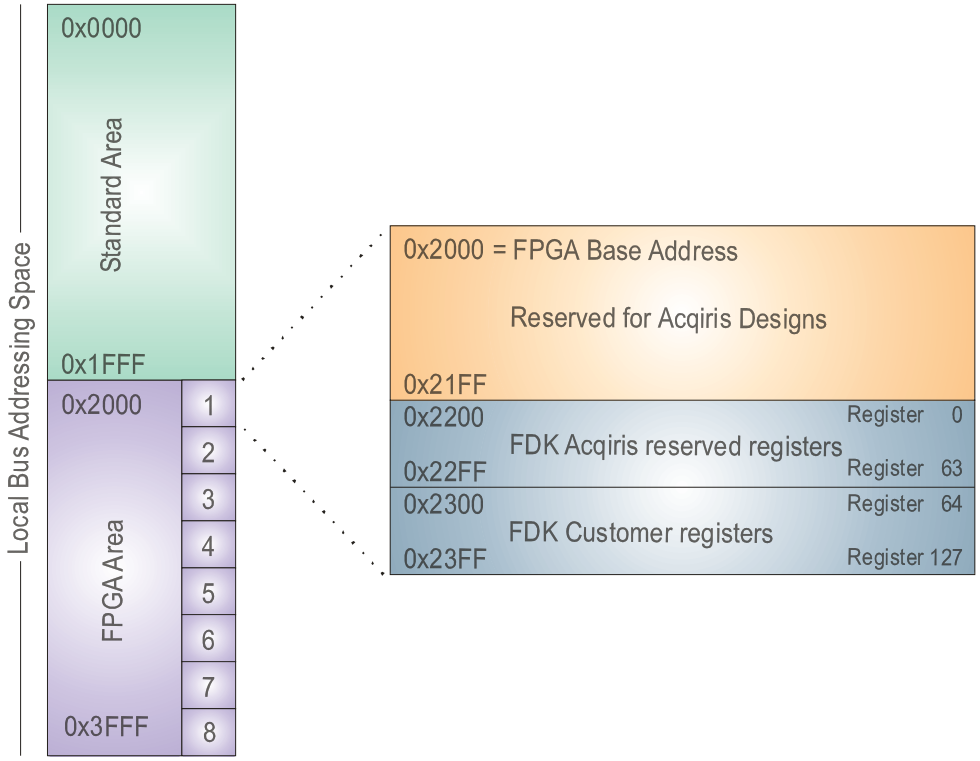
4.4.1 Local Bus Address

On the AC/SCxxx board, all resources are accessible from a host computer through the PCI bus. The control CPLD translates the PCI requests into Local Bus requests. Therefore, all resources of the board are connected on the Local Bus which has a 14-bit address bus and provides a 16 KB address space. Note that all addresses are in bytes, but the Local Bus only deals with 32-bit words, therefore the two LSB's of the address is always 0.

The Local Bus address space is divided in two main areas which are shown in the following table.

Addr Low	Addr High	Size	Purpose
0x0000	0x1FFF	8K x 8 bits	Standard area for all resources on the base board (without the mezzanines)
0x2000	0x3FFF	8K x 8 bits	Reserved area for FPGAs. Divided in eight areas to handle up to eight FPGA. Each FPGA occupies a space of 1K x 8 bits

The processing FPGA of the AC/SCxxx occupies the first FPGA area defined in the range from 0x2000 to 0x23FF. Half this range is reserved for Agilent Acqiris designs; the second half contains the FDK registers. There are 128 32-bit FDK registers. The user application communicates with the FPGA through its registers by using the Agilent Acqiris-supplied API-function **Acqrs_logicDeviceIO**. Although the customer has r/w access to all registers, the definition of the registers 0 to 63 is reserved for Agilent Acqiris. The definition of the remaining 64 registers is entirely open for use by the firmware developer.



Local Bus Address	Target
0x2200 to 0x23FC, step 0x4	Processing FPGA Register 0 to 127, each 32 bits

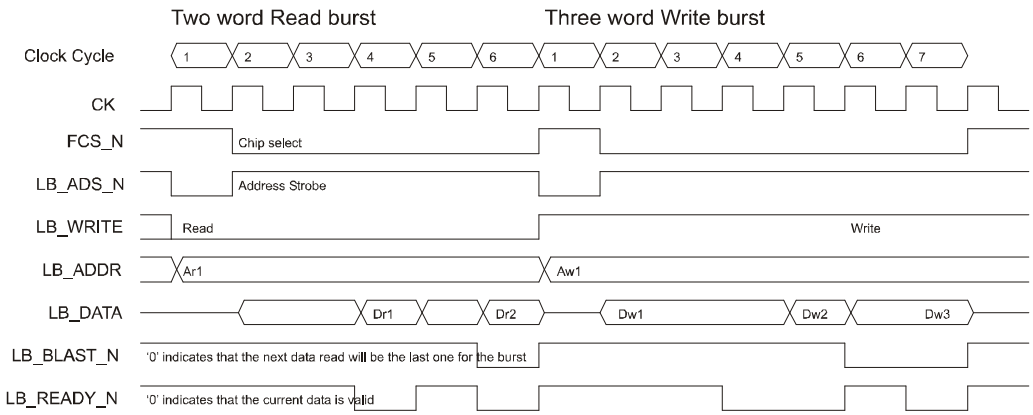
As noted earlier, the CPLD translates the PCI requests into Local Bus requests. The Local Bus requests are translated within the FPGA to Internal Bus requests. This last translation is done by the core **`lb_interface`**. Accessing the FDK registers leads to request on the Internal Bus.

4.4.2 Local Bus Signals

Signal	Short Description
LB_LHOLD	Request from the PCI interface to access the Local Bus
LB_LHOLDA	Grant to the PCI interface to access the Local Bus
LB_WRITE	Local Bus access direction
LB_ADS_N	Local Bus address strobe
LB_ADDR	Local Bus address, 14 bits, bi-directional
LB_DATA	Local Bus data, 32 bits, bi-directional
LB_READY_N	Local Bus ready
LB_BLAST_N	Local Bus flag for last transfer of a burst
LB_BREQ	Local Bus request
LB_EOT_N	Abort DMA (not supported)
FCS_N	FPGA Chip select from the PCI interface
FREADY_N	Ready to the PCI interface (present two cycles prior to LB_READY_N)
FREAD	Local Bus access direction from the PCI interface

4.4.3 Local Bus Timing

Each Local Bus transaction starts with the address strobe **LB_ADS_N** and ends with **LB_BLAST_N** and **LB_READY_N** simultaneously low (both active '0'). Once **LB_BLAST_N** is '0', it remains '0' until the first occurrence **LB_READY_N**. If the transaction is a single word read or write, **LB_BLAST_N** will already be active in cycle 2.



4.5 Internal Bus

The Local Bus protocol is not well adapted for an implementation inside the FPGA. Therefore Agilent Acqiris has defined an Internal Bus (IB-BUS). It handles single and burst transfers. The **lb_interface** core connects the Local Bus to the Internal Bus (see description in section 6.7 ***LOCAL BUS INTERFACE***). The Internal Bus protocol is described there.

An access starts with one of the 3 selection signals rising to '1' (**IB_Customer, IB_Acqiris,** or **IB_Cpld**). A firmware designer only deals with the Customer Space. The selected target then replies with **IB_Rdy** until the access ends when the selection signal falls to '0'. If a target needs additional time at the end of the access before it can handle another one, the signal **IB_End** should be driven to '0', forcing the next access cycle to wait until the target sets the signal **IB_End** back to '1'.

The IB-BUS has two 32-bit data busses: **IB_DataW** to write to internal registers and **IB_DataR** to read from internal registers.

Read data multiplexing to the **IB_DataR** bus is implemented with a simple OR. This works because any target should set its **IB_DataR** output to '0' when it is not selected. The same rule applies for Ready multiplexing to **IB_Rdy**. It can also be used for **IB_End** with a simple AND function because the active state is negative.

When the access is a Burst Indirect Access (length of 1 is still possible), the state of **IB_Direct** remains '0' during the entire access. Otherwise, in case of direct accesses, it is driven high.

If the access addresses an unimplemented target, the Local Bus interface will generate a timeout by setting **IB_TimeO** to '1' until the access cycle ends. After this **IB_TimeO** is set back to '0'.

4.5.1 Extension of the Internal Bus Addressing Space

Although the Local Bus protocol supports burst transfers on any address, the core **lb_interface** supports burst transfers only on a single address. This address corresponds to the first FDK register at 0x2200 (or register 0) and is called the **Indirect Data Port**. All other registers are called the **Direct Access Registers**.

The **Indirect Data Port** must be used, in any case, when reading buffers. It operates with two other registers in order to implement indirect addressing. These are the **Buffer Identifier Register** and the **Indirect Address Register**. The **Buffer Identifier Register** extends the capability of the **Indirect Data Port** to address up to 2x128 different buffers (128 for the Customer, 128 for Agilent Acqiris). The **Indirect Address Register** extends the per buffer address space to 2^{32} addresses. Burst accesses through the **Indirect Data Port** reach a transfer rate up to 132 MB/s.

There are 64 Direct Access Registers reserved for the Customer and 64 for Agilent Acqiris. This is not enough for many designs so we have added a simplified indirect addressing capability for direct accesses. The **Direct Access Registers** can be made to use the **Direct Access Block Register,** an additional address register, if the developer desires. The **Direct Access Block Register** extends the number of possible

Direct Access Registers to 256 blocks of 64 registers for both the Customer FDK registers and the Agilent Acqiris reserved FDK registers.

For a description of the signals IB_xxx mentioned in the table below, please read the next section.

Register	Access	Description
AGILENT RESERVED, customer has r/w access, but developer cannot define any bit in this space		
0	R, W	**Indirect Data Port:** During Indirect (Burst) Access, data are read from, or written to, this register. The signal **IB_Addr** gets the value of the Indirect Address Register (see below).
1	R,W	**Indirect Address Register:** Before an Indirect (Burst) transfer, this register is loaded with the start address from which to read (or to which to write).The signal **IB_Addr** takes the value of this register when accessing the **Indirect Data Port**. This address is defined in bytes and auto-incremented by 4 for each 32-bit word that is read or written.
2	R, W	**Buffer Identifier Register:** This register is intended to distinguish between different data buffers. Its value is never auto incremented. The signal **IB_IndirCtr** takes the value of this register for all accesses to the Internal Bus. Values 0 to 0x7F are reserved for Agilent Acqiris. Values 0x80 to 0xFF are free to be used by firmware developers.
3-8	R, W	Reserved for Agilent Acqiris use. Customers shall not define any bits at these locations.
9		**Direct Access Block register:** This register is intended to distinguish between different blocks of Direct Access Registers. Its value is never auto incremented. The bits 23..16 of the signal **IB_Addr** takes the value of this register for all Direct Accesses to the Internal Bus. Simultaneously, the bits 13..0 of the signal **IB_Addr** takes the value of the bits 13..0 of the Local Bus address **LB_Addr** (The other bits of **IB_Addr** remain '0').
10-63	R,W	Reserved for Agilent Acqiris use. Customers shall not define any bits at these locations.
OPEN REGISTERS, customer has r/w access, all bits are defined by the firmware developer		
64-127	R, W	Block of 64 registers of 32 bits each. Usage is totally free for Customers.

4.5.2 List of Internal Bus Signals

Signal	Type	Short Description
IB_Customer	Out	Customer space selection signal: 0 = not in the User Core address space 1 = within the User Core address space
IB_Cpld	Out	Agilent Acqiris reserved usage, select CPLD addressing space
IB_Acqiris	Out	Agilent Acqiris reserved usage, select Agilent Acqiris addressing space
IB_Dirsel	Out	Addressing type of current access: 0 = Indirect, 1 = Direct
IB_IndirCtr	Out	8-bit Buffer Identifier number. It takes the value of the **Buffer Identifier register** for accesses to the **Indirect Data Port**. 0x0 to 0x7F are reserved for Agilent Acqiris. 0x80 to 0xFF are available to the firmware developer.
IB_Write	Out	Access direction: 1 = write (data from the driver), 0 = read (data to the driver)
IB_Addr	Out	32-bit address. It takes the value of the **Indirect Address Register** for the accesses to the **Indirect Data Port** or the value of the Local Bus address for the accesses to the **Direct Access registers**.
IB_DataW	Out	32-bit write Data bus
IB_DataR	In	32-bit read Data bus All unselected devices connected to the IB bus shall drive 0x00000000 on

Signal	Type	Short Description
		IB_DataR to implement a data read multiplexer with a simple OR function.
IB_Rdy	In	The selected device shall drive '0' when the data (read) or the device (write) is not ready or not selected. The selected device shall drive '1' when the data (read) or the device (write) is ready and selected. All unselected devices connected to theIB bus shall drive '0' on this line to implement a ready selection multiplexer with a simple OR function.
IB_TimeO	Out	Set to '1' when a device failed to acknowledge **IB_Rdy** within 2.0 us after an IB cycle has been started. The cycle will end without retry. **IB_TimeO** will then come back to '0'.
IB_Valid	Out	**IB_Valid** is '1' when **IB_DataW** is valid. In case of a burst write, it should be used to load the data to the selected device. For a burst write access **IB_Valid** can be '1' during multiple clock periods, denoting successive writings.
IB_End	In	When set to '1', allows the Local Bus interface to execute an access to the Internal Bus. It is normally driven '1' except in the case of an Indirect Access. For Indirect Accesses, the selected device must drive '0' on this line for the entire time of the access. **IB_End** must be set to '1' when the device has completed the access and when it is ready for another one.

4.5.3 Internal Bus Address for the Direct Access Registers

Any transaction on the Internal Bus starts with the signal **IB_Customer** rising to '1'. The transaction ends with the signal **IB_Customer** falling back to '0'. **IB_Customer** is set to '1' for each Internal Bus access to the FDK registers. The signal **IB_Customer** and the Internal Bus address **IB_Addr** must be used to validate the access to the Direct Access Registers.

While **IB_Customer** is '1', the Internal Bus address **IB_Addr** takes the following value:

31-24	23-16	15-14	13-2	1-0
00000000	DIR_BLOCK_NBR(7..0)	00	LB_ADDR(13..2)	00

[13-2]	LB_ADDR	The register address within the register block. It is the value of the bits 13 to 2 of the Local Bus address.
[23-16]	DIR_BLOCK_NBR	It is the **Direct Access Block** number. It is equal to the value of the bits 7 to 0 of the **Direct Access Block Register** (It is '0' by default).

The value of the Signal **IB_indirCtr** is not relevant and must not be used to validate accesses to the **Direct Access Registers**.

4.5.4 Internal Bus Address for the Indirect Data Port

Any transaction on the Internal Bus starts with the signal **IB_Customer** rising to '1'. The transaction ends with the signal **IB_Customer** falling back to '0'. **IB_Customer** is set to '1' for each Internal Bus access to the **Data Port**. The signal **IB_Customer** and the Internal Bus buffer identifier signal **IB_IndirCtr** must be used to validate the access to a buffer. The Internal Bus address **IB_Addr** can be used to determine the address within the buffer at which the transfer will begin.

While **IB_Customer** is '1', the Internal Bus address **IB_Addr** takes the following value:

31..0
IND_ADDR

[31-0]	IND_ADDR	The value of the **Indirect Address register**

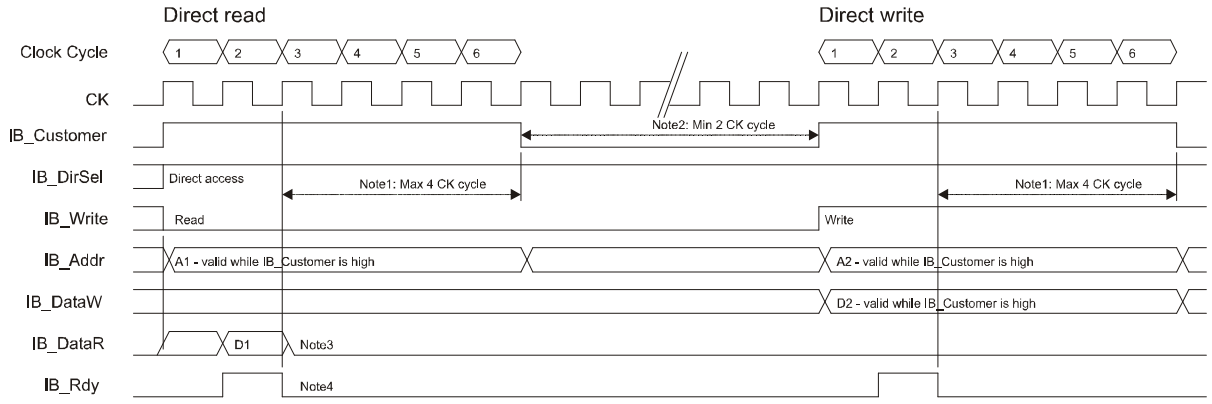
While **IB_Customer** is '1', the internal buffer identifier **IB_IndirCtr** takes the following value:

31..8	7..0
0x000000	BUF_ID_REG

[7-0] BUF_ID_REG The bits 7 to 0 of the **Buffer Identifier Register**

4.5.5 Direct Access

Direct Access is commonly used to configure a control register or read a status register. It is not intended for reading or writing large amounts of data. Burst transfer is not available for Direct Access mode.



Note1: This value is actually fixed to 4 CK cycles. Any target should support a minimum of 2 CK cycles for future compatibility.

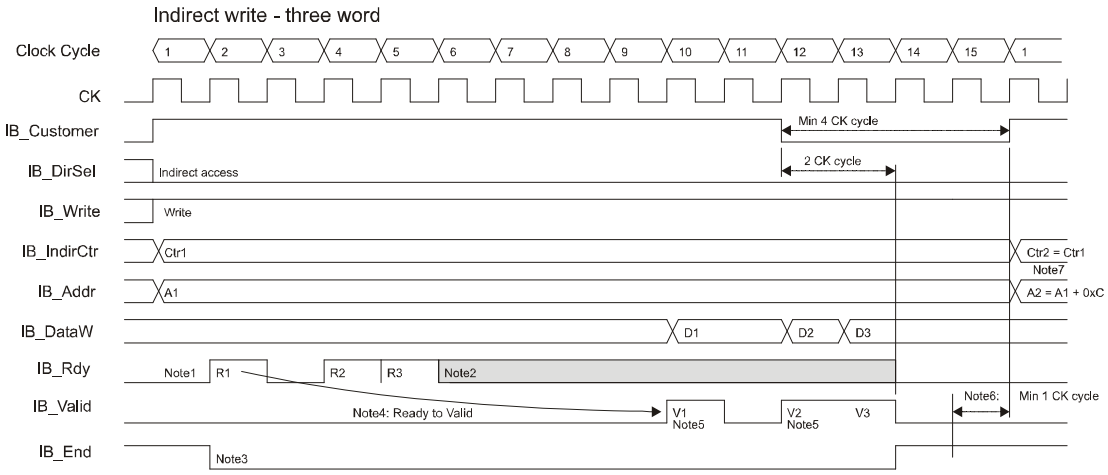
Note2: Although the current minimum is 9, any target should support a minimum separation of 2 CK cycles for future compatibility.

Note3: **`IB_DataR`** must be valid for readout when **`IB_Rdy`** is set to '1'. Otherwise **`IB_DataR`** should be set to '0'.

Note4: It is possible to assert **`IB_Rdy`** already in the first clock cycle. A target should set **`IB_Rdy`** only when it is the source or destination of the current transfer.

4.5.6 Indirect Access Write

Indirect Access write operations may or may not be executed as burst transfers, i.e. as sustained data transmissions without any dead time. The decision on whether the transfer is a burst depends on the driver and/or the PCI-interface. It cannot be influenced by the firmware developer. Therefore all designs must support burst transfers. As long as Write DMA is not supported by the driver, all bursts (which are initiated by the PCI interface) will be of length 1.



The delay Ready to Valid is due to the backward and forward pipelines within the Local Bus interface.

Note1: It is possible to set **IB_Rdy** to '1' already in the first clock cycle. A target should set **IB_Rdy** to '1' only if it is the source or destination of the current transfer.

Note2: As the target does not know the size of the burst, it should set **IB_Rdy** to '1' as long as it is ready to receive data and as long as **IB_Customer** remains '1'. **IB_Rdy** shall not be set to '1' for more than two cycles after **IB_Customer** has gone to '0' (in the example above, it should not be set after cycle 13).

Note3: It is possible to set **IB_End** to '0' already in the first clock cycle. A target should set **IB_End** to '0' only if it is the source or destination of the current transfer. The target should set **IB_End** to '0' until it is ready to start a new burst cycle. A target shall not set **IB_End** back to '1' before the third cycle after **IB_Customer** is set to '0' (in the example above, it should not be assigned before cycle 14).

Note4: **IB_Valid** is the acknowledgement from the core **lb_interface** to the target after having set **IB_Rdy** to '1'. The data on **IB_DataW** are valid while **IB_Valid** is '1'. The latency from **IB_Rdy** to **IB_Valid** is currently 8 but could change in the future, so any target should wait for **IB_Valid**.

Note5: **IB_DataW** is valid while **IB_Valid** is '1'. The last **IB_Valid** is always two cycles after **IB_Customer** is set to '0'.

Note6: A new burst cycle will start, at the earliest, two clock cycles after **IB_End** has been set to '1'.

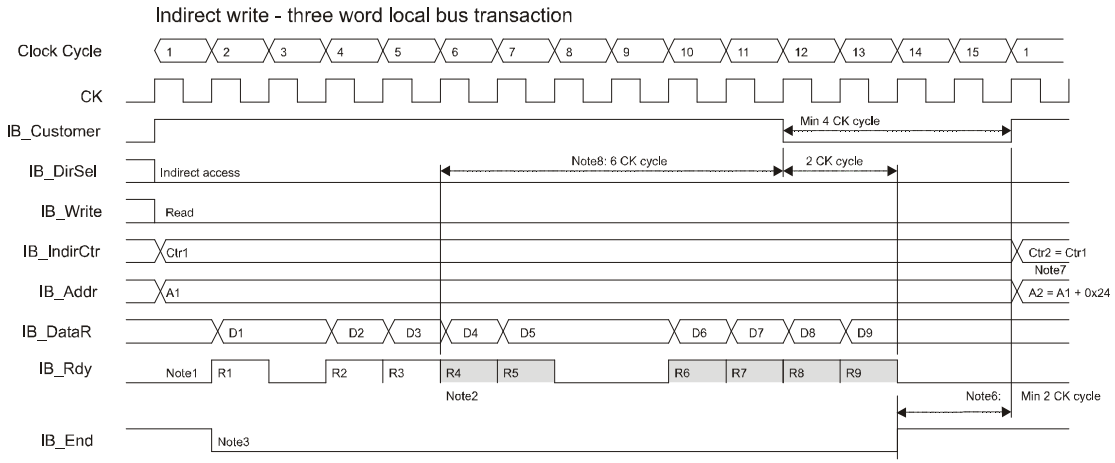
Note7: If Ctr2 is equal to Ctr1 and if both accesses are write–indirect, then the address is A2 = A1 + 0x4 * (the number of previously read data).

4.5.7 Indirect Access Read

Indirect Access read operations may or may not be executed as burst transfers, i.e. as sustained data transmissions without any dead time. The decision on whether the transfer is a burst depends on the driver and/or the PCI-interface. It cannot be influenced by the firmware developer. Therefore all designs must support burst transfers. If the driver uses DMA, the bursts may be of arbitrary length. If the data transfer is not DMA, the bursts are of length 1.

Normally, the driver uses DMA for large data transfers. The programmer can forbid the use of DMA and force the use of single-word transactions on the PCI bus (and consequently on the Local and Internal Busses) by setting the option "DMA=0" when initializing the analyzer board with the function **Acqrs_InitWithOptions**. The non-DMA option may be useful as a diagnostic tool.

A target should not stop sending data before the target selection signal is back to ‘0’.



Note1: It is possible to set **IB_Rdy** to ‘1’ already in the first clock cycle. A target should set **IB_Rdy** to ‘1’ only if it is the source or destination of the current transfer.

Note2: As the target does not know the size of the burst, it should set **IB_Rdy** to ‘1’ as long as it is ready to send data and as long as **IB_Customer** remains ‘1’. **IB_Rdy** shall not be set to ‘1’ more than two cycles after **IB_Customer** has gone to ‘0’ (in the example above, it should not be assigned after the cycle 13). In the example, the Data D1 to D3 are effectively read out to the PCI bus. The remaining D4 to D9 will be read to the PCI without any further transaction on the FPGA Internal Bus. At the next burst, the target should send first the Data D10 and then continue.

Note3: It is possible to set **IB_End** to ‘0’ already in the first clock cycle. A target should set **IB_End** to ‘0’ only if it is the source or destination of the current transfer. The target should keep **IB_End** at ‘0’ until it is ready to start a new burst cycle. A target shall not set **IB_End** back to ‘1’ before the third cycle after **IB_Customer** is set to ‘0’ (in the example above, it should not be assigned before cycle 14).

Note4: **IB_Valid** is the acknowledgement from the core **lb_interface** to the target that has set **IB_Rdy** ‘1’. The Data on **IB_DataW** are valid while **IB_Valid** is ’1’. The latency **IB_Rdy** to **IB_Valid** is currently 8 but could change in the future, so any target should wait for **IB_Valid**.

Note5: **IB_DataW** is valid while **IB_Valid** is ‘1’. The last **IB_Valid** is always two cycles after **IB_Customer** is set to ‘0’.

Note6: A new burst cycle will start at least two clock cycles after **IB_End** has been set to ‘1’.

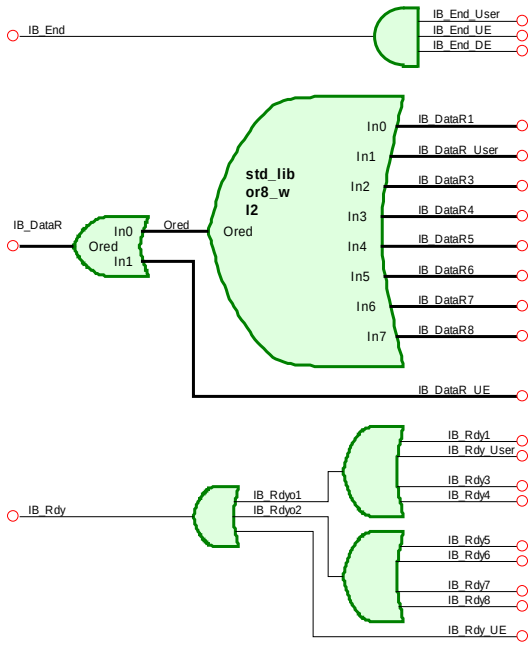
Note7: If Ctr2 is equal to Ctr1 and if both accesses are read–indirect, then the address is A2 = A1 + 0x4 * (the number of previously read data, in this case 9).

4.5.8 Multi Target Connection

A single target can be connected directly to the Internal Bus port (all IB_xxx signals). A single target could decode multiple direct and/or indirect addresses.

When connecting multiple targets, all of them need to drive the signals IB_Rdy, IB_DataR, and IB_End back to the lb_interface. These lines must be set to a specific state (low for IB_Rdy and IB_DataR, high for IB_End) when the target is not connected, permitting the use of a simple OR or AND as a multiplexer.

The following example shows these connections in case of 8 connected targets, 2 of them using indirect addressing:



IB_Rdy: The signal from all targets should be OR'd.

IB_End: The signal from all targets should be AND'd.

IB_DataR: The signals from all targets should be bit-wise OR'd.

5. Base Design

The Agilent Acqiris-supplied Base Designs are simple but complete designs for use as:

- A starting point for the development of new firmware
- A test application to verify the behavior of the AC2x0/SC2x0 hardware and/or the Agilent Acqiris-supplied cores.

The base designs contain most of the available cores.

5.1 Multiple Base Designs

There are no specific base designs for the SC210. You should use the base design for the SC240, i.e. use **`sc240_top_sysclk_str1`**.

The four Base Designs in the developer's library are examples of complete, functional firmware including data acquisition, internal buffering, and readout capabilities. The implemented cores offer a necessary set of functions, in order to minimize the work needed to start a new design. They include several registers, the DE-Buffer, and the DE-Monitor. The DE-Buffer is implemented in the block **`de_interface`** and is always available. It can be read to monitor the input data stream. The DE-Monitor buffer is implemented in the component **`user_block_example`** and can also be read to the host computer, to monitor the data stream within the FPGA.

Model	BaseDesign	Comment
AC240	**`ac240_top_sysclk`**	Two channel Base Design without interface to the external memory
AC240 + mem option	**`ac240_top_sysclk_ddr`**	Two channel Base Design with interface to the external memory
AC210	**`ac210_top_sysclk`**	Single channel Base Design without interface to the external memory
SC240	**`sc240_top_sysclk_str1`**	Two channel Base Design with interface to the external optical link

Each Base Design is accompanied by a Test bench component

Model	Test bench	Comment
AC240	**`ac240_top_sysclk_tb`**	Test bench component for the two channel Base Design without interface to the external memory
AC240 + mem option	**`ac240_top_sysclk_ddr_tb`**	Test bench component for the two channel Base Design with interface to the external memory
AC210	**`ac210_top_sysclk_tb`**	Test bench component for the single channel Base Design without interface to the external memory
SC240	**`sc240_top_sysclk_str1_tb`**	Test bench component for the two channel Base Design with interface to the external optical link

5.2 Bitfile name for the Base Design

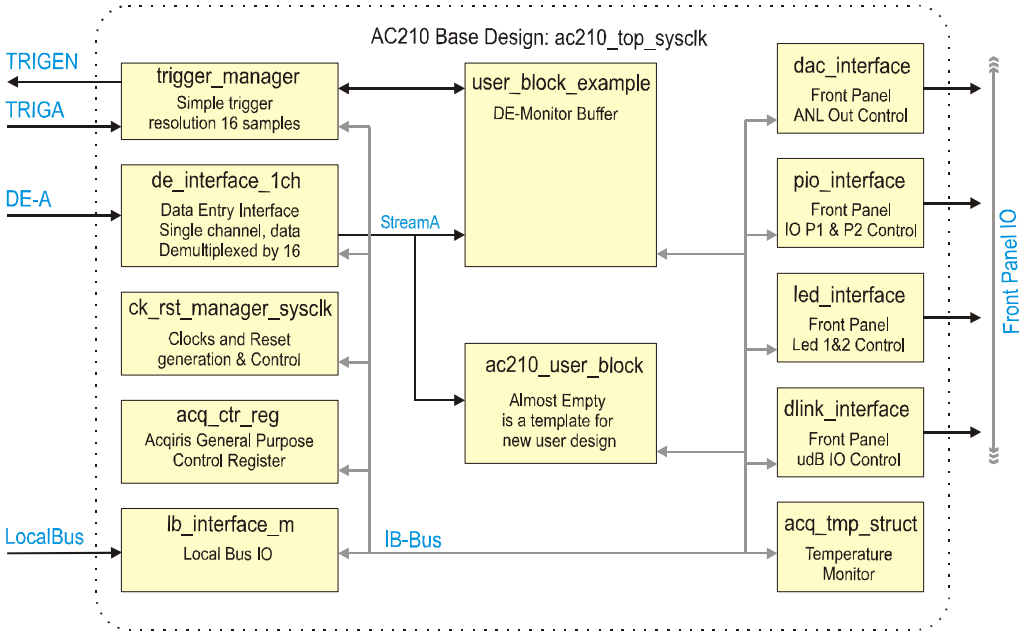
Each base design is completely implemented. The corresponding bitfile could be exercised with the program AcqirisAnalyzer. The cosrrespondance between the base design component and the bitfile is given in this table:

Model	BaseDesign	Bitfile
ac240	**`ac240_top_sysclk`**	ac240.bit
AC240 + mem option	**`ac240_top_sysclk_ddr`**	ac240mem.bit
AC210	**`ac210_top_sysclk`**	ac210.bit
SC240	**`sc240_top_sysclk_str1`**	sc240str1.bit

5.3 Overview of the Base Designs

5.3.1 AC210 Base Design

The base test **`ac210_top_sysclk`** is delivered as an example of buffering the acquisition data to a buffer within the FPGA: the **`DE-Monitor`** buffer. This buffer has the ability to store incoming data, either asynchronously or synchronized with the trigger. It is instantiated within the block **`user_block_example`**. The bloc ac210_user_block is a template for developers to insert custom designs.



5.3.2 Trigger accuracy versus Sampling Rate

This base design implements the simple trigger block **`trigger_manager`** which has an accuracy directly related to the sampling frequency.

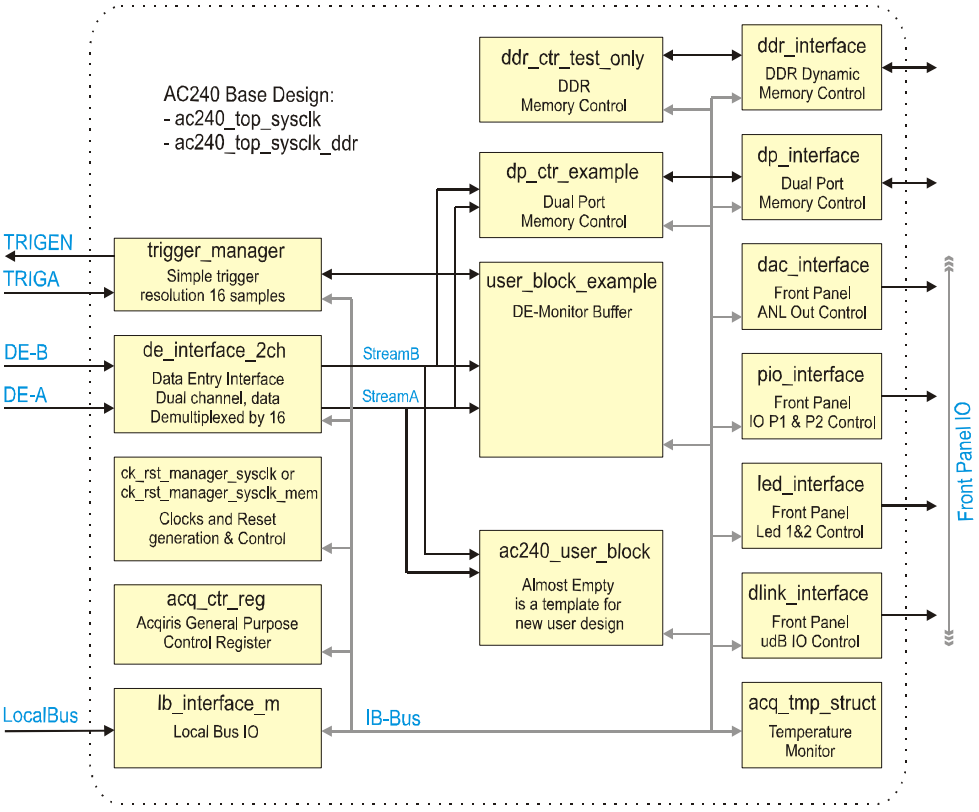
Module	Trigger Accuracy
AC210	Equal to the sampling period multiplied by 16 (16ns at 1 GS/s)

5.4 AC240 Base Design

There are two base test examples for the ac240. The base test **`ac240_top_sysclk`** is delivered as an example of buffering the acquisition data to a buffer within the FPGA: the **`DE-Monitor`** buffer. This buffer can be read by program. The base test **`ac240_top_sysclk_mem`** is identical and contains additional blocks to control and verify the external optional memories.

5.4.1 Architecture

The drawing below shows the data flow inside the ac240 Base firmware. The converted data from the ADCs are transmitted as data flows DE-A and DE-B to the data processing FPGA where they are received by a data entry interface. The data are channeled through the two internal data streamsA and B to the block **`user_block_example`** and **`dp_ctr_example`**. The **`DE-Monitor`** lies within the block **`user_block_example`**. Monitoring the data stream could be immediate or depending on the arrival of a trigger, with a trigger precision of 16 ns.



5.4.2 Trigger accuracy versus Sampling Rate

This base design implements the simple trigger block **`trigger_manager`**. The trigger accuracy depends on the sampling rate and on the fact that the channels could be interleaved or not.

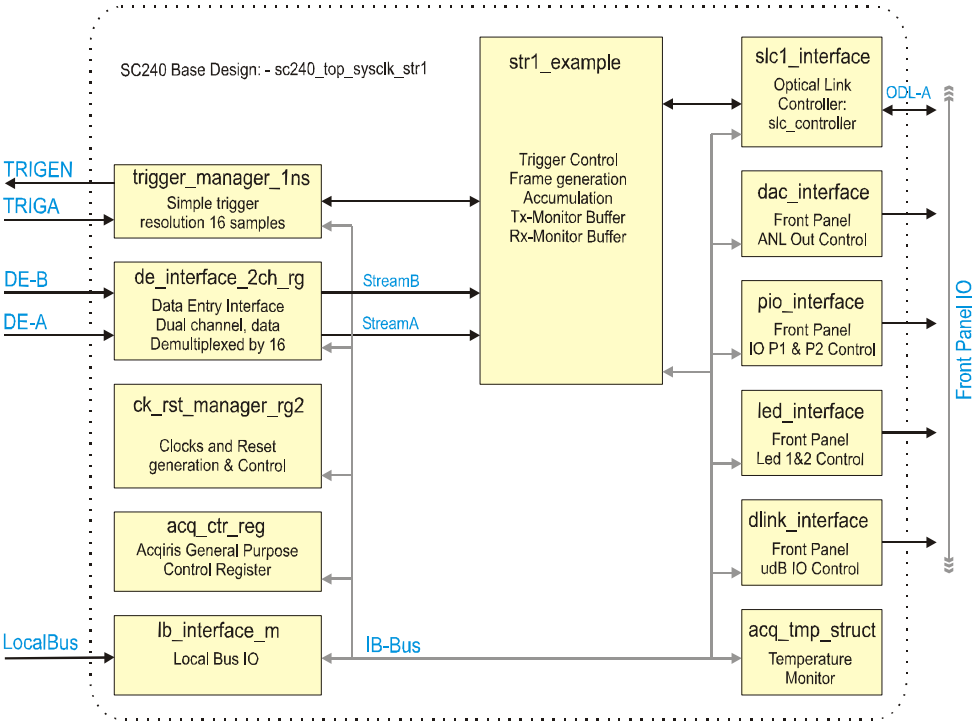
Module	Two channel mode	Single channel mode
AC240	Equal to the sampling period multiplied by 16 (16ns at 1 GS/s)	Equal to the sampling period multiplied by 32 (16 ns at 2 GS/s)

5.5 SC240 Base Design

The base streamer example **`sc240_top_sysclk_str1`** is delivered as an example of streaming data to the front panel optical link by the use of the serial front panel data port interface.

5.5.1 Architecture of the Base Streaming Firmware

The drawing below shows the data flow inside the Base Streaming firmware. The converted data from the ADCs are transmitted as data flow DE-A and DE-B to the data processing FPGA where they are received by a data entry interface (includes the DE-Buffer). The data are channeled through the two internal data streams, A and B, to the streamer example block.



The streamer example block generates, manages, and sends three differents type of frames to the sfpdf controller **`slc_controller`**: the raw data frame, the accumulated data frame, and the parameter data frame.

The block **`str1_example`** and its functions are described with more details in paragraph **6.23**. The block **`scl1_interface`** is a simple encapsulation of the core **`slc_controller`** and is not described in details. The core **`slc_controller`** is described in details in the paragraph **6.20**.

The TX-Monitor Buffer is a spy of the transmitted frames (TX-Frame). It can be read by the user program.

For verification, the TX output could be looped back to the RX input and the received data could be monitored with the RX-Monitor Buffer and readout by the user program.

5.5.2 Trigger Positioning Resolution versus Sampling Rate

There are two trigger modes that could be set by program, the standard trigger and the high resolution trigger. While the standard trigger could be used with any sampling rate, the high resolution trigger should be enabled only for the following sampling rates:

Module	Two channel mode		Single channel mode	
SC240	500 MS/s	1 GS/s	1 GS/s	2 GS/s
	2ns	1ns	2ns	1ns
SC210	Not available		500 MS/s	1 GS/s
			2ns	1ns

Table 5-1 : Valid sampling rates and resolution of the High resolution trigger

The resolution of the standard trigger is 16 samples in non-interleaved operation and 32 samples when the two channel of the SC240 are interleaved.

5.5.3 Trigger Time Stamp

The trigger Time Stamp is only valid for the high resolution trigger. The Time Stamp resolution is identical to the trigger positioning resolution.

5.5.4 Front Panel LED Status

Item	Comments
L1	Link 0 Status : • Red: ODL faulty. • Green: ODL successfully initialized and active in TX Mode. • Orange: ODL successfully initialized.
L2	Acquisition Status : • Orange: transfer disabled • Green: Transfer & Trigger Enabled • Red: Transfer Enabled & Trigger received

5.6 List of Cores Instantiated in Base Designs

Core	Base Design				Library	Description
	ac210_top_sysclk	ac240_top_sysclk	ac240_top_sysclk_ddr	sc240_top_sysclk_str1		
ac210_user_block	☑				ac240_developer_lib	Main block for AC210
ac240_user_block		☑	☑		ac240_developer_lib	Main block for AC240
acq_ctr_reg	☑	☑	☑	☑	ac240_fdk	Main Acqiris control register
acq_tmp_struct	☑	☑	☑	☑	ac240_fdk	Interface to the on-chip FPGA temperature measurement
ck_rst_manager_sysclk	☑	☑			ac240_fdk	Clock and reset management for designs without external memory
ck_rst_manager_sysclk_mem			☑		ac240_fdk	Clock and reset management for designs with external memory
ck_rst_manager_rg2				☑	ac240_fdk	Clock and reset management for designs with Optical Data Link and high resolution trigger
dac_interface	☑	☑	☑	☑	ac240_fdk	Interface to the DAC driving the front-panel analog output
ddr_ctr_test_only			☑		ac240_fdk	Example of interfacing the user port of the core ddr_interface
ddr_interface			☑		ac240_fdk	Interface to the DRAM memory banks
ddr_interface_buffer			☑		ac240_fdk	Block handling the Xilinx IO primitive to the DRAM memory
de_interface_1ch	☑				fdk_lib	Acquisition data stream interface for single channel module
de_interface_2ch		☑	☑		fdk_lib	Acquisition data stream interface for dual channel module

Core	Base Design				Library	Description
	ac210_top_sysclk	ac240_top_sysclk	ac240_top_sysclk_ddr	sc240_top_sysclk_str1		
de_interface_2ch_rg				☑	fdk_lib	Acquisition data stream interface for dual channel module with Optical Data Link and high resolution trigger
de_chip_io	☑	☑	☑	☑	fdk_lib	Block handling the Xilinx IO primitive to the data input stream bus
dlink_interface	☑	☑	☑	☑	ac240_fdk	Multi-line digital front panel IO interface
dp_ctr_example			☑		ac240_fdk	Example of interfacing the user port of the core dp_interface
dp_interface			☑		ac240_fdk	Interface to the dual port SRAM memory
dp_interface_io			☑		ac240_fdk	Block handling the Xilinx IO primitive to the SRAM memory
lb_interface_m	☑	☑	☑	☑	fdk_lib	Interface to the Local Bus. The user program communicates with the FPGA through this interface.
lb_interface_io	☑	☑	☑	☑	fdk_lib	Block handling the Xilinx IO primitive to the Local Bus
led_interface	☑	☑	☑	☑	ac240_fdk	Front panel LED control
pio_interface	☑	☑	☑	☑	ac240_fdk	Interface to the MMCX digital IO on the front-panel
slc1_interface				☑	ac240_fdk	Single interface to the optical link
str1_example				☑	ac240_fdk	Simple streaming example
trigger_manager	☑	☑	☑		ac240_fdk	Trigger control
trigger_manager_1ns				☑	ac240_fdk	High resolultion trigger control
user_block_example	☑	☑	☑		ac240_fdk	Example of managing the data stream

5.7 Register List in Base Designs

Register Number	Register Address	Access Right	Available in component	Base Design				Comment
				ac210_top_sysclk	ac240_top_sysclk	ac240_top_sysclk_ddr	sc240_top_sysclk_str1	
Customer Register Space – Reserved for Definition by Agilent								
0	0x2200	RW	--	☑	☑	☑	☑	Indirect Data Port
1	0x2204	RW	lb_interface	☑	☑	☑	☑	FPGA Indirect Address
2	0x2208	RW	lb_interface	☑	☑	☑	☑	FPGA Buffer Identifier
3	0x220C	RW	acq_ctr_reg	☑	☑	☑	☑	FPGA Main Control
4	0x2210	R	lb_interface	☑	☑	☑	☑	FPGA Code Protection

5	0x2214							Reserved
6	0x2218	R	acq_ctr_reg	☑	☑	☑	☑	FPGA Main Status
7	0x221C							FPGA Temperature
8	0x2220	RW	de_interface_1ch de_interface_2ch	☑	☑	☑	☑	FPGA DE-Bus Control
9	0x2224	RW	lb_interface	☑	☑	☑	☑	FPGA Direct Access Block
10-31	--							Reserved
32	0x2280	RW	pio_interface	☑	☑	☑	☑	Front Panel PIO Control
33	0x2284	RW	dac_interface	☑	☑	☑	☑	Front Panel DAC Control
34	0x2288	RW	led_interface	☑	☑	☑	☑	Front Panel LED Control
35	0x228C							Reserved
36	0x2290	RW	dlink_interface	☑	☑	☑	☑	Front Panel µDB-IO Control
37	0x2294	RW	dlink_interface	☑	☑	☑	☑	Front Panel µDB-IO Output
38	0x2298	R	dlink_interface	☑	☑	☑	☑	Front Panel µDB-IO Input
39	0x229C	RW	dp_interface			☑		Dual Port Memory Control
40	0x22A0	RW	dp_interface			☑		Dual Port Memory Test Pattern
41	0x22A4	R	dp_interface			☑		Dual Port Memory Status
42	0x22A8	R	dp_interface					Dual Port Memory Test Value
43	0x22AC	R	dp_interface			☑		Dual Port Memory Test Result
44	0x22B0	RW	ddr_interface			☑		DDR A Control / Status
45	0x22B4	RW	ddr_interface			☑		DDR A Self-test Control / Status
46	0x22B8	R	ddr_interface			☑		DDR A Self-test Status 1
47	0x22BC	R	ddr_interface			☑		DDR A Self-test Status 2
48	0x22C0	R	ddr_interface			☑		DDR A Self-test Status 3
49	0x22C4	R	ddr_interface					DDR A Self-test Status 4
50	0x22C8	R	ddr_interface			☑		DDR A Self-test Error Count
51	0x22CC					☑		Reserved
52	0x22D0	RW	ddr_interface			☑		DDR B Control / Status
53	0x22D4	RW	ddr_interface			☑		DDR B Self-test Control / Status
54	0x22D8	R	ddr_interface			☑		DDR B Self-test Status 1
55	0x22DC	R	ddr_interface			☑		DDR B Self-test Status 2
56	0x22E0	R	ddr_interface					DDR B Self-test Status 3
57	0x22E4	R	ddr_interface			☑		DDR B Self-test Status 4
58	0x22E8	R	ddr_interface			☑		DDR B Self-test Error Count
59	0x22EC					☑		Reserved
60	0x22F0	RW	ddr_interface			☑		DCM Phase Shift Control / Status
61-63								Reserved
Customer Register Space for AC2x0 Base Design								
64	0x2300	RW	user_block_example	☑	☑	☑	☑	Base Design Control

65	0x2304	R	user_block_example	☑	☑	☑	☑	Base Design Status
66	0x2308	RW	dp_ctr_example			☑		Example of SRAM Interfacing
67	0x230C	RW	ddr_ctr_example			☑		Example of DRAM Interfacing
68-127	--	RW	--					Unused
Customer Register Space for SC2x0 Base Design								
64	0x2300	RW	str1_example				☑	Main Control
65	0x2304						☑	Unused
66	0x2308	RW	str1_example				☑	TX Monitor Control and Status
67	0x230C	RW	str1_example				☑	RX Monitor Control and Status
68-72							☑	Unused
73	0x2324	RW	str1_example				☑	Streamer Configuration
74-79	0x2328	RW					☑	Unused
80-82	0x2340 0x2344 0x2348	RW	slc1_interface				☑	SLC Control Link 0 SLC Status Link 0 SLC Signal Link 0
83-127							☑	Unused

Note: The registers number 64 to 127 shall not be used for another purpose if the corresponding cores remain in the design.

5.8 Indirect Addressing in AC2x0 Base Designs

The buffers are accessed through the Indirect Data Port at address 0x2200.

Buffer Identifier	Address Range	Access Right	Available in component	Comment
Customer Register Space – Agilent Reserved				
0x00	0x0 to 0x3FFFFFC	RW	`ddr_interface`	DDR BANK A, 256 MB
0x01	0x0 to 0x3FFFFFC	RW	`ddr_interface`	DDR BANK B, 256 MB
0x04	0x0 to 0xFFFFC	RW	`dp_interface`	Dual Port Memory 1MB
0x08	0x0 to 0x1FFC	RW	`de_interface`	DE-Buffer, 8K samples per channel
0x0C	0x0 to 0x1FFC	RW	`user_block_example`	DE-Monitor, 8K samples per channel
0x00- 0x7F				Reserved for Agilent
Customer Register Space – Customer Reserved				
0x80-0xFF				Reserved for Customer

5.9 Indirect Addressing in SC2x0 Base Designs

The buffers are accessed through the Indirect Data Port at address 0x2200.

Buffer Identifier	Address Range	Access Right	Available in component	Comment
Customer Register Space – Agilent Reserved				
0x08	0x0 to 0x1FFC	RW	`de_interface`	DE-Buffer, 8K samples per channel
0x10	0x0 - 0xFFFC	RW	`str1_example`	TX-Monitor 64K bytes
0x20	0x0 - 0xFFFC	RW	`str1_example`	RX-Monitor 64K bytes

0x00- 0x7F				Reserved for Agilent
Customer Register Space – Customer Reserved				
0x80-0xFF				Reserved for Customer

5.10 Simulation

Complete information about the Acqiris Test Bench environment and the available script commands is available in chapter 7 ***VHDL TEST BENCH***. There is one Test bench for each base design.

The base design executes the script **Control.txt**, which invokes the other files listed in the table below. All scripts associated to a Test bench component are stored within the side data directory of the Test bench. For the single channel Test bench, this will be:

*$AcqirisFdkRoot/lib_projects/ac240_developer_lib/hdlgraphic/***base_design_tb***/struct.bd.info/Sim/**

Script	ac210_top_sysclk_tb	ac240_top_sysclk_tb	ac240_top_sysclk_ddr_tb	Description
Control.txt	☑	☑	☑	Main script, executesother sub-scripts
Acqiris_Cst.txt	☑	☑	☑	Definition of constants for Acqiris_Ctr.txt
Acqiris_Ctr.txt	☑	☑	☑	Test script to verify basic function of the cores that interface to the front-panel I/O
USR_Cst.txt	☑	☑	☑	Definition of constants for USR_Ctr.txt
USR_Ctr.txt	☑	☑	☑	Test script to verify the IN- and DE-Buffers and the access to them
USR_TestDDR.txt			☑	Script to verify read & write access to the dual bank DRAM from the user application
USR_TestDDRU.txt			☑	Script to verify the read & write access to the dual bank DRAM within the FPGA through the user ports
USR_TestDP.txt			☑	Script to verify read & write access to the dual port SRAM from the user application
USR_TestDPU.txt			☑	Script to verify read & write access to the dual port SRAM within the FPGA through the user ports

5.11 Constraints

Two constraint files are supplied. The ".sdc" constraint file is for the Mentor flow with Precision Synthesis, while the ".ucf" constraint file is for all other design flows.

The ".sdc" constraint file is located in the side data directory of each base design:

*…ac240_developer_lib/hdlgraphic/***base_design***/struct.bd.info/Synthesis/Constraints/*

This file handles clock constraints, FPGA IO timing constraints, and pad location constraints. There are also additional location constraints for the DCM and BUFG primitives.

Otherwise notified, the ".ucf" constraint file (to be used for all other design flows) is based on the Xilinx UCF format. This file is generated by Precision Synthesis and is located at:

*…ac240_developer_lib/precision/***base_design_struct/base_design_struct/**

Other constraints, like signaling type and strength, are passed from the VHDL design with attributes.

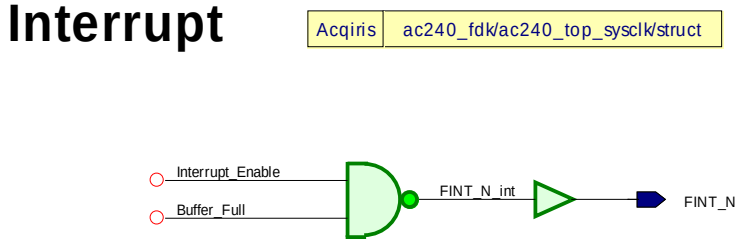
More details about design flow and file locations can be found in the chapters **8 *DESIGN FLOW* and 9 *VHDL LIBRARIES*** of this document.

5.12 Interrupt Control

The FPGA is able to generate a hardware interrupt, **processing interrupt**. It must first be enabled in the FPGA by setting the bit[0], INTE of the control register of the core **acq_ctr_reg** to '1'. The interrupt handling can be used by the software with a call to the function **AcqrsD1_waitForEndOfProcessing**.

The interrupt is managed by the interrupt controller in the PCI interface. The FPGA only has to supply a signal at the end of the processing.

The example below shows the interrupt to be generated after the DE-Monitor has been filled. The PCI interface interrupt controller will detect and memorize the rising edge of the signal *Buffer_Full*. The signal *Buffer_Full* can only generate another interrupt after having gone low again.



5.13 Resource Utilization

There are more details for each core in chapter 9 ***VHDL LIBRARIES***.

The table below shows the resource usage for the base design **`ac240_top_sysclk_ddr`** synthesized with Precision Synthesis and XST. It is compared to the resources that are available in the target Xilinx Virtex II Pro – XC2VP70-6FF1517.

PRECISION SYNTHESIS				**XST**			
Device Utilization Summary				Device Utilization Summary			
Logic Utilization	Used	Available	Utilization	Logic Utilization	Used	Available	Utilization
Number of Slice Flip Flops	10,994	66,176	16%	Number of Slice Flip Flops	11,617	66,176	17%
Number of 4 input LUTs	9,987	66,176	15%	Number of 4 input LUTs	11,387	66,176	17%
Logic Distribution				Logic Distribution			
Number of occupied Slices	10,433	33,088	31%	Number of occupied Slices	11,610	33,088	35%
Total Number 4 input LUTs	10,785	66,176	16%	Total Number 4 input LUTs	12,450	66,176	18%
Number used as logic	9,987			Number used as logic	11,387		
Number used as a route-thru	798			Number used as a route-thru	1,063		
Number of bonded IOBs	765	964	79%	Number of bonded IOBs	765	964	79%
IOB Flip Flops	1,020			IOB Flip Flops	1,059		
IOB Master Pads	10			IOB Master Pads	10		
IOB Slave Pads	10			IOB Slave Pads	10		
IOB Dual-Data Rate Flops	292			IOB Dual-Data Rate Flops	292		
Number of PPC405s	0	2	0%	Number of PPC405s	0	2	0%
Number of Block RAMs	45	328	13%	Number of Block RAMs	45	328	13%
Number of GCLKs	12	16	75%	Number of GCLKs	12	16	75%
Number of DCMs	7	8	87%	Number of DCMs	7	8	87%
Number of GTs	0	20	0%	Number of GTs	0	20	0%

The table below shows the resource usage for the base design **`ac240_top_sysclk`**. It is compared to the resources that are available in the target Xilinx Virtex II Pro – XC2VP70-6FF1517.

PRECISION SYNTHESIS			
Device Utilization Summary			
Logic Utilization	Used	Available	Utilization
Number of Slice Flip Flops	3,115	66,176	4%
Number of 4 input LUTs	3,696	66,176	5%

Logic Distribution			
Number of occupied Slices	2,907	33,088	8%
Number of Slices containing only related logic	2,907	2,907	100%
Number of Slices containing unrelated logic	0	2,907	0%
Total Number 4 input LUTs	4,024	66,176	6%
Number used as logic	3,696		
Number used as a route-thru	328		
Number of bonded IOBs	364	964	37%
IOB Flip Flops	355		
IOB Master Pads	11		
IOB Slave Pads	11		
Number of PPC405s	0	2	0%
Number of Block RAMs	19	328	5%
Number of GCLKs	11	16	68%
Number of DCMs	6	8	75%
Number of GTs	0	20	0%

The table below shows the resource usage for the base design **`ac210_top_sysclk`**. It is compared to the resources that are available in the target Xilinx Virtex II Pro – XC2VP70-6FF1517.

PRECISION SYNTHESIS			
Device Utilization Summary			
Logic Utilization	Used	Available	Utilization
Number of Slice Flip Flops	2,661	66,176	4%
Number of 4 input LUTs	2,958	66,176	4%
Logic Distribution			
Number of occupied Slices	2,443	33,088	7%
Number of Slices containing only related logic	2,443	2,443	100%
Number of Slices containing unrelated logic	0	2,443	0%
Total Number 4 input LUTs	3,286	66,176	4%
Number used as logic	2,958		
Number used as a route-thru	328		
Number of bonded IOBs	235	964	24%
IOB Flip Flops	231		
IOB Master Pads	11		
IOB Slave Pads	11		
Number of PPC405s	0	2	0%
Number of Block RAMs	15	328	4%
Number of GCLKs	10	16	62%
Number of DCMs	6	8	75%
Number of GTs	0	20	0%

5.14 Version History

Date	Version	Comments
April 05	Beta 1	Initial Version
June 05	Beta 2	Implemented write function to the DE-Buffer and IN-Buffer Added simulation test script for the DE-Buffer and IN-Buffer
February 06	Beta 6	New base design with external memory interface and external memory
May06	Beta 7	Revised the resources utilization.
January 07	1.0	New SC240 Base Design

6. FDK Core Library

This chapter describes in detail the FPGA cores supplied by Agilent Acqiris. By default, the cores are instantiated in a base design, showing how they should be instantiated and interconnected. If a core is not instantiated in one of the base designs, it is marked and typically described in more detail. The chapter on base designs, **5 Base Design**, lists all instantiated cores.

6.1 Index of Available Cores

Core	Library	Short Description
ac210_user_block	**ac240_developer_lib**	User block skeleton for the ac210
ac240_user_block	**ac240_developer_lib**	User block skeleton for the ac240
acq_ctr_reg	**ac240_fdk**	Standard Agilent Register
acq_tmp_struct	**ac240_fdk**	Temperature monitoring
ck_rst_manager_rg2	**ac240_fdk**	Clock management for streamer base design
ck_rst_manager_sysclk	**ac240_fdk**	Clock management for base design without external memory
ck_rst_manager_sysclk_mem	**ac240_fdk**	Clock management for base design with external memory
dac_interface	**ac240_fdk**	Front Panel MMCX analog output control
ddr_interface	**ac240_fdk**	Double Data Rate DRAM interface
ddr_ctr_example	**ac240_fdk**	Example of driving the DDR memory interface
de_interface_1ch	**fdk_lib**	1 channel interface for data input from the ADC multiplexer
de_interface_2ch	**fdk_lib**	2 channel interface for data input from the ADC multiplexer
de_interface_2ch_rg		2 channel interface for data input from the ADC multiplexer for streamer application
dlink_interface	**ac240_fdk**	Front Panel µDB digital IO control
dp_interface	**ac240_fdk**	Dual Port SRAM interface
dp_ctr_example	**ac240_fdk**	Example driving the dual port memory interface
lb_interface_m	**fdk_lib**	Local Bus Interface, wrapper to **lb_interface** of the library **fdk_lib_h**
led_interface	**ac240_fdk**	Front Panel LED control
pio_interface	**ac240_fdk**	Front Panel MMCX digital IO control
slc1_interface	**ac240_fdk**	Implementation example of a single link Serial Front Panel Data Port controller
slc_controller	**fdk_lib**	Serial Front Panel Data Port controller
slc1_interface	**ac240_fdk**	Single interface to the optical link
str1_example	**ac240_fdk**	Simple streamer example
trigger_manager	**ac240_fdk**	Trigger control
trigger_manager_1ns	**ac240_fdk**	High resolution trigger control
user_block_example	**ac240_fdk**	Example of a basic implementation, including two registers and a buffer to store data from the incoming acquisition stream

6.2 Base Clock Manager

The core **ck_rst_manager_sysclk** is the base clock manager core. This core implements all resources related to the clocking and the general reset for designs without external memories. It is also the core to choose for designs using the full capability of the available RocketIO serializers/deserializers.

It provides two global clocks (available everywhere within the Data Processing Unit) that should be used for the Internal Bus (**Lbclkg**) and for the User Core (**Sysclk**). The frequency of **Lbclkg** is set to 33 MHz whereas **Sysclk** frequency is fixed at 133 MHz.

The core also provides two clocks for the top and bottom RocketIO instances (**Usrclka, Usrclka2, Usrclkb,** and **Usrclkb2**) with some area restrictions (detailed below).

6.2.1 Functional Description

After the bit file has been loaded, the two clocks **Lbclkg** and **Sysclk** begin running. **Lbclkg** is never stopped because it handles communication with the PCI bus. **Sysclk** can be stopped by setting the input **Talarm** to **'1'**. This might be useful for shutting down the FPGA or reducing its power consumption in case its temperature rises above a limit. Apart from this unusual situation **Sysclk** should not be stopped. For monitoring the FPGA temperature, please read the descriptions of the cores **acq_tmp_struct** and **acq_ctr_reg**.

Any other clocks are disabled until they are enabled by software. The core **acq_ctr_reg** contains a control and a status register for clock control (Enable/Disable) and monitoring (DCM locked).

The Base Clock Manager core is also able to provide all external memory clocks except the clock for the port B of the SRAM (**DPMem_CKB)**. It is only for this reason that a specific memory clock manager is provided (**ck_rst_manager_sysclk_mem)**.

6.2.2 Port Description

Signal	Size	Type	Short Description
User Clock A			
REFCKA_p / _n	1	In	Clock reference for RocketIO clock generation
Refclka	1	Out	Dedicated routing path to Rocket IO instances (Top Edge)
Usrclka2	1	Out	Global clock for RocketIO instances (assuming 32-bit data path)
Usrclka	1	Out	Global clock for RocketIO instances (assuming 32-bit data path)
Usrclka_reset	1	Out	Becomes low 8 clocks after **Usrclka** DCM gets locked
User Clock B			
REFCKB_P / _N	1	In	Clock reference for bottom RocketIO clocks generation
Refclkb	1	Out	Dedicated routing path to Rocket IO instances (Bottom Edge)
Usrclkb2	1	Out	Global clock for RocketIO instances (assuming 32-bit data path)
Usrclkb	1	Out	Global clock for RocketIO instances (assuming 32-bit data path)
Usrclkb_reset	1	Out	Becomes low 8 clocks after **Usrclkb** DCM gets locked
DE-Bus clock			
DECLKA	1	In	DE-Bus A clock input
DECLKB	1	In	DE-Bus B clock input
declkag	1	Out	Internal DE-Bus A clock, through BUFG (no DCM)
declkaQg	1	Out	Reserved (Range Gate Clocking)
Declka4g	1	Out	Reserved (Range Gate Clocking)
Declka4Qg	1	Out	Reserved (Range Gate Clocking)
declkbg	1	Out	Internal DE-Bus B clock, through BUFG (no DCM)

Signal	Size	Type	Short Description
System Clock			
CK33M	1	In	Local Bus clock input
Lbclkg	1	Out	Copy of **CK33M** through DCM and BUFG (33 MHz)
Sysclk	1	Out	Global Clock 133 MHz
DmemClk	1	Out	Main Clock for DDR Controllers (166 MHz)
Talarm	1	In	Temperature Alarm input, could be used to reset the DCM that drives **Sysclk**
Sel_Fsysclk	1	In	Unused
Dual Port Memory Clocks			
DPMem_CKA	1	Out	Clock for port A of Dual Port Memory (133 MHz)
DPMem_CKFBA	1	In	Clock feedback input for Port A of Dual Port Memory
DPMem_CKFBB	1	In	Clock feedback input for Port B of Dual Port Memory
DDR Memory Clocks			
DmemClk	1	Out	Main Clock for DDR Controllers (166 MHz)
DmemClk_PS	1	Out	Phase Shifted Clock for DDR Controllers (166 MHz)
DmemClk_FB	1	Out	Clock feedback for DDR Controllers (166 MHz)
Dmem_FB_p	1	In	Clock feedback input (Differential buffer)
Dmem_FB_n	1	In	Clock feedback input (Differential buffer)
Miscellaneous			
ENB_DCM	8	In	Enable DCM Control. Shall be connected to **Enb_DCM** output of the core **acq_ctr_reg**. (2) Enable **Usrclka**. Clocks for the top edge Rocket IO instance. (3) Enable **Usrclkb**. Clocks for port the bottom edge Rocket IO instance. (4) Enable **DPMem_CKA**. Clock for port A of Dual Port Memory. (0,1,5,6,7) Not used
Lck_DCM	8	Out	DCM Lock Status. Shall be connected to **Lck_DCM** input of the core **acq_ctr_reg**. (0) '1' => **Lbclkg** locked (1) '1' => **Sysclk** locked (2) '1' => **Usrclka** locked (3) '1' => **Usrclkb** locked (4) '1' => **DPMem_CKA** locked (5) '1' => NU (6) '1' => **DmemClk_PS** locked (7) '1' => **DmemClk_FB** locked
Dreset	1	Out	General reset. **XRESETN** delayed by 12 periods of **lbclkg**.
Dreset_n	1	Out	**Dreset** inverted

6.2.3 DCM Location Constraints

The following location constraints must be applied:

DCM	Allocation	Comments

Top side DCM		
X0Y0	Dual Port Memory	Used for Dual Port Memory Clock A
X1Y0	DDR Clock Feedback	Used for DDR Controller. (Memory Option)
X2Y0	MGT Reference Clocks B	Used for RocketIO for SC Hi-Rate
X3Y0	Main Clocks	Used to generate **Lbclkg** / **MemClk**
Bottom side DCM		
X0Y1	Unused	Unused
X1Y1	DDR Phase Shifted Clock	Used for DDR Controller. (Memory Option)
X2Y1	MGT Reference Clocks A	Used for RocketIO for all SC Modules
X3Y1	User Clocks	Used to generate **Sysclk**.

6.2.4 BUFG Location Constraints

The following location constraints must be applied:

Clock PAD	Allocation	Comments
Top side BUFG		
BUFG0S	**DeclkA**	Data Entry Clock A
BUFG1P	**Unused**	--
BUFG2S	**Usrclka**	User clock for MGT (125 MHz)
BUFG3P	**Usrclka2**	User clock for MGT (62.5 MHz)
BUFG4S	**Lbclk2_fb**	Not available. Local clock for generation of **Sysclk**.
BUFG5P	**Sysclk**	Global Clock for Processing (133 MHz)
BUFG6S	**DmemClk_PS**	DDR Clock Phase Shifted (Memory Option)
BUFG7P	**DeclkB**	Data Entry Clock B
Bottom side BUFG		
BUFG0P	**DmemClk**	DDR Main Clock (Memory Option)
BUFG1S	**Lbclkg**	Internal Bus Clock – Global Clock
BUFG2P	**Usrclkb**	User clock for MGT (125 MHz)
BUFG3S	**Usrclkb2**	User clock for MGT (62.5 MHz)
BUFG4P	**DmemClk_FB**	DDR Clock Feedback (Memory Option)
BUFG5S	**Unused**	--
BUFG6P	**Lbck2_int**	Not available. Local clock for generation of **Sysclk**.
BUFG7S	**DP_CKB_int_A**	Not available. Local clock for generation of the SRAM memory clock.

6.2.5 Area Restrictions

The two clocks **Lbclkg** and **Sysclk** are distributed throughout the whole FPGA. This reduces the number of available clocking lines to 14. Firmware for the SC or AC Analyzers could use up to 14 different clocks domains with the area restrictions described below.

As stated by the Xilinx rules, any quarter of the FPGA can be fed with only up to 8 different clocks.

As much as possible, the User Core should only use the two main clocks **Lbclkg** and **Sysclk**. The other clocks are used by the Agilent Acqiris-supplied cores.

Signal	NW	NE	SW	SE
Lbclkg	[BUFG1S]	[BUFG1S]	[BUFG1S]	[BUFG1S]
Sysclk	[BUFG5P]	[BUFG5P]	[BUFG5P]	[BUFG5P]

DmemClk	[BUFG0P]		[BUFG0P]	
DmemClk_PS	[BUFG6S]		[BUFG6S]	
DmemClk_FB	[BUFG4P]		[BUFG4P]	
DeclkA		[BUFG0S]		[BUFG0S]
DeclkB	[BUFG7P]	[BUFG7P]		
UsrCka			[BUFG2S]	[BUFG2S]
UsrCka2			[BUFG3P]	[BUFG3P]
UsrCkb	[BUFG2P]	[BUFG2P]		
UsrCkb2	[BUFG3S]	[BUFG3S]		
Lbclk2_int		[BUFG6P]		
Lbclk2_fb		[BUFG4S]		
DP_CKB_int_A				[BUFG7S]

6.2.6 Clock Period Constraints

All the DCM, BUFG, IBUFG, and IBUFGDS instantiated for the clocking scheme are using "LOC" constraints to freeze the clock distribution, whatever the User firmware. For more details, please refer to the location constraints files of the Base Designs.

The IO_STANDARD as well as other constraints are enclosed in the VHDL design and are passed to the synthesizer as VHDL attributes. The synthesizer itself must transfer these constraints on to ISE.

The following clock period constraints must be applied:

Pad Name	Period	Comment
CK33M	28 ns	Continuous Clock (issued from PCI)
DECLKA	14 ns	Frequency depends on the acquisition settings
DECLKB	14 ns	Frequency depends on the acquisition settings
REFCKA_p	7.5 ns	For SC2x0 (driven by an external PLL)
REFCKB_p	7.5 ns	For SC2x0 (driven by an external PLL)
DMEM_FB_p	6 ns	For DDR Controller

Note: The synthesizer must be able to propagate these clock constraints down to the clocks generated from the DCM settings.

6.2.7 Resource Utilization

Resource count and relative usage in the target Xilinx Virtex II Pro – XC2VP70-6FF1517:

Resources	Used	Available	Utilization
DCM	6	8	75 %
Global Buffers (BUFG)	13	16	81 %
Function Generators	10	66176	~0 %
CLB Slices	20	33088	~0 %
Dffs or Latches	40	69068	~0 %

6.2.8 Version History

Date	FDK Version	Comments
April 05	Beta 1	Initial Version

Date	FDK Version	Comments
July 05	Beta 3	New Clocking Scheme for Memory Options
September 05	Beta 5	Removed the capability to select the sysclk source. Sysclk is fixed to 133 MHz.
February 06	Beta 6	Added another clock core for design with memory option: ck_rst_manager_sysclk_mem
May 06	Beta 7	Update core description

6.3 Memory Option Clock Manager

The core **ck_rst_manager_sysclk_mem** implements all resources related to the clocking and the general reset for designs including the memory option.

It provides two global clocks (available everywhere within the Data Processing Unit) that should be used for the Internal Bus (**Lbclkg**) and for the User Core (**Sysclk**). The frequency of **Lbclkg** is set to 33 MHz whereas **Sysclk** frequency is fixed to 133 MHz.

The core provides the clocks for the external memories (**DPA_CLK, DPB_CLK**) and for the core **ddr_interface** (**DmemClk, DMemClk_FB,** and **DMemClk_PS**). The clocks to the DRAM memory are driven by the core **ddr_interface_buffer** (**DDRA_CK, DDRB_CK**).

It also provides two clocks for the top RocketIO instances (**Usrclka, Usrclka2**) with some area restrictions (detailed below). It does not supply clocks for the bottom RocketIO.

6.3.1 Functional Description

After the bit file has been loaded, the two clocks **Lbclkg** and **Sysclk** begin running. **Lbclkg** is never stopped because it handles communication to the PCI bus. **Sysclk** can be stopped by setting the input **Talarm** to '1'. This might be useful for shutting down the FPGA or reducing its power consumption in case its temperature rises above a limit. Apart from this unusual situation, **Sysclk** should not be stopped. For monitoring the FPGA temperature, please read the description of the cores **acq_tmp_struct** and **acq_ctr_reg**.

The clocks for the DRAM will be active some time after **Sysclk** begins running. There is no way to stop the DDR clocks other than to disable **Sysclk**. The phase of the clocks **DMemClk_FB** and **DMemClk_PS** are calibrated during the DDR calibration phase (see the core **ddr_interface**).

The clocks for the DRAM and the RocketIO are disabled until they are enabled by software. The core **acq_ctr_reg** contains a control and a status register for clock control (Enable/Disable) and monitoring (DCM locked).

6.3.2 Port Description

Signal	Size	Type	Short Description
User Clock A			
REFCKA_p / _n	1	In	Clock reference for RocketIO clock generation
Refclka	1	Out	Dedicated routing path to Rocket IO instances (Top Edge)
Usrclka2	1	Out	Global Clock for RocketIO instances (assuming 32-bit data path)
Usrclka	1	Out	Global Clock for RocketIO instances (assuming 32-bit data path)
Usrclka_reset	1	Out	Becomes low 8 clocks after **Usrclka** DCM gets locked
User Clock B			
REFCKB_P / _N	1	In	Clock reference for bottom RocketIO clocks generation
Refclkb	1	Out	Dedicated routing path to Rocket IO instances. (Bottom Edge)
Usrclkb2	1	Out	Global Clock for RocketIO instances (assuming 32-bits data path)
Usrclkb	1	Out	Global Clock for RocketIO instances (assuming 32-bits data path)
Usrclkb_reset	1	Out	Becomes low 8 clocks after Usrclkb DCM gets locked.
DE-Bus clock			
DECLKA	1	In	DE-Bus A clock input
DECLKB	1	In	DE-Bus B clock input
declkag	1	Out	Internal DE-Bus A clock, through BUFG (no DCM)
declkaQg	1	Out	Reserved (Range Gate Clocking)
Declka4g	1	Out	Reserved (Range Gate Clocking)

Signal	Size	Type	Short Description
Declka4Qg	1	Out	Reserved (Range Gate Clocking)
declkbg	1	Out	Internal DE-Bus B clock, through BUFG (no DCM)

System Clock

CK33M	1	In	Local Bus clock input
lbclkg	1	Out	Copy of **CK33M** through DCM and BUFG (33 MHz)
Sysclk	1	Out	Global Clock 133 MHz
DmemClk	1	Out	Main Clock for DDR Controllers (166 MHz)
Talarm	1	In	Temperature Alarm input, could be used to reset the DCM that drives **Sysclk**
Sel_Fsysclk	1	In	Unused

Dual Port Memory Clocks

DPMem_CKA	1	Out	Clock for port A of Dual Port Memory (133 MHz)
DPMem_CKB	1	Out	Clock for port B of Dual Port Memory (133 MHz)
DPMem_CKFBA	1	In	Clock feedback input for Port A of Dual Port Memory.
DPMem_CKFBB		In	Clock feedback input for Port B of Dual Port Memory.

DDR Memory Clocks

DmemClk	1	Out	Main Clock for DDR Controllers (166 MHz)
DmemClk_PS	1	Out	Phase Shifted Clock for DDR Controllers (166 MHz)
DmemClk_FB	1	Out	Clock feedback for DDR Controllers (166 MHz)
Dmem_FB_p	1	In	Clock feedback input (Differential buffer)
Dmem_FB_n	1	In	Clock feedback input (Differential buffer)
Dcm_ckfb_*		In/ Out	Control for DmemClk_FB phase calibration
Dcm_ckps_*		In/ Out	Control for DmemClk_PS phase calibration
DmemClk_ reset_n		Out	Reset for the core **ddr_interface**

Miscellaneous

ENB_DCM	8	In	Enable DCM Control. Shall be connected to **Enb_DCM** output of the core **acq_ctr_reg**. (2) Enable **Usrclka**. Clocks for the top edge Rocket IO instance. (4) Enable **DPMem_CKA**. Clock for port A of Dual Port Memory. (5) Enable **DPMem_CKB**. Clock for port B of Dual Port Memory. (0,1,3,6,7) Not used
Lck_DCM	8	Out	DCM Lock Status. Shall be connected to **Lck_DCM** input of the core **acq_ctr_reg**. (0) '1' => **Lbclkg** locked (1) '1' => **Sysclk** locked (2) '1' => **Usrclka** locked (3) '1' => NU (4) '1' => **DPMem_CKA** locked (5) '1' => **DPMem_CKB** locked (6) '1' => **DmemClk_PS** locked (7)

Signal	Size	Type	Short Description
			'1' => **DmemClk_FB** locked
Dreset	1	Out	General reset. Is **XRESETN** delayed by 12 periods of **lbclkg**.
Dreset_n	1	Out	**Dreset** inverted

6.3.3 DCM Location Constraints

The following location constraints shall be applied.

DCM	Allocation	Comments
Top side DCM		
X0Y0	Dual Port Mem ClkA	Used for Dual Port Memory Clock A
X1Y0	DDR Clock Feedback	Used for DDR Controller. (Memory Option)
X2Y0	MGT Reference Clocks B	Used for RocketIO for SC Hi-Rate
X3Y0	Main Clocks	Used to generate **Lbclkg** / **MemClk**
Bottom side DCM		
X0Y1	Unused	Unused
X1Y1	DDR Phase Shifted Clock	Used for DDR Controller. (Memory Option)
X2Y1	MGT Reference Clocks A	Used for RocketIO for all SC Modules
X3Y1	User Clocks	Used to generate **Sysclk**.

6.3.4 BUFG Location Constraints

The following location constraints shall be applied.

Clock PAD	Allocation	Comments
Top side BUFG		
BUFG0S	**DeclkA**	Data Entry Clock A
BUFG1P	**Unused**	--
BUFG2S	**Usrclka**	User clock for MGT (125 MHz)
BUFG3P	**Usrclka2**	User clock for MGT (62.5 MHz)
BUFG4S	**Lbclk2_fb**	Not available. Local clock for generation of **Sysclk**.
BUFG5P	**Sysclk**	Global Clock for Processing (133 MHz)
BUFG6S	**DmemClk_PS**	DDR Clock Phase Shifted (Memory Option)
BUFG7P	**DeclkB**	Data Entry Clock B
Bottom side BUFG		
BUFG0P	**DmemClk**	DDR Main Clock (Memory Option)
BUFG1S	**LBClk**	Internal Bus Clock – Global Clock
BUFG2P	**DP_CKB_int_B**	Not available. Local clock for generation of the SRAM memory clock.
BUFG3S	**Unused**	--
BUFG4P	**DmemClk_FB**	DDR Clock Feedback (Memory Option)
BUFG5S	**Unused**	--
BUFG6P	**Lbck2_int**	Not available. Local clock for generation of **Sysclk**.
BUFG7S	**DP_CKB_int_A**	Not available. Local clock for generation of the SRAM memory clock.

6.3.5 Area Restrictions

The two clocks **Lbclkg** and **Sysclk** are distributed throughout the whole FPGA. This reduces the number of available clocking lines to 14. Firmware for the SC or AC Analyzers could use up to 14 different clocks domains with the area restrictions described below.

As stated by the Xilinx rules, any quarter of the FPGA can be fed with only up to 8 different clocks.

As much as possible, the User Core should only use the two main clocks **Lbclkg** and **Sysclk**. The other clocks are used by the Agilent Acqiris-supplied cores.

Signal	NW	NE	SW	SE
Lbclk	[BUFG1S]	[BUFG1S]	[BUFG1S]	[BUFG1S]
Sysclk	[BUFG5P]	[BUFG5P]	[BUFG5P]	[BUFG5P]
DmemClk	[BUFG0P]		[BUFG0P]	
DmemClk_PS	[BUFG6S]		[BUFG6S]	
DmemClk_FB	[BUFG4P]		[BUFG4P]	
DeclkA		[BUFG0S]		[BUFG0S]
DeclkB	[BUFG7P]	[BUFG7P]		
UsrCka			[BUFG2S]	[BUFG2S]
UsrCka2			[BUFG3P]	[BUFG3P]
DP_CKB_int_B	[BUFG2P]	[BUFG2P]		
Unused	[BUFG3S]	[BUFG3S]		
Lbclk2_int		[BUFG6P]		
Lbclk2_fb		[BUFG4S]		
DP_CKB_int_A				[BUFG7S]

6.3.6 Clock Period Constraints

All the DCM, BUFG, IBUFG, and IBUFGDS instantiated for the clocking scheme are using “LOC” constraints to freeze the clock distribution, whatever the User firmware. For more details, please refer to the location constraints files of the Base Designs.

The IO_STANDARD as well as other constraints are enclosed in the VHDL design and are passed to the synthesizer as VHDL attributes. The synthesizer itself must transfer these constraints on to ISE.

The following clock period constraints must be applied:

Pad Name	Period	Comment
CK33M	28 ns	Continuous Clock (issued from PCI)
DECLKA	14 ns	Frequency depends on the acquisition settings
DECLKB	14 ns	Frequency depends on the acquisition settings
REFCKA_p	7.5 ns	For SC2x0 (driven by an external PLL)
DMEM_FB_p	6 ns	For DDR Controller

Note: The synthesizer must be able to propagate these clock constraints down to the clocks generated from the DCM settings.

6.3.7 Resource Utilization

Resource count and relative usage in the target Xilinx Virtex II Pro – XC2VP70-6FF1517:

Resources	Used	Available	Utilization
DCM	7	8	87.5%
Global Buffers (BUFG)	13	16	81 %
Function Generators	12	66176	~0 %
CLB Slices	25	33088	~0 %

Resources	Used	Available	Utilization
Dffs or Latches	49	69068	~0 %

6.3.8 Version History

Date	FDK Version	Comments
February 06	Beta 6	Initial version.
May06	Beta 7	Description updated.

6.4 Streamer Clock Manager

The core **ck_rst_manager_rg2** is the clock manager for streamer application that includes the high resolution trigger core **trigger_manager_1ns**. It is instantiated in the streamer base design.

It provides two global clocks (available everywhere within the Data Processing Unit) that should be used for the Internal Bus (**Lbclkg**) and for the User Core (**Sysclk2**). The frequency of **Lbclkg** is set to 33 MHz whereas **Sysclk2** frequency is fixed at 133 MHz.

The core also provides two clocks for the top RocketIO instances (**Usrclka, Usrclka2**) with some area restrictions (detailed below).

The core also provides clocks for the data-entry interface and for the high resolution trigger interpolator (**declkbg, declkb4g, and declkb4Qg**). These clocks shall only be used for the data entry interface and for the high resolution trigger core. These clocks shall be enabled and could be phase adjusted by program using the register Trigger Control of the core **trigger_manager_1ns**.

After the bit file has been loaded, the two clocks **Lbclkg** and **Sysclk** begin running. **Lbclkg** is never stopped because it handles communication with the PCI bus. **Sysclk** can be stopped by setting the input **Talarm** to **'1'**. This might be useful for shutting down the FPGA or reducing its power consumption in case its temperature rises above a limit. Apart from this unusual situation **Sysclk** should not be stopped. For monitoring the FPGA temperature, please read the descriptions of the cores **acq_tmp_struct** and **acq_ctr_reg**.

Any other clocks are disabled until they are enabled by software. The core **acq_ctr_reg** contains a control and a status register for clock control (Enable/Disable) and monitoring (DCM locked).

NOTE Because DCM are used for generation of all clocks **DeclkX**, this core should be used only for ADC sampling rate of 500 MS/s and 1 GS/s (in interleaved mode, this is equal to a sampling rate of 1 GS/s and 2 GS/s). For lower sampling rates, the DCM will unlock and the behavior will not be guaranteed. Lower sampling rates can be implemented by sparsing the data within the firmware.

6.4.1 Port Description

Signal	Size	Type	Short Description
User Clock A			
REFCKA_p / _n	1	In	Clock reference for RocketIO clock generation
Refclka	1	Out	Dedicated routing path to Rocket IO instances (Top Edge)
Usrclka2	1	Out	Global clock for RocketIO instances (assuming 32-bit data path)
Usrclka	1	Out	Global clock for RocketIO instances (assuming 32-bit data path)
Usrclka_reset	1	Out	Becomes low 8 clocks after **Usrclka** DCM gets locked
DE-Bus clock			
DECLKA	1	In	DE-Bus A clock input
DECLKB	1	In	DE-Bus B clock input
declkag	1	Out	Internal DE-Bus A clock
declkbg	1	Out	Internal DE-Bus B clock
declkbQg	1	Out	Reserved (HiRes trigger Clocking)
Declkb4g	1	Out	Reserved (HiRes trigger Clocking)
Declkb4Qg	1	Out	Reserved (HiRes trigger Clocking)
High resolution trigger DCM phase asjustment			
DcmRG_PS_Cor	8	In	Phase value
DcmRG_Rst	1	In	Reset Phase
Dcm_RG_PDone	1	Out	Phase Done
System Clock			

Signal	Size	Type	Short Description
CK33M	1	In	Local Bus clock input
lbclkg	1	Out	Copy of **CK33M** through DCM and BUFG (33 MHz)
Sysclk	1	Out	Clock 66 MHz
Sysclk2	1	Out	Global Clock 133 MHz
Talarm	1	In	Temperature Alarm input, could be used to reset the DCM that drives **Sysclk**
Miscellaneous			
ENB_DCM	8	In	Enable DCM Control. Shall be connected to **Enb_DCM** output of the core **acq_ctr_reg**. (1) Enable **Usrclka**. Clocks for the top edge Rocket IO instance. (0, 2 to 7) Not used
Lck_DCM	8	Out	DCM Lock Status. Shall be connected to **Lck_DCM** input of the core **acq_ctr_reg**. (0) '1' => **Lbclkg** locked (1) '1' => **Sysclk** locked (2) '1' => **Usrclka** locked (3) Equal to **Enb_DCM**(3) (4) '1' => **declkbg and declkbQg** locked (5) '1' => **declkb4g and declkb4Qg** locked (6) Equal to **Enb_DCM**(4) (7) Always '1'
Dreset	1	Out	General reset. Is **XRESETN** delayed by 12 periods of **lbclkg**.
Dreset_n	1	Out	**Dreset** inverted
XRESETN	1	In	**General reset**

6.4.2 DCM Location Constraints

The following location constraints must be applied:

DCM	Allocation	Comments
Top side DCM		
X0Y0	Unused	Unused
X1Y0	Unused	Unused
X2Y0	Internal Bus Clocks	Used to generate **Lbclkg**
X3Y0	Global system clock	Used to generate **Sysclk** and **Sysclk2**.
Bottom side DCM		
X0Y1	Unused	Unused
X1Y1	MGT Reference Clocks A	Used for RocketIO for the simple streamer
X2Y1	Data entry clock	Used for de_interface **declkbg and declkbQg**.
X3Y1	Trigger manager clocks	Used for **declkb4g and declkb4Qg**

6.4.3 BUFG Location Constraints

The following location constraints must be applied:

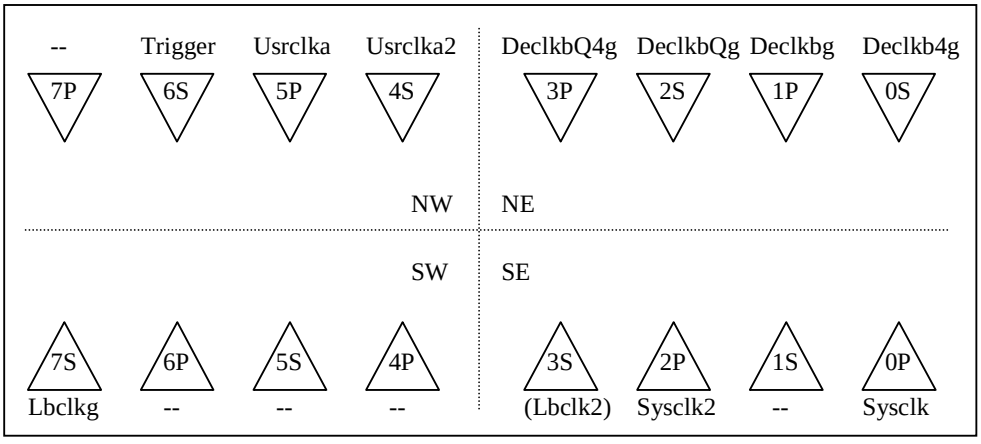
Clock PAD	Allocation	Comments
Top side BUFG		
BUFG0S	**Declkb4g**	Data Entry Clock B times 4
BUFG1P	**Declkbg**	Data Entry Clock B
BUFG2S	**DeclkbQg**	Data Entry Clock B in quadrature
BUFG3P	**Declkb4Qg**	Data Entry Clock B times 4 in quadrature
BUFG4S	**Usrclka2**	User clock for MGT (62.5 MHz)
BUFG5P	**Usrclka**	User clock for MGT (62.5 MHz)
BUFG6S	**Trigger**	This BUFG is instantiated in the core **trigger_manager_1ns**
BUFG7P	**Unused**	--
Bottom side BUFG		
BUFG0P	**Sysclk**	Clock for Processing (66 MHz)
BUFG1S	**Unused**	--
BUFG2P	**Sysclk2**	Global Clock for Processing (133 MHz)
BUFG3S	**Lbck2_int**	Not available. Local clock for generation of **Sysclk**.
BUFG4P	**Unused**	--
BUFG5S	**Unused**	--
BUFG6P	**Unused**	--
BUFG7S	**Lbclkg**	Internal Bus Clock – Global Clock

6.4.4 Area Restrictions

The two clocks **Lbclkg** and **Sysclk2** are distributed throughout the whole FPGA. This reduces the number of available clocking lines to 14. Firmware for the SC or AC Analyzers could use up to 14 different clocks domains with the area restrictions described below.

As stated by the Xilinx rules:

- Any quarter of the FPGA can be fed with only up to 8 different clocks.
- Facing BUFG could not access the same quadrant.



As much as possible, the User Core should only use the two main clocks **Lbclkg** and **Sysclk2**. The other clocks are used by the Agilent Acqiris-supplied cores.

Sysclk2 is a global clock because the facing BUFG handle the clock **declkbQg** which drives no ressources at the exception of the feedback loop of a DCM.

Sysclk could access all quadrants except the NE quadrant. Only one of clock **Sysclk** and **Sysclk2** can be the global clock. The choice here is to have the faster clock as the global clock. This could be modified by using the BUFG 2P for **Sysclk** and 0P for **Sysclk2**. In this case, **Sysclk** will be global and **Sysclk2** will not.

All the DCM, BUFG, IBUFG, and IBUFGDS instantiated for the clocking scheme are using “LOC” constraints to freeze the clock distribution, whatever the User firmware.

The IO_STANDARD, the BUFG location, as well as other constraints are enclosed in the VHDL design and are passed to the synthesizer as VHDL attributes. The synthesizer itself must transfer these constraints on to ISE.

6.4.5 Clock Period Constraints

The following clock period constraints must be applied:

Pad Name	Period	Comment
CK33M	28 ns	Continuous Clock (issued from PCI)
DECLKA	14 ns	Frequency depends on the acquisition settings
REFCKA_p	7.5 ns	For SC2x0 (driven by an external PLL)

Note: The synthesizer must be able to propagate these clock constraints down to the clocks generated from the DCM settings.

6.4.6 Resource Utilization

Resource count and relative usage in the target Xilinx Virtex II Pro – XC2VP70-6FF1517:

Resources	Used	Available	Utilization
DCM	5	8	62.5 %
Global Buffers (BUFG)	10	16	62.5 %
Function Generators	51	66176	0.08 %
CLB Slices	26	33088	0.08 %
Dffs or Latches	43	69068	0.06 %

6.4.7 Version History

Date	FDK Version	Comments
January 07	1.0	New core for the Streamer Base Design

6.5 User Block Skeleton

The component **ac240_user_block** is the skeleton to complete when designing firmware for the AC240 or SC240. The component **ac210_user_block** is the equivalent for the AC210 and SC210; only the connections of the second channel are removed.

Each User Block is essentially empty, but has almost all possible IO connections to the other cores that are instantiated in the Base Designs. All output signals are set to their default state.

The external µdB Signal IOs are configurable by the customer and thus should be adapted to the requirements.

The two User Block skeletons reside in the library **ac240_fdk** as well as in the library **developer_lib**. Developers should only modify those in **developer_lib** or a copy of it.

6.5.1 Port Description

Port Name	Size	Type	Default	Description
IB_Customer **IB_Dirsel** **IB_Write** **IB_Valid** **IB_Rdy** **IB_TimeO** **IB_End** **IB_IndirCtr** **IB_Addr** **IB_DataW** **IB_DataR**	1 1 1 1 1 1 1 32 32 32 32	In In In In Out In Out In In In Out	 '0' '0' all '0'	Should be connected to the IB-BUS signal with the same name. For details, please refer to the description of the IB-BUS.
Internal Bus				
IB_Clk	1	In		Internal Bus Clock
Reset	1	In		Start Up Reset
Enable_Trigger	1	Out	'0'	Trigger Enable to the Trigger core
SP_Data_A	128	In		Samples from channel A
SP_Data_Val_A	1	In		Data valid from channel A
Sp_first_A	4	In		Position of the trigger in the data block
SP_Data_B	128	In		Samples from channel B (AC240 only)
SP_Data_Val_B	1	In		Data valid from channel B (AC240 only)
SP_Trigger	1	In		Trigger marker
Sysclk	1	In		System clock
Big-Endian	1	In		Input control '1' for Big-Endian, '0' for little-Endian '0'
DAC Controller				
DA_Data	16	Out	'0'	Data Output to the DAC interface
DA_Write	1	Out	'0'	Write strobe for Data Output to the DAC interface
DA_Done	1	In		'1' when done writing to the DAC interface
DA_Busy	1	In		'1' when the DAC interface is busy
MMCX IO Controller				
io1_dir	1	Out	'1'	Direction control for IO1 line if not overridden by Register Control
io1_in	1	In		Input from IO1 line if not overridden by Register Control
io1_out	1	Out	'0'	Signal to output on IO1 line if not overridden by Register Control

Port Name	Size	Type	Default	Description
io2_dir	1	Out	'1'	Direction control for IO2 line if not overridden by Register Control
io2_in	1	In		Input from IO2 line if not overridden by Register Control
io2_out	1	Out	'0'	Signal to output on IO2 line if not overridden by Register Control
IO_Fct_Usr	32	Out	All '0'	User-defined signals to be multiplexed on the PIO Outputs
LED Controller				
Li1_CCd	2	Out	All '0'	User control for front panel LED L1
Li2_CCd	2	Out	All '0'	User control for front panel LED L2
µDB Controller				
DIO_in	7	In		7 inputs from µDB front panel connector. Arbitrarily set to input. Developers may modify this choice as required.
DIO_out	7	Out	All '0'	7 outputs to µDB front panel connector. Arbitrarily set to input. Developers may modify this choice as required.

6.5.2 Version History

Date	Version	Comments
April 05	Beta 1	Initial Version.
May 06	Beta 7	Description updated.

6.6 User Block Example

The core **user_block_example** is delivered as an example in the Base Design. It has a built-in buffer, the IN-Buffer, capable of storing 8K samples per channel from the internal data stream. The IN-Buffer contents can be displayed with the application AcqirisAnalyzers.

There is a control register to configure the mode of operation and a status register indicating if the IN-Buffer is full.

This component can be found in the library ac240_fdk as well as in the library developer_lib. Developers should only modify the one in developer_lib or a copy of it.

6.6.1 Functional Description

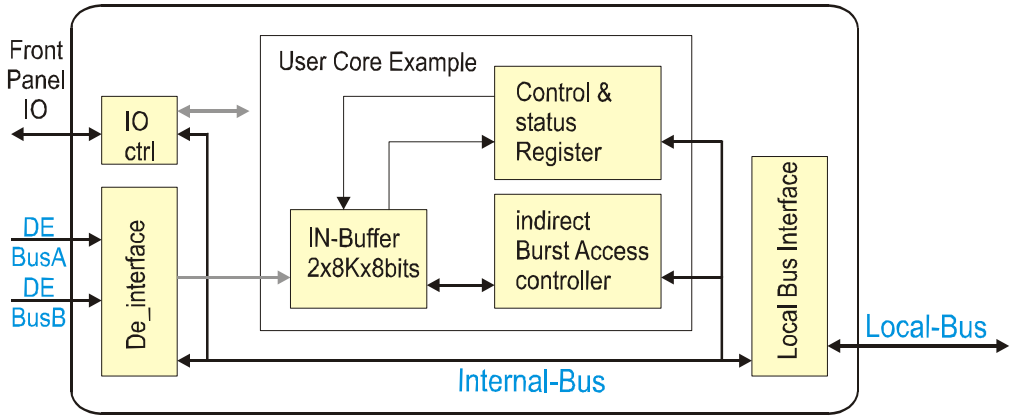
The configuration of the user firmware is typically done ***after*** the continuous acquisition has been started. The operating mode of the IN-Buffer must be configured either to triggered, by setting the bit **Triggered** of the control register to '0', or to non-triggered by setting the bit **Triggered** to '1'. Filling the IN-Buffer is enabled by setting the bit **Start** in the control register to '1'.

Filling the IN-Buffer will start either immediately or after a trigger occurred. The bit **Full** of the status register will be set '1' after the IN-Buffer has been completely filled. The signal **Buffer_full** reflects the state of the bit **Full**.

Reading the IN-Buffer should begin only after the buffer has been filled, i.e. **Full** is '1'. Before reading the IN-Buffer you should select the read mode, either CHA, CHB, or interleaved, by setting the bits **InRwMode** to the desired value. In the case of AC210 or SC210, the mode should be set to CHA.

The bit **Full** of the status register and the signal **Buffer_full** from the User Block will be reset to '0' after the bit **Start** in the control register is set back to '0'.

One could use the signal **Buffer_full** to generate an interrupt.



6.6.2 Port Description

Port Name	Size	Type	Description
IB_Customer	1	In	Should be connected to the IB-BUS signal with the same name. For details, please refer to the description of the IB-BUS.
IB_Dirsel	1	In	
IB_Write	1	In	
IB_Valid	1	In	
IB_Rdy	1	Out	
IB_TimeO	1	In	
IB_End	1	Out	
IB_IndirCtr	32	In	
IB_Addr	32	In	
IB_DataW	32	In	
IB_DataR	32	Out	
IB_Clk	1	In	Internal Bus Clock

Port Name	Size	Type	Description
Reset	1	In	Start Up Reset
Enable_Trigger	1	Out	Trigger Enable to the Trigger core
SP_Data_A	128	In	Samples from channel A
SP_Data_Val_A	1	In	Data valid from channel A
Sp_first_A	4	In	Position of the trigger in the data block
SP_Data_B	128	In	Samples from channel B (AC240 only)
SP_Data_Val_B	1	In	Data valid from channel B (AC240 only)
SP_Trigger	1	In	Trigger marker
Sysclk	1	In	System clock
Big-Endian	1	In	Input control '1' for Big-Endian, '0' for little-Endian
SW_LED	1	Out	Software LED control
Full	1	Out	Buffer Status, '1' for full. The activity of this signal is controlled by the bit LED of the control register. It should be set to '1' to enable the output.
Test	16	Out	Test signals output for monitoring
DA_Data	16	Out	Data Output to the DAC interface
DA_Write	1	Out	Write strobe for Data Output to the DAC interface
DA_Done	1	In	'1' when write done from the DAC interface
DA_Busy	1	In	'1' when the DAC interface is busy

6.6.3 Registers

6.6.3.1 User Control Register

Register Space	Register Number	Register Address
Customer	**64**	**0x2300**

31..13	12	11	10	9..8
	Start		Triggered	

7..6	5..4	3..1	0
	InRWMode		LED

[0] **LED** **RW** Software LED control for front-panel LED L1. This bit also controls the activity on the output signal **Full**. **LED** must be set to '1' to enable the signal **Full**.

[5..4] **InRWMode** **RW** Configure IN-Buffer for read and write operation

- 00 Channel A
- 01 Channel B (AC240 only)
- 10 Channel A and B Interleaved (A0,B0,A1,B1…) (AC240 only)

[10] **Triggered** **RW** Select triggered or non-triggered mode

- 0 Sample will be stored to the IN-Buffer immediately after the Start is issued.
- 1 Sample will be stored to the IN-Buffer after next Trigger following the Start.

[12] **Start** **RW** Start acquiring immediately if the bit **Triggered** is set to '0', or after the first trigger occurrence if the bit **Triggered** is set to '1'.

Start must be maintained '1' until the buffer is read.

6.6.3.2 User Status Register

Register Space	Register Number	Register Address
Customer	**65**	**0x2304**

31	30..0
Full	

[31] **Full** **R** Buffer full. It becomes '1' after the buffer has become full. It is cleared when the bit Start returns to '0', enabling a new acquisition to start.

6.6.4 Accessing the IN-Buffer

The IN-Buffer can be read using the Indirect Addressing register. The IN-Buffer contains 8K samples per channel. The Indirect Address Register and Buffer Identifier Register should be set prior to reading or writing the IN-Buffer.

6.6.4.1 IN-Buffer

Register Space	Register Number	Register Address	Buffer Identifier
Customer	**0**	**0x2200**	**0x0C**

31..24	23..16	15..8	7..0
D3	D2	D1	D0

[31..0] **D3 - D0** **RW** The bytes D0-D3 each correspond to an 8-bit sample.

The order, big or little Endian, is configured in the Acqiris general control register (see the core **acq_ctr_reg**).

6.6.5 Resource Utilization

Resource count and relative usage in the target Xilinx Virtex II Pro – XC2VP70-6FF1517:

Resources	**Used**	**Available**	**Utilization**
Function Generators	738	66176	1.1%
CLB Slices	369	33088	1.1%
Dffs or Latches	219	69068	0.3%
Block RAMs	8	328	2.4%

6.6.6 Version History

Date	**FDK Version**	**Comments**
June 05	Beta 2	Implemented write function to the IN-Buffer.
April 05	Beta 1	Initial Version.
May 06	Beta 7	Description updated.

6.7 Local Bus Interface

The core **lb_interface_m** is the FPGA interface to the Local Bus. Its source is not open. Agilent delivers a compiled version for Modelsim and an EDIF file for ISE.

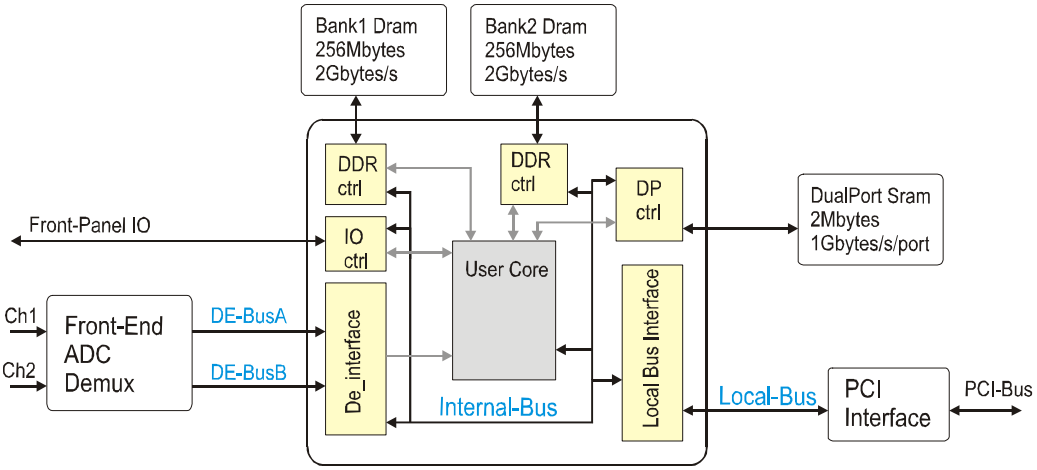
This core also contains a firmware identifier used by the driver to prevent unauthorized use of the firmware. For details, refer to the section 6.7.5 ***PROTECTION OF FIRMWARE CODE***.

Examples of standard target components handling single and burst transfers can be found among the cores **lb_iotarget_*** in the library **fdk_lib**.

The Local Bus and Internal Bus protocols were described in detail in sections 4.4 ***LOCAL BUS*** and 4.5 ***INTERNAL BUS***.

6.7.1 Functional Description

The queries from the PCI interface are decoded and generate transactions to the Internal-Bus, IB bus. The dialog is very simple, based on a selection signal passed along with address, data, and direction. The core **lb_interface_m** waits for the acknowledgement (**IB_Ready** & **IB_End**) to end the transaction.



The Local Bus interface is entirely synchronous to the PCI clock, always running at 33 MHz.

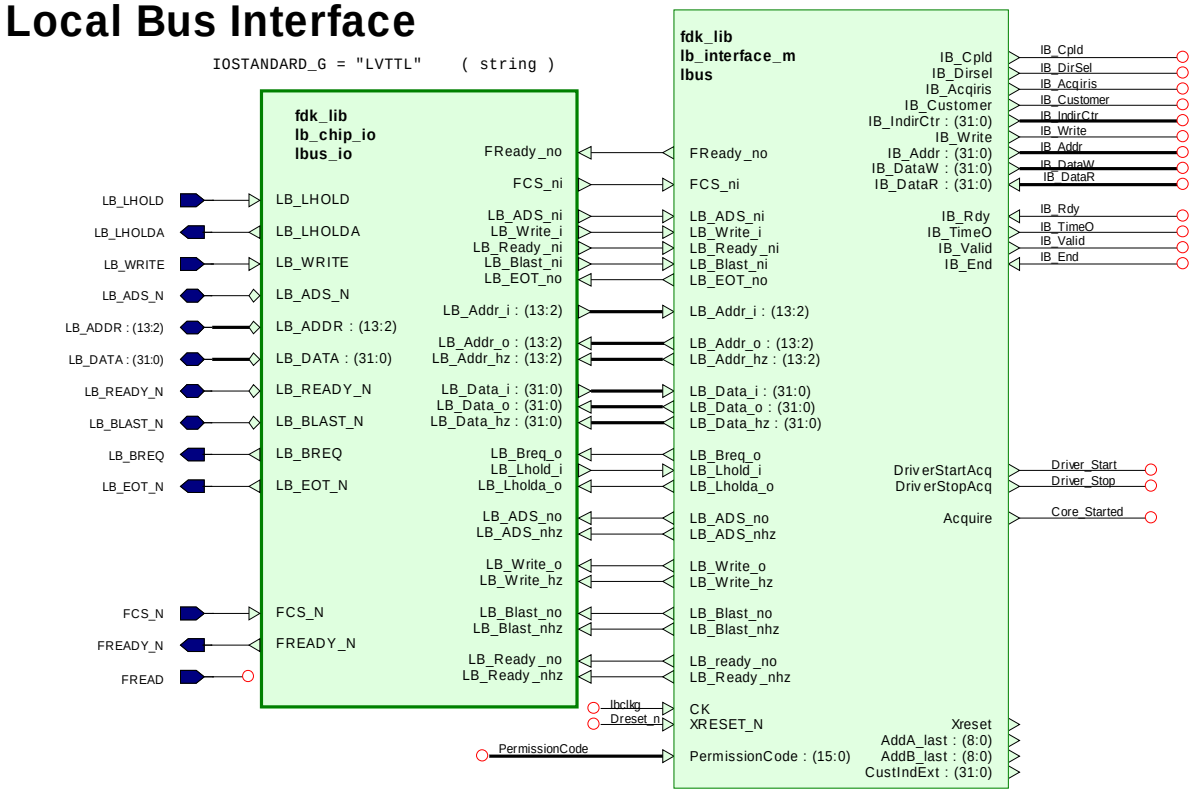
Two access types have been defined, Direct and Indirect. The Direct Access type directly reads or writes a single 32-bit word, while Indirect Access handles large data buffers.

The Local Bus interface contains a number of data pipelines. If a data read operation is the continuation of a previous read (i.e. is initiated without a new address), then the Local Bus interface will present the data in the pipelines until they are empty. Only at that point will the Local Bus interface request more data from the internal target.

For example if Indirect Access is used in non-DMA mode, each read will be a single read. The first read transaction will fill the entire pipeline; the subsequent ones will read the pipeline until empty and then read the target again.

6.7.2 Instantiation

This core is connected to the FPGA Local Bus port through its companion, the core **lb_chip_io** which includes all Xilinx I/O buffers. On the left are the connections outside the FPGA, on the right the connections to the FPGA Internal Bus.



6.7.3 Port Description

The table below only lists the connections to the Local Bus and to the Internal Bus.

Signal	Size	Type	Short Description
LOCAL BUS			
LB_LHOLD	1	In	External device request Local Bus transfer
LB_LHOLDA	1	Out	Grant to External to access the Local Bus
LB_WRITE	1	In	Local Bus access direction
LB_ADS_N	1	InOut	Local Bus address strobe
LB_ADDR	12	InOut	Local Bus address
LB_DATA	32	InOut	Local Bus data
LB_READY_N	1	InOut	Local Bus ready
LB_BLAST_N	1	InOut	Local Bus flag for last transfer of a burst
LB_BREQ	1	Out	Local Bus request
LB_EOT_N	1	Out	Abort DMA (not supported)
FCS_N	1	In	FPGA Chip select from the CPLD
FREADY_N	1	Out	Ready to the CPLD (is two cycles prior **LB_READY_N**)
FREAD	1	In	Local Bus access direction from the CPLD
CK	1	In	Local bus clock : 33 MHz
XRESET_N	1	In	General reset active ‘0’
XRESET	1	Out	General reset active ‘1’
PermissionCode	16	In	Code for firmware protection, by default set to ‘0’
FPGA INTERNAL BUS			
IB_Customer	1	Out	Addressing destination of the current access:

Signal	Size	Type	Short Description
			0 = not in the User Core address space 1 = within the User Core address space
IB_Cpld	1	Out	Agilent reserved usage, select CPLD addressing space
IB_Acqiris	1	Out	Agilent reserved usage, select Agilent addressing space
IB_Dirsel	1	Out	Addressing type of current access: 0 = indirect, 1 = direct
IB_IndirCtr	32	Out	This signal defines the target for Indirect Access. **IB_IndirCtr** uses the value of the register **IndirectControl** when the current access is Indirect.
IB_Write	1	Out	Access direction: 1 = write (data from the driver) 0 = read (data to the driver)
IB_Addr	32	Out	This signal defines the start address for accesses to the **Indirect Data Port** or the register number for accesses to the **Direct Access Registers**. For accesses to the **Indirect Data Port**, the value will be a copy of the **IndirectAddress** Register. For accesses to the **Direct Access Registers**, the bits 16 to 23 takes the value of the bits 0 to 7 of the register **Direct Access Block**, the bits 0 and 1 will be '0' and the bits 2 to 13 will take the value of the bits 2 to 13 of the signal **LB_ADDRESS**. (The bits 2 to 8 represent the Direct Register Number).
IB_DataW	32	Out	32-bit Write Data Bus
IB_DataR	32	In	32-bit Read Data Bus All unselected devices connected to the IB bus must drive x"00000000" onto these lines to implement a data read multiplexer with a simple OR function.
IB_Rdy	1	In	The selected device must drive '0' when the data (read) or the device (write) is not ready or not selected. The selected device must drive '1' when the data (read) or the device (write) is ready and selected. All unselected devices connected to the IB bus must drive '0' onto this line to implement a multiplexer with a simple OR function.
IB_TimeO	1	Out	Set '1' when a device failed to acknowledge **IB_Rdy** within 64-clock cycle after an IB cycle has been started. The cycle will end without retry. **IB_TimeO** will then come back to '0'.
IB_Valid	1	Out	**IB_Valid** is '1' when **IB_DataW** is valid. In case of burst write, it should be used to load the data to the selected device. For burst write access **IB_Valid** can be one for multiple clock periods, denoting burst write of successive data.
IB_End	1	In	If this input is '1' it enables the Local Bus interface to execute an access to the Internal Bus. It is normally driven '1' except for Indirect Access. For Indirect Access, the selected device must drive '0' onto this line for the entire time of the access. **IB_End** must be set to '1' when the device has completed the access and when it is ready for another one.

ACQUISITION STATUS

Acquire	1	Out	Becomes '1' after the driver function **AcqrsD1_acquire** is

Signal	Size	Type	Short Description
			executed and '0' after the driver function **AcqrsD1_stopAcquisition** is executed.
DriverStartAcq	1	Out	Becomes '1' for a single clock cycle when the driver function **AcqrsD1_acquire** is executed.
DriverStopAcq	1	Out	Becomes '1' when the driver function **AcqrsD1_stopAcquisition** is executed.

6.7.4 Access Time Out

If a target does not respond **IB_Rdy** within 64 clock cycles, a timeout will be issued setting **IB_TimeO** to '1'. The Local Bus interface automatically generates an acknowledge signal to the Local Bus, setting **LB_READY_N** to '0' until completion of the Local Bus access. Any data will be lost, both on reading and writing. Any target should reset itself to the idle state when time out occurs.

6.7.5 Protection of Firmware Code

The core **lb_interface_m** contains a firmware identifier used by the driver to prevent unauthorized use of the firmware.

If the 16 lower bits of the Permission Code Register are set to 0x0000 or 0xFFFF with the signal **PermissionCode**, the driver software considers the firmware as unprotected and permits software access to the FPGA. If another value is set (for details, see Code Protection Register in the next section), the driver only grants access to the FPGA if the on-board EEPROM contains the corresponding code. This mechanism permits the authorization of protected firmware usage on each AC/SC2x0 individually. A single board can support multiple protection codes so that it can be used with a number of different protected firmware versions.

The EEPROM must be loaded by Agilent, so developers wishing to use the protection mechanism should contact Agilent.

6.7.6 Registers

6.7.6.1 Overview

Register	Register Space	Register Address	Ac2x0_FDK	Short Description
0	Cust/Reserved	0x2200	Available	Indirect Access Port
1	Cust/Reserved	0x2204	Available	Indirect Address Register
2	Cust/Reserved	0x2208	Available	Buffer Identifier Register
4	Cust/Reserved	0x2210	Available	Code for Firmware Protection
9	Cust/Reserved	0x2224	Available	FPGA Direct Access Block Registers

6.7.6.2 Indirect Access Port

This register gives access to large data blocks, together with the Indirect Address Register and the Buffer Identifier Register. As seen by the control software, it acts like a FIFO data port.

Register Space	Register Number	Register Address
Customer	**0**	**0x2200**

31..0
IndirData

[31..0] **IndirData** **RW** Indirect Data value. Every read or write access uses the indirect address defined by the Indirect Address and Buffer Identifier registers.

6.7.6.3 Indirect Address Register

Register Space	Register Number	Register Address

Customer	1	0x2204

31..0
IndirAddr

[31..0] **IndirAddr** **RW** This register defines the address for Indirect Access.

It is used when accessing the Indirect Access Port. This address is defined in bytes and is auto incremented by 4 for each read or written word from / to the Indirect Access Port. The signal **IB_Addr** will take the value of **IndirAddr** when the access on the Internal Bus is an Indirect Access.

6.7.6.4 Buffer Identifier Register

Register Space	Register Number	Register Address
Customer	**2**	**0x2208**

31..0
IndirCtr

[31..0] **IndirCtr** **RW** This register defines the target for Indirect Access.

The signal **IB_IndirCtr** takes the value of **IndirCtr** when the access on the Internal Bus is an Indirect Access.

6.7.6.5 Code Protection Register

Register Space	Register Number	Register Address
Customer	**4**	**0x2210**

31..16	15..4	3..0
	DeveloperID	FirmwareID

NOTE: A firmware permission code is a 16-bit value that is embedded within any FPGA Firmware. This is the value of the signal **PermissionCode** connected to the core **lb_interface_m**.

The values 0x0000 and 0xFFFF are reserved, to define unprotected firmware.

Using a different permission code value would block any FPGA register access on modules that do not contain this value within their EEPROM. Please contact Agilent Technical support if you want to modify the EEPROM for such protection.

[3..0] **FirmwareID** **R** 16 Firmware identifiers:

Use only the values 0x0 to 0xE. Each of these 15 values, together with a unique value of **DeveloperID**, identifies a firmware. Of course, several firmware codes may contain the same identifier if there is no need to give them different permissions.

The AC/SC2x0 EEPROM must contain the same **DeveloperID** and **FirmwareID** combination as the firmware. Otherwise, the driver refuses to access the FPGA.

If the EEPROM contains the value 0xF in the **FirmwareID**, then all firmware versions with the same **DeveloperID** are permitted, independently of the value of **FirmwareID** in the firmware.

[15..4] **DeveloperID** **R** Each Developer ID lets the FDK developer define and use up to 15 Firmware IDs to protect the firmware against unauthorized use.

6.7.6.6 Direct Access Block Register

Register Space	Register Number	Register Address

Customer	9	0x2224

31..8	7..0
- -	**DIR_BLOCK_NBR**

[7..0] **DIR_BLOCK_NBR** **RW** This register defines the IB-BUS register block for subsequent accesses to the **Direct Access Registers**.

The bit signal **IB_Addr** takes the value of **IndirCtr** when the access on the Internal Bus is an Indirect Access.

6.7.7 Constraints

- Clock Constraint

This core assumes a CLK frequency of 33 MHz. This constraint must be defined for the clock manager component and is automatically propagated throughout the whole design.

- Input Signal Constraints

Pad to register delay must not exceed 10 ns. For more details, please read the .sdc or .ucf constraint files of the BaseDesign.

- Output Signal Constraints

All output signals are registered. All output registers should be located within the IOB. All are of type LVTTL, slow, 12 mA.

Clock to output delay must not exceed 10 ns. For more details, please read the .sdc or .ucf constraint files of the BaseDesign.

6.7.8 Resource Utilization

Resource count and relative usage in the target Xilinx Virtex II Pro – XC2VP70-6FF1517:

Resources	Used	Available	Utilization
Block RAMs	1	328	0.3%
Function Generators	529	66176	0.8%
CLB Slices	369	33088	1.1%
Dffs or Latches	737	69068	1.1%

6.7.9 Version History

Date	Version	Comments
April 05	Beta 1	Initial Version
May 06	Beta 7	Description updated

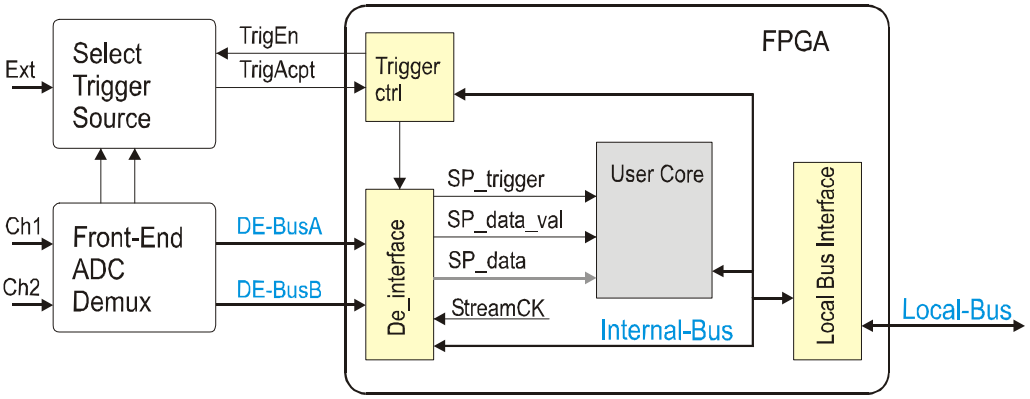
6.8 DE Interface for 1 and 2 Channels

The cores **de_interface_1ch** and **de_interface_2ch** are the FPGA interfaces handling the digitized data input streams for the AC210 and the AC240.

6.8.1 Functional Description

After the ADC conversion, the samples from each channel are de-multiplexed to the DE-Bus. Each DE-Bus is 16 samples wide, transmitting 16 samples every 16 ns when running at the maximum sample rate, 1 GS/s/channel.

When the AC240 is configured to run as a single channel at 2 GS/s, the input signal Ch1 or Ch2 is sent to both ADCs. Each ADC drives the DE-Bus in an identical manner as in the two-channel configuration.



Once the acquisition is started, a continuous stream of data flows to the FPGA, along with the DE-Bus clocks **DECLKA** and **DECLKB**. Both are equal to the ADC clock, divided by 16.

The core has a built-in Buffer, the DE-Buffer, which has two ports and behaves like a FIFO. The data are clocked at the input on the falling edge of **DECLKA,** respectively **DECLKB,** and are read out with the clock **StreamCK**. **StreamCK** can be completely asynchronous to the input clock. Of course, **StreamCK** cannot be slower than **DECLKx**. When the frequency of **StreamCK** is greater than the frequency of the **DECLKx** clock, the data valid signals **SP_Data_Val_A** and **SP_Data_Val_B** are set '1' when there is valid data on the data output **SP_Data_A** and **SP_Data_B**.

StreamCK is usually connected to the system clock **Sysclk**, defined in the base designs.

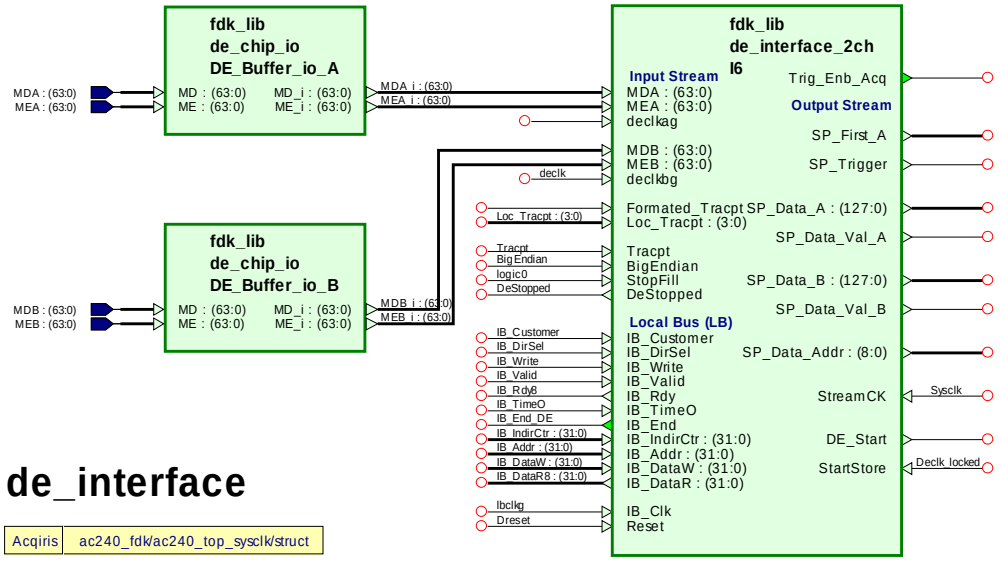
The DE-Buffer also handles the trigger status signal which can be used to determine the position of a trigger with the resolution of one sample or one data block of 16 samples, depending on the trigger core.

The DE-Buffer can be read or written through the Local Bus. Its contents can be frozen (by preventing the incoming data stream from overwriting the buffer) and continuously repeated on the output stream port. This might be useful for verifying the operation of the firmware with exactly known data (acquired data always contains a small amount of noise). DE-Buffer is 8 KB per channel.

6.8.2 Instantiation

The core **de_interface_2ch** is connected to two DE-Bus instances through its companion **de_chip_io** that includes all Xilinx I/O buffers for one bus. On the left are the connections outside the FPGA, on the right the connections to the FPGA internal cores.

Instantiation of the core **de_interface_1ch** is identical, except that the component **DE_Buffer_io_B** is not instantiated.



6.8.3 Port Description

Signal	Size	Type	Short Description
INTERNAL BUS			
IB_Customer **IB_Dirsel** **IB_Write** **IB_Valid** **IB_Rdy** **IB_TimeO** **IB_End** **IB_IndirCtr** **IB_Addr** **IB_DataW** **IB_DataR**	1 1 1 1 1 1 1 32 12 32 32	In In In In Out In Out In In In Out	Should be connected to the IB-BUS signal with the same name. For details please refer to the description of the IB-BUS.
IB_Clk	1	In	Internal Bus Clock It must be connected to **lbclkg**, the Local Bus clock.
Reset	1	In	Reset It must be connected to the general reset **Dreset.**
DE BUS – Data input Stream			
MDA	64	In	Data input streamA, samples 0 to 7 of the incoming data block (gray coded)
MEA	64	In	Data input streamA, samples 8 to 15 of the incoming data block (gray coded)
declkag	1	In	Data input streamA clock. **MDA** and **MEA** are clocked by the falling edge of the clock.
MDB	64	In	AC240 only. Data input streamB, samples 0 to 7 of the incoming data block (gray coded)
MEB	64	In	AC240 only. Data input streamB, samples 8 to 15 of the incoming data block (gray coded)
DECLKBG	1	In	AC240 only. Data input streamB clock. **MDB** and **MEB** are clocked by the falling edge of the clock.
Formated_tracpt	1	In	After the trigger has been enabled (see details in the description of the core **trigger_manager**), indicates that a trigger occurred in the current data block.

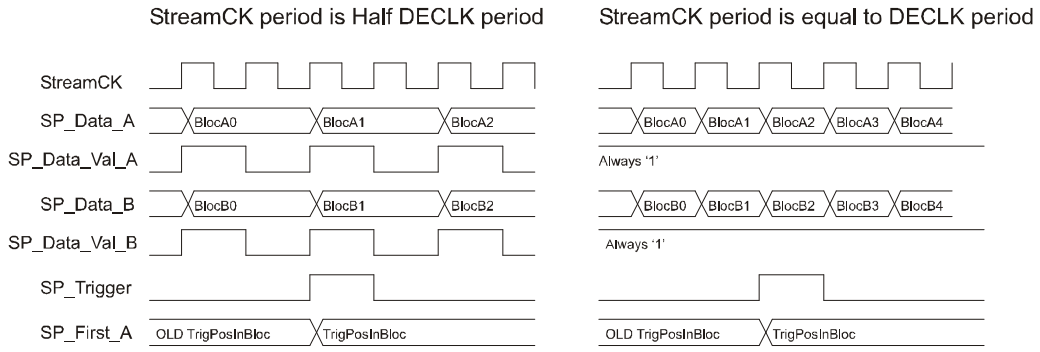
Signal	Size	Type	Short Description
			Formated_tracpt remains '1' for a single **declk** period. It must be connected to the equivalent signal out of the core **trigger_manager**.
Loc_tracpt	4	In	Location of the trigger within the block of 16 samples. This is available only if the trigger core **trigger_manager_1ns** is used. It must be connected to the equivalent signal out of the trigger core.
Tracpt	1	In	Raw accepted trigger signal. It must be connected to the equivalent signal out of the trigger core.
BigEndian	1	In	Select format for FIFO readout to the application: '0': Little Endian '1': BigEndian
StopFill	1	In	Function not available, reserved for future use. It must be connected to '0'.
DeStopped	1	Out	A '1' denotes the **de_interface** is actually stopped. It is '0' when running.

OUTPUT STREAM

SP_First_A	4	Out	4-bit trigger position in the data block. Only valid if the core **trigger_manager_1ns** is used.
SP_Trigger	1	Out	'1' indicates a trigger occurred in the current data block.
SP_Data_A	128	Out	FIFO output 128 bits, 16 x 8-bit samples from DE-Bus A
SP_Data_Val_A	1	Out	Data valid at FIFO output, channel A
SP_Data_B	128	Out	AC240 only. FIFO output 128 bits, 16 x 8-bit samples from DE-Bus B
SP_Data_Val_B	1	Out	AC240 only. Data valid at FIFO output, channel B. It is strictly identical to **SP_Data_Val_A**.
Sp_Data_Addr	9	Out	9 bits, unused, reserved for Agilent.
StartStore	1	In	Starts the de_interface FIFO and output stream. This should be done only after the continuous acquisition mode setup is complete. See the dedicated paragraph below.
DE_Start	1	Out	Bit **DeStart** of the **DEControl** register. It must be connected to the input **StartStore**, either directly or delayed by the DCM locking time in case the firmware uses the core **ck_manager_sysclk_declk**.
Trig_Enb_Acq	1	Out	It must be connected to the corresponding input of the core **trigger_manager**.
StreamCK	1	In	Output Stream Clock. Can be any frequency equal or greater than 62.5 MHz (maximum frequency of the clocks **DECLKx**).

6.8.4 Output Stream Bus

The two data streams are fully coherent. The two signals **SP_Data_Val_A** and **SP_Data_Val_B** are always '1' simultaneously and '0' simultaneously.



For the AC240 dual channel mode (non-interleaved), **SP_Data_A** corresponds to the front-panel signal connector "INPUT2" while **SP_Data_B** corresponds to the front-panel signal connector "INPUT1".

For the AC210, **SP_Data_A** corresponds to the front-panel signal connector "INPUT".

The most significant bits (120 to 127) correspond to the first (oldest) acquired sample, and the lowest bits (0 to 7) correspond to the last acquired sample. Each value can be raw or signed, selectable with the bit **Unsigned** of the **de_control** register.

In the case of interleaved acquisition (AC240 in single channel mode), **SP_Data_A** and **SP_Data_B** are interleaved. All even samples (samples 0, 2, 4, 6 …30 of a 32-sample data block) are on **SP_Data_A** and all odd samples (samples 1, 3, 5, 7 …31) are on the bus **SP_Data_B**. The sample 0 is the oldest, i.e. the first acquired sample.

6.8.4.1 Data Source and Ordering

Module	Mode & Source	Sample oldest = 0	
AC240	Dual channel Source is **input1**	0 to 15	S(i) = SP_Data_B(127-(i*8) to 120-(i*8))
AC240	Dual channel Source is **input2**	0 to 15	S(i) = SP_Data_A(127-(i*8) to 120-(i*8))
AC240	Single channel Source is **input1** and **input2**	0 to 30, step 2	S(i) = SP_Data_A(127-(i*4) to 120-(i*4))
		1 to 31, step 2	S(i) = SP_Data_B(127-((i-1)*4) to 120-((i-1)*4))
AC210	Single channel Source is **input1**	0 to 15	S(i) = SP_Data_A(127-(i*8) to 120-(i*8))

6.8.4.2 ADC Code Correspondence

Position	Raw	Signed	
ADC top	0xFF	0x7F	Overflow, >= +Full Scale / 2 @ Offset = 0 V
ADC middle	0x80	0x00	Corresponds to 0V @ Offset = 0 V
ADC bottom	0x00	0x80	Underflo, <= -Full Scale / 2 @ Offset = 0 V

6.8.5 Registers

6.8.5.1 DEControl Register

Register Space	Register Number	Register Address
Customer	**8**	**0x2220**

31	30..24
DeStart	--

23	22	21	20	19..18	17	16

--	Replay	StopFillOnTr	DisableFill	DeRWMode	Gray2BinOff	Unsigned

15..0
DeAddress

[15..0] **DeAddress** **R** Indicate the trigger position or the automatic generated trigger position for the Base Design test mode. This address can be used as start address for reading the DE-Buffer. The value must be written to the Indirect Address Register prior to reading the DE-Buffer.

The value is valid for CHA, CHB or interleaved readout. The value must be multiplied by two in case of interleaved mode.

[16] **Unsigned** **RW** Select Unsigned / Signed format for **SP_Data_A** & **SP_Data_B**

0 Signed

1 Unsigned

[17] **Gray2BinOff** **RW** Disable Gray to binary conversion for **SP_Data_A** & **SP_Data_B**. Binary to gray should be disabled if you write data to the DE-Buffer with the intention of replaying the data.

0 **SP_Data_A** and **SP_Data_B** take on the values of the DE-Buffer with conversion from Gray code to binary. This is necessary for the ADC data stream that is coded in Gray.

1 **SP_Data_A** and **SP_Data_B** take on the values of the DE-Buffer without conversion from Gray to binary.

[19..18] **DeRWMode** **RW** Configure DE-Buffer for read and write operation

0 Channel A
0

0 Channel B (AC240 only)
1

1 Channel A and B Interleaved (A0, B0, A1, B1…).
0

1 Undefined
1

[20] **DisableFill** **RW** Stop writing the incoming data to the DE-Buffer. Should be set to ‘0’ in normal operation mode or ‘1 when replaying of the DE-Buffer is required.

[21] **StopFillOnTr** **RW** Special Test mode when set to ‘1’:

Input data are continuously stored to the DE-Buffer. The controller waits for a valid trigger. When a trigger occurs, the input data continue to be stored until the DE-Buffer is Full.

[22] **Replay** **RW** Special Test mode when set ‘1’:

The contents of the DE_Buffer are frozen and will be continuously replayed to **SP_Data_A** & **SP_Data_B**. The stream should be started. Replay uses the **declk** rate to output the stream.

[24] **Sp_Trig_Reorder** **RW** The state of the register bit **Sp_Trig_Reorder** is transferred to the ouput Sp_Trig_Reorder of the core (this is true only for the core **de_interface_2ch_rg** described in the next section)

[31] **DeStart** When set to ‘1’, enables the data stream from the DE-Buffers to **SP_Data_A** and **SP_Data_B**.

6.8.5.2 DE_Buffer Operating Mode

Use only the first mode (00) for normal operation!

DisableFill	StopFillOnTr	Operation
0	0	The trigger is controlled by the user core. The DE-Buffer streams data continuously. The IN-Buffer can be set to Triggered or non-Triggered mode.
1	0	BaseDesign Test mode: Untriggered operation When **`DisableFill`** is set '1', the core **`de_interface`** generates a trigger on **SP_Trigger** at the current DE-Buffer location and stops filling the DE-Buffer just before reaching again the same location, so the buffer is completely filled with new data. The output stream will continuously repeat the DE-Buffer contents with a trigger at the beginning of the waveform. The DE-Buffer has to be read from the position **DeAddress**. Filling will start again when **`DisableFill`** is set '0'.
1	1	BaseDesign Test mode: Triggered operation. The only difference with the mode 10 is the trigger. It is **not** automatically generated by the core **`de_interface`** but is a true trigger, derived from the input **`Formatted_Tracpt`**.

6.8.6 Accessing the DE-Buffer

The DE-Buffer contents can be read in burst mode using the Indirect Addressing Register. The DE-Buffer contains 8K samples per channel. The Indirect Address Register and the Buffer Identifier Register should be set prior to reading or writing the DE-Buffer.

6.8.6.1 DE-Buffer

Register Space	Register Number	Register Address	Buffer Identifier
Customer	**0**	**0x2200**	**0x08**

31..24	23..16	15..8	7..0
D3	D2	D1	D0

[31..0] **`D3 - D0`** **`RW`** The bytes D0- D3 each correspond to an 8-bit sample.

The format is signed or unsigned as defined by the **`DEControl`** register.

The order, big or little Endian, is configured in the Agilent general control register (see the core **`acq_ctr_reg`**).

6.8.7 Constraints

- Clock Constraint

This core assumes an **IB_Clk** frequency of 33 MHz. This constraint must be defined for the clock manager component and is automatically propagated throughout the whole design.

- **declkag** and **declkbg** Clock Constraint

The maximum frequency of DECLKA and DECLKB is 62.5 MHz, for a sample rate of 1 GS/s (non-interleaved), or 2 GS/s when the 2 channels of an AC240 are interleaved. This constraint must be defined for the clock manager component and is automatically propagated throughout the whole design.

- Input Signal Constraints

All FPGA inputs are registered. All input registers must be located within the IOB.

6.8.8 Resource Utilization

De_interface_1ch:
Resource count and relative usage in the target Xilinx Virtex II Pro – XC2VP70-6FF1517:

Resources	Used	Available	Utilization
Block RAMs	6	328	1.8%
Function Generators	1059	66176	1.6%
CLB Slices	611	33088	1.9%
Dffs or Latches	1222	69068	1.8%

De_interface_2ch:
Resource count and relative usage in the target Xilinx Virtex II Pro – XC2VP70-6FF1517:

Resources	Used	Available	Utilization
Block RAMs	10	328	3.0%
Function Generators	1754	66176	2.7%
CLB Slices	899	33088	2.7%
Dffs or Latches	1798	69068	2.6%

6.8.9 Version History

Date	FDK Version	Comments
April 05	Beta 1	Initial Version
June 05	Beta 2	Implemented write function to the DE-Buffer
July 05	Beta 3	Add Replay bit, make replay work
September 05	Beta 4	Make the IB-BUS access work with any frequency of StreamCK.
March 06	Beta7	Description updated.

6.9 DE Interface for SC240 and High Resolution Trigger

The core **de_interface_2ch_rg** is identical to the core **de_interface_2ch** except for the DE input clocking scheme that uses only the clock **declkbg** instead of the two clocks **declkbg** and **declkag**. This was necessary to be able to place and route all the clocks used for the streamer and the high resolution trigger.

Please read the previous section for a complete description. This section only contains the essentials.

NOTE: Because DCM are used for generation of all clocks **DeclkX**, this core should be used only for ADC sampling rate of 500 MS/s and 1 GS/s (in interleaved mode, this is equal to a sampling rate of 1 GS/s and 2 GS/s). For lower sampling rates, the DCM will unlock and the behavior will not be guaranteed. Lower sampling rate can be implemented by sparsing the data within the firmware.

6.9.1 Instantiation

The core **de_interface_2ch_rg** shall be used for streamer applications if the **trigger_manager_1ns** core is instantiated for the trigger.

6.9.2 Port Description

Signal	Size	Type	Short Description
INTERNAL BUS			
IB_Customer **IB_Dirsel** **IB_Write** **IB_Valid** **IB_Rdy** **IB_TimeO** **IB_End** **IB_IndirCtr** **IB_Addr** **IB_DataW** **IB_DataR**	1 1 1 1 1 1 1 32 12 32 32	In In In In Out In Out In In In Out	Should be connected to the IB-BUS signal with the same name. For details please refer to the description of the IB-BUS.
IB_Clk	1	In	Internal Bus Clock It must be connected to **lbclkg**, the Local Bus clock.
Reset	1	In	Reset It must be connected to the general reset **Dreset.**
DE BUS – Data input Stream			
MDA	64	In	Data input streamA, samples 0 to 7 of the incoming data block (gray coded)
MEA	64	In	Data input streamA, samples 8 to 15 of the incoming data block (gray coded)
MDB	64	In	SC240 only. Data input streamB, samples 0 to 7 of the incoming data block (gray coded)
MEB	64	In	SC240 only. Data input streamB, samples 8 to 15 of the incoming data block (gray coded)
declkbg	1	In	SC240 only. Data input streamB clock. **MDB** and **MEB** are clocked by the falling edge of the clock.
Formated_tracpt	1	In	After the trigger has been enabled (see details in the description of the core **trigger_manager_1ns**), indicates that a trigger occurred in the current data block. **Formated_tracpt** remains '1' for a single **declk** period. It must be connected to the equivalent signal out of the core **trigger_manager_1ns**.
Loc_tracpt	4	In	Location of the trigger within the block of 16 samples. This is available only if the trigger core

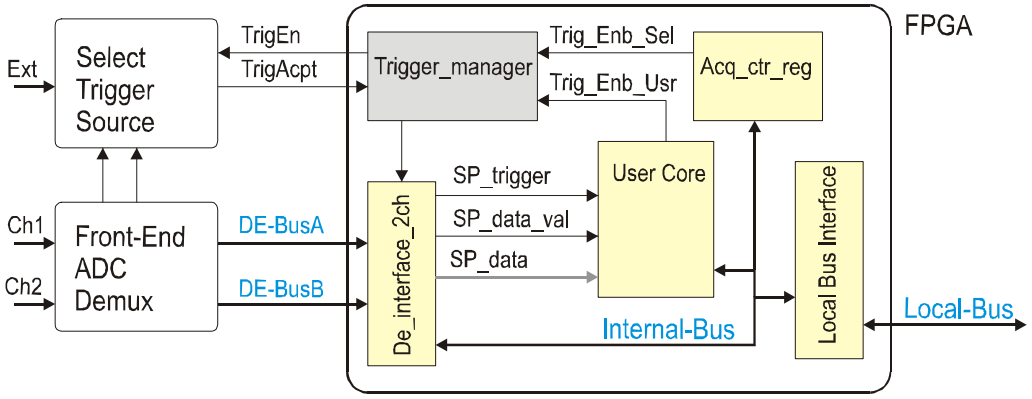
Signal	Size	Type	Short Description
			trigger_manager_1ns is used. It must be connected to the equivalent signal out of the trigger core.
Tracpt	1	In	Raw accepted trigger signal. It must be connected to the equivalent signal out of the trigger core.
BigEndian	1	In	Select format for FIFO readout to the application: '0': Little Endian '1': BigEndian
StopFill	1	In	Function not available, reserved for future use. It must be connected to '0'.
DeStopped	1	Out	A '1' denotes the **de_interface** is actually stopped. '0' when running.
OUTPUT STREAM			
SP_First_A	4	Out	4-bit trigger position in the data block. Only valid if the core **trigger_manager_1ns** is used.
SP_Trigger	1	Out	'1' indicates a trigger occurred in the current data block.
SP_Data_A	128	Out	FIFO output 128 bits, 16 x 8-bit samples from DE-Bus A
SP_Data_Val_A	1	Out	Data valid at FIFO output, channel A
SP_Data_B	128	Out	SC240 only. FIFO output 128 bits, 16 x 8-bit samples from DE-Bus B
SP_Data_Val_B	1	Out	SC240 only. Data valid at FIFO output for channel B. It is strictly identical to **SP_Data_Val_A**.
Sp_Data_Addr	9	Out	9 bits, unused, reserved for Agilent.
Sp_Trig_Reorder	1	Out	Trigger Reorder control. Takes the value of the bit **Sp_Trig_Reorder** of the register **DEControl Register** described in the previous section.
StartStore	1	In	Starts the de_interface FIFO and output stream. This should be done only after the continuous acquisition mode setup is complete. See the dedicated paragraph below.
DE_Start	1	Out	Bit **DE_Start** of the **DE_control** register. It must be connected to the input **StartStore**, either directly or delayed by the DCM locking time in case the firmware uses the core **ck_manager_sysclk_declk**.
Trig_Enb_Acq	1	Out	It must be connected to the corresponding input of the core **trigger_manager_1ns**.
StreamCK	1	In	Output Stream Clock. Can be any frequency equal or greater than 62.5 MHz (maximum frequency of the clocks **DECLKx**).

6.9.3 Version History

Date	FDK Version	Comments
January 07	1.0	New core for the streamer Base Design

6.10 Trigger Manager

The core **trigger_manager** generates a trigger derived from the trigger system on the module. The trigger can be one of these possible trigger sources: Ch1, Ch2, or External Trigger In.



6.10.1 Functional Description

The core **trigger_manager** controls if the trigger is enabled or not. It also formats the accepted trigger signal and forwards it to the **de_interface**. This is because the trigger signal has to be routed consistently through the DE_Buffer in order to keep the simultaneity between the incoming sample and the trigger signal.

The trigger does not necessarily have to be used. It only generates a marker in the data stream. It does not directly affect the data stream which remains continuous.

There are two differential trigger input signals: **TRIG_p** / **TRIG_n** and **TRIGA_p** / **TRIGA_n**. **TRIG** directly reflects the output of the trigger comparator, while **TRIGA** represents the 'accepted' trigger. **TRIGA** remains low as long as **TRIGEN** remains '0', i.e. the trigger accept circuit has not been enabled. Setting **TRIGEN** to '1' enables the trigger accept circuit. After enabling, the signal **TRIGA** becomes '1' at the first occurrence of a trigger. **TRIGA** then remains '1' and goes low only after **TRIGEN** is reset to '0'.

The core **trigger_manager** has the two trigger enable inputs **Trig_Enb_Acq**, **Trig_Enb_Usr**. The signal **Trig_Enb_Sel** selects which of the input trigger enables drives the trigger enable output **TRIGEN**.

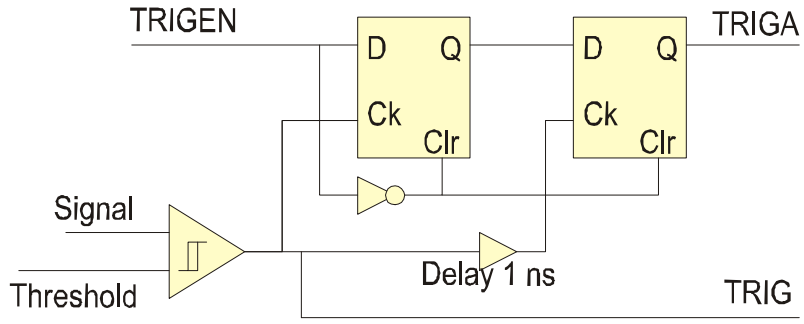
6.10.2 Port Description

Signal	Size	Type	Short Description
Reset	1	In	Reset, active '1'.
declk	1	In	Clock input. It is used to synchronize the signal **Formatted_Tracpt**.
TRIG_p / _n	2	In	Differential Raw trigger input, reflects the output of the trigger comparator.
TRIGA_p / _n	2	In	Accepted trigger. Trigger signal "gated" by the trigger enable signals (see below).
Trig_Enb_Sel	1	In	Select **TRIGEN** signal source: '0': **Trig_Enb_Usr** '1': **Trig_Enb_Acq**
Trig_Enb_Acq	1	In	Trigger Enable input reserved for Agilent
Trig_Enb_Usr	1	In	Trigger Enable input reserved for the developer
TRIGEN	1	Out	Trigger Enable to the trigger circuitry.
Trigger	1	Out	Raw trigger output

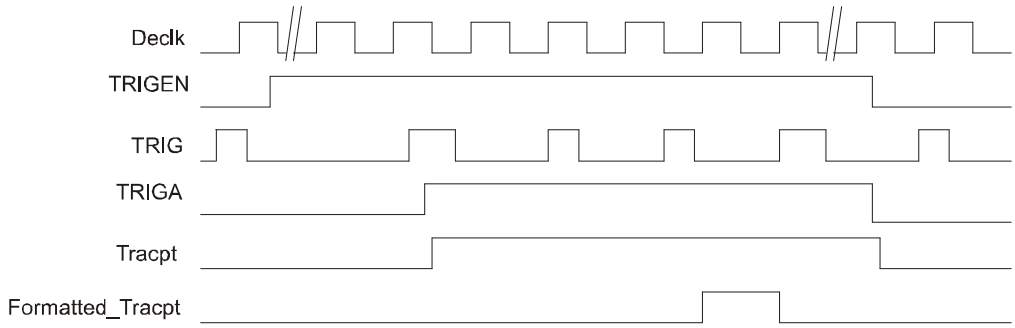
Signal	Size	Type	Short Description
Tracpt	1	Out	Accepted trigger
Loc_Tracpt	4	Out	Always 0x0000. It is the trigger position within the data block of 16 samples. Not used in this core.
Formatted_Tracpt	1	Out	Single pulse of width equal to one **declk** period, derived from the rising edge of **Tracpt**.

6.10.3 Trigger and Trigger Accept Circuit

The diagram below shows the trigger and accepted trigger as implemented on the module (outside the FPGA). This circuit is implemented with high speed logic in order to reduce the meta-stability of the accepted trigger.



6.10.4 Trigger Control Timing Diagram



6.10.5 Constraints

- **declk** Clock Constraint

This core assumes a **declk** frequency of maximum 62.5 MHz. This constraint must be defined for the clock manager component and is automatically propagated throughout the whole design.

- Input Signal Constraints

Timing for **TRIG** and **TRIGA** need not be constrained, but there is a location constraint due to the clock distribution. Thus the IOB of the **TRIGA** buffer can not be used with the **declkag** clock; otherwise the Place & Route path fails. The first register location is constrained to be in the nearest quarter where **declkag** is routed. Please refer to the *.sdc file for more details about the LOC constraints associated with the **TRIGA** input.

- Output Signal Constraints

TRIGEN output delay need not be constrained. The default LVTTL 12mA slow output driver is fine.

6.10.6 Resource Utilization

Resource count and relative usage in the target Xilinx Virtex II Pro – XC2VP70-6FF1517:

Resources	Used	Available	Utilization

Resources	Used	Available	Utilization
Function Generators	2	66176	~0%
CLB Slices	2	33088	~0%
Dffs or Latches	4	69068	~0%

6.10.7 Version History

Date	FDK Version	Comments
April 05	Beta 1	Initial Version
September 05	Beta 4	Compared to the data input stream, the trigger has been delayed by one clock period in order to fit the reality.
March 06	Beta 7	Description updated.

6.11 High Resolution Trigger Manager

The core **trigger_manager_1ns** generates a trigger derived from the trigger system on the module. The trigger can be one of these possible trigger sources: Ch1, Ch2 or External Trigger In. It is comparable to the core **trigger_manager** but with an improved resolution.

The core **trigger_manager_1ns** has two operating modes. The first mode replicates the function of the core **trigger_manager**. Please read the previous section for details about this mode. The second mode has an enhanced resolution in order to achieve a trigger resolution of one sample (inputs non-interleaved) or two samples (inputs interleaved).

6.11.1 Functional Description

The core **trigger_manager_1ns** controls if the trigger is enabled or not. It also formats the accepted trigger signal and forwards it to the **de_interface**. This is because the trigger signal has to be routed consistently through the DE_Buffer in order to keep the simultaneity between the incoming sample and the trigger signal.

The trigger does not necessarily have to be used. It only generates a marker along the data stream. It does not directly affect the data stream which remains continuous.

The two bits **TRM** of the Trigger Control Register control the trigger mode. There are four modes. The first mode (**TRM** set to '00') replicates the function of the core **trigger_manager**. The enhanced resolution mode is enabled when **TRM** is set to '01'. The two additional modes are reserved for Agilent verification purpose.

The core **trigger_manager** has the two trigger enable inputs **Trig_Enb_Acq**, **Trig_Enb_Usr**. The signal **Trig_Enb_Sel** selects which of the input trigger enables drives the trigger enable output **TRIGEN**.

There are two differential trigger input signals: **TRIG_p** / **TRIG_n** and **TRIGA_p** / **TRIGA_n**. **TRIG** directly reflects the output of the trigger comparator, while **TRIGA** represents the 'accepted' trigger. **TRIGA** remains low as long as **TRIGEN** remains '0', i.e. the trigger accept circuit has not been enabled. Setting **TRIGEN** to '1' enables the trigger accept circuit. After enabling, the signal **TRIGA** becomes '1' at the first occurrence of a trigger. **TRIGA** then remains '1' and goes low only after **TRIGEN** is reset to '0'.

After **TRIGA** rises to '1', the output signal **Formatted_Tracpt** will rise to '1' and remains '1' for one **declk** period. The signal **Loc_Tracpt** will indicate the sample (in the bloc: 0 to 15) at which the trigger occurred.

The core delivers a timestamp with a resolution equal to the resolution of the trigger. The value is updated when **TRIGA** rises to '1'.

The signals **Formatted_Tracpt_dly** and **Loc_Tracpt_dly** are the signals **Formatted_Tracpt** and **Loc_Tracpt** digitally delayed by a programmable value (**TRDL** of the register **TriggerDelay**) with a resolution equal to one sample. This is useful for adjusting the position of the trigger relative to the data.

The Status of the core can be interogated in a program by reading the status registers **Trigger Status Lo** and **Trigger Status Hi**.

6.11.2 Port Description

Signal	Size	Type	Short Description
Trigger Control IO			
declk	1	In	Clock input. It is used to synchronize the signal **Formatted_Tracpt**.
TRIG_p / _n	2	In	Differential Raw trigger input, reflects the output of the trigger comparator.
TRIGA_p / _n	2	In	Accepted trigger. Trigger signal "gated" by the TRIGEN signal (see below).
Trig_Enb_Sel	1	In	Select **TRIGEN** signal source: '0': **Trig_Enb_Usr** '1': **Trig_Enb_Acq**

Signal	Size	Type	Short Description
Trig_Enb_Acq	1	In	Trigger Enable input reserved for Agilent
Trig_Enb_Usr	1	In	Trigger Enable input reserved for the developer
TRIGEN	1	Out	Trigger Enable to the trigger circuitry.
Trigger	1	Out	Raw trigger output
Tracpt	1	Out	Accepted trigger
Loc_Tracpt	4	Out	Trigger position within the data block of 16 samples (fixed to '0000' for the low resolution trigger mode).
Formatted_Tracpt	1	Out	Single pulse of width equal to one **declk** period, derived from the rising edge of **Tracpt**.
Loc_Tracpt_dly	4	Out	**Loc_Tracpt_dly** takes the value: **Loc_Tracpt** + **TRDL** modulo 16. **Loc_Tracpt_dly** is delayed in time as **Formatted_Tracpt_dly** is delayed.
Formatted_Tracpt_dly	1	Out	**Formatted_Tracpt_dly** delayed by N **declk** periods. Where N is equal to the truncated integer part of: (**TRDL** + **Loc_Tracpt**) / 16
INTERNAL BUS			
IB_Select **IB_Dirsel** **IB_Write** **IB_Rdy** **IB_TimeO** **IB_Addr** **IB_DataW** **IB_DataR** **IB_Clk**	1 1 1 1 1 12 32 32 1	In In In Out In In In Out In	Should be connected to corresponding signal on the IB-BUS. For details please refer to the description of the IB-BUS.
Reset	1	In	Reset, active '1'.

6.11.3 Registers

6.11.3.1 Trigger Control Register

NOTE Users are only allowed to control the bit **TRDI** to reset the DCM (in order to get them locked), the bits **TRM** to set the trigger mode and the bit **TRTC** to reset the timestamp. All other bits shall remain '0'.

Register Space	Register Number	Register Address
Customer	**12**	**0x2230**

31	30..25	24
TRPM	TRPH	TRPE

23..19	18..8	7	3	2	0..1
--	SDEL	ESDS	TRTC	TRDI	TRM

[1..0] **TRM** **RW** Trigger Mode

00 Low resolution trigger. The resolution is 16 samples when the channels are not interleaved or 32 samples in case of the SC240 in interleaved channel mode.

01 High resolution trigger. . The resolution is 1 sample when the channels are not interleaved or 2 samples in case of the SC240 in interleaved channel mode.

The high resolution trigger is valid only for a sampling rate of 1 GS/s.

`10` Automatic Asynchronous Trigger based on the local bus clock. For test purpose.

`11` Synchronous Trigger for test purpose. Needs the external clock and trigger generator.

[2] **TRDI** **RW** Initialisation of the DCM

[3] **TRTC** **RW** Enable the Restart the Timestamp Trigger counter to 0

[7] **ESDS** **RW** Enable Shift out the synchronous delay

[18..8] **SDEL** **RW** Delay for synchronous trigger test. Needs the external board for clock and trigger generation.

[24] **TRPE** **RW** Enable Trigger fine phase correction:

`0` Phase correction disabled

`1` Phase correction enabled

[30..25] **TRPH** **RW** Initial phase value: unsigned 0 to 64

[31] **TRPM** **RW** Phase increment / decrement

`0` Decrement phase relative by **TRPH**

`1` Increment phase relative by **TRPH**

6.11.3.2 Trigger Status Lo

This register is used to retrieve the trigger status and the lower part of the trigger timestamp value

Register Space	Register Number	Register Address
Customer	**13**	**0x2234**

31..8	7..4
TSTL	--

3	2	2	1
PDN	GRN	PDN	TRO

[0] **TRO** **R** Trigger Occurred

`0` No Trigger

`1` A trigger has occurred

[1] **PDN** **R** Set '1' at the end of the DCM calibration phase

[2] **GRN** **R** '1' indicates the gray value of the trigger interpolator was not valid

[3] **LCK** **R** '1' denotes the trigger DCMs are locked

[31..8] **TSTL** **R** Bit 23 to 0 of the Trigger Timestamp.
The value is Valid if **GRN** is '0' and **TRO** is '1'

6.11.3.3 Trigger Status Hi

This register is used to retrieve the upper part of the trigger Timestamp value.

Register Space	Register Number	Register Address
Customer	**14**	**0x2238**

31..0
TSTH

[31..0] **TSTH** **R** Bit 55 to 24 of the Trigger Timestamp.
This value is Valid if **GRN** is '0' and **TRO** is '1'.

6.11.3.4 Trigger Delay

This register could be used to delay the trigger event relative to the data. The resolution is one sample when the module operates in dual channel mode or two samples when the two channels are interlaced to double the sample rate.

Register Space	Register Number	Register Address
Customer	**15**	**0x223C**

31..8	6..0
	TRDL

[6..0] **TRDL** **RW** Trigger Compensation. The Trigger event is shifted forward, relative to the data stream, by a number of samples equal to the value of **TRDL**.

The valid range is 0 to 127.

6.11.4 Constraints

- declk Clock Constraint

This core assumes a **declk** frequency of minimum 31.25 MHz and maximum 62.5 MHz. This constraint must be defined for the clock manager component and is automatically propagated throughout the whole design.

- Input Signal Constraints

In order to ensure the same delay of the **TRIGA** input across successives implementations, a location constraint is applied to the registers TraQ1 through TraQ4.

There are other crucial placement constraints to insure the resolution and linearity of the trigger positioning.

Please refer to the *.sdc file for more details about the LOC constraints associated with this core.

- Output Signal Constraints

TRIGEN output delay need not be constrained. The default LVTTL 12mA slow output driver is fine.

6.11.5 Resource Utilization

Resource count and relative usage in the target Xilinx Virtex II Pro – XC2VP70-6FF1517:

Resources	Used	Available	Utilization
Global Buffers (BUFG)	1	16	6.25 %
Function Generators	300	66176	0.45%
CLB Slices	216	33088	0.65%
Dffs or Latches	431	69068	0.62%

6.11.6 Version History

Date	FDK Version	Comments
January 07	1.0	New core for the streamer Base Design

6.12 Acqiris Register

6.12.1 Functional Description

The core **acq_ctr_reg** implements a control register and a status register that must be preserved in all designs.

The Control register is used to control the clocking schemes, to define the data format during the readout and to enable the use of the processing interrupt.

The Status register contains the status of clocks (DCM locked), the status of the trigger enable line, the status of the temperature alarm, and the status of the user core (Core_started).

Since this core is primarily intended for controlling the clocking resources, the user should refer to the description of the clock manager cores for more details about the clocking scheme.

6.12.2 Port Description

Signal	Size	Type	Short Description
INTERNAL BUS			
IB_Acqiris **IB_Customer** **IB_Dirsel** **IB_Write** **IB_Rdy** **IB_TimeO** **IB_Addr** **IB_DataW** **IB_DataR**	1 1 1 1 1 1 12 32 32	In In In In Out In In In Out	Must be connected to the IB-BUS signal with the same name. For details please refer to the description of the IB-BUS.
IB_Clk	1	In	Internal Bus clock, must be connected to the Local Bus clock **lbclkg**
Reset	1	In	Reset, must be connected to the general reset **Dreset**
Control - Out			
INTERRUPT_ENABLE	1	Out	Enable Interrupt to Local Bus
Trig_Enb_Sel	1	Out	Select trigger Enable source. It must be connected to the equivalent input of the core **trigger_manager**.
Enb_DCM	8	Out	8-bits DCM Enable bus. It must be connected to the **Enb_DCM** input of the clock manager instance.
Sel_Fsysclk	1	Out	Sysclk clock mux selection. It must be connected to the **Sel_Fsysclk** input of **ck_rst_manager_sysclk**.
SGReset_n	1	Out	Software General reset. Active after configuration. It must be set to '1' for de-activation.
Status - In			
TRIGGER_ENABLE	1	In	Status of the trigger enable signal for output from the user core. It must be connected to the equivalent output of the user core. Otherwise should be kept at '0'.
Core Started	1	In	Status of the acquisition. Normally connected to the output **acquire** of the user core.
Lck_DCM	8	In	Status of each DCM (lock line). It must be connected to **Lck_DCM** output of the clock manager instance.
Tmp_Alarm	1	In	Status of the Temperature Alarm. It must be connected to the corresponding output of the core **acq_tmp_struct**.

6.12.3 Registers

6.12.3.1 AcqirisPrivateControl Register

This register is located in the Agilent Space. There is no driver function available for developers to access this register.

Register Space	Register Number	Register Address
Agilent	**64**	**0x2100**

31..1	0
--	SGReset_n

[0] **SGRseset_n** **RW** Software General reset. Active low. Default value = '0'. It is de-asserted by the driver after new bit files are load to the FPGA. This bit is forwarded to the output **SGReset_n**.

6.12.3.2 AcqirisControl Register

Register Space	Register Number	Register Address
Customer	**3**	**0x220C**

31	30..28	27..25	24	23..16
SGReset_n			Sel_Fsysclk	End_Dcm

15..9	8	7..4	3..2	1	0
	BigEndian			TrigEnSel	IntEn

[0] **IntEn** **RW** Bit forwarded to the output **Interrupt_Enable**. There is more detail on how to manage interrupts in the chapter about the base designs. It must be set to '1' to enable the interrupt.

[1] **TrigEnSel** **RW** Source selection for trigger enable. It is forwarded to the output **Trig_Enb_Sel**.

0 User Trigger Enable

1 Acqiris Trigger Enable

[8] **BigEndian** **RW** Selection of the readout format: Big Endian / Little Endian. This bit is forwarded to the output **BigEndian**.

0 Little Endian

1 Big Endian

[23..16] **Enb_Dcm** **RW** Enable DCM 0 to 7. See details in the port description of the used clock manager core. These bits are forwarded to the output **End_Dcm**.

[24] **Sel_Fsysclk** **RW** Reserved for future use

[31] **Reserved** **RW** No function defined. It must be set '0'.

6.12.3.3 AcqirisStatus Register

Register Space	Register Number	Register Address
Customer	**6**	**0x2218**

31	30..28	27..25	24	23..16
				Lck_Dcm

15	14..8	7..3	2	1	0
Tmp_Alarm			TrigEn		CoreStarted

[0]	**CoreStarted**	**R**	Core Started. This bit is driven by the input signal **Core_Started**.
[2]	**TrigEn**	**R**	Trigger Enable. This bit is driven by the input signal **Trigger_Enable**.
[15]	**Tmp_Alarm**	**R**	Temperature Alarm. This bit is driven by the input signal **Tmp_Alarm**.
[16..23]	**Lck_Dcm**	**R**	DCM 0 to 7 lock status. '1' Locked. See details in the port description of the used clock manager core. These bits are driven by the input signals **Lck_Dcm**.

6.12.4 Constraints

- **IB_Clk** Clock Constraint

This core assumes an IB_Clk frequency of 33 MHz. This constraint must be defined for the clock manager component and is automatically propagated throughout the whole design.

6.12.5 Resource Utilization

Resource count and relative usage in the target Xilinx Virtex II Pro – XC2VP70-6FF1517:

Resources	Used	Available	Utilization
Function Generators	59	66176	0.1%
CLB Slices	38	33088	0.1%
Dffs or Latches	75	69068	0.1%

6.12.6 Version History

Date	FDK Version	Comments
April 05	Beta 1	Initial Version
July 05	Beta 3	Updated Version.
February 06	Beta 6	Added Software General Reset output.
March 06	Beta 7	Description updated.

6.13 LED Interface

6.13.1 Functional Description

The LED interface core **led_interface** provides control of 2 front panel LEDs of the AC/SC2x0 modules, either via software through the Internal Bus or directly from the User firmware through dedicated ports.

These bicolor LEDs are labeled L1 and L2 on the module's front panel. Each LED can be controlled independently by defining the code of the color that should be displayed. Are available: black (or switched off), red, green, and orange.

6.13.2 Port Description

Port Name	Size	Type	Description
IB_Customer **IB_Dirsel** **IB_Write** **IB_Rdy** **IB_TimeO** **IB_Addr** **IB_DataW** **IB_DataR**	1 1 1 1 1 32 32 32	In In In Out In In In Out	Must be connected to the IB-BUS signal with the same name. For details please refer to the description of the IB-BUS.
IB_Clk	1	In	Internal Bus Clock
Reset	1	In	Start Up Reset
Ll1_CCd	2	In	L1 LED Color Code, if not overridden by LED Register
Ll2_CCd	2	In	L2 LED Color Code, if not overridden by LED Register
Ll1_Green	1	Out	Drives L1 Green LEDs Anode and L1 Red LEDs Cathode
Ll1_Red	1	Out	Drives L1 Red LEDs Anode and L1 Green LEDs Cathode
Ll2_Green	1	Out	Drives L2 Green LEDs Anode and L2 Red LEDs Cathode
Ll2_Red	1	Out	Drives L2 Red LEDs Anode and L2 Green LEDs Cathode

6.13.3 Detailed Description

There are two different ways of controlling the LED interface. Each LED color can be defined either by using the LCOL field of the ***REGISTER*** or by driving the Ll1_CCd and Ll2_CCd according to ***TABLE*** 2.

By default the LEDs are controlled by the firmware. It implies that the LEDs color is defined by the state of the Ll1_CCd and Ll2_CCd busses if the software does not override it. The default "00" value displays the red color. This can be observed whenever the FPGA is loaded with the default base design firmware.

Color Code	LED Color
00	Red
01	Green
10	Yellow
11	Black (Switched OFF)

Table 2 LED Color Code

The User application can override the color code driven by the firmware by setting the **LMD** fields in the ***REGISTER*** to 1. In this case the LEDs color is driven by the associated LCOL field whatever the value of Ll1_CCd/Ll2_CCd ports.

The LED Interface Core is an open core (VHDL source is available) that should be instantiated in any firmware using the FDK framework, in order to allow remote testing of the LED features.

6.13.4 Register

The LED Register is mapped to a fixed location (0x2288) within the Customer Reserved FPGA Register space and is available whenever the LED Interface Core is used.

6.13.4.1 LED Control

Register Space	Register Number	Register Address
Customer	**34**	**0x2288**

7	6	5..4	3	2	1..0
LMD2		LCOL2	LMD1		LCOL1

[1..0] **LCOL1** **RW** L1 LED Color Control
'00' Red, '01' Green, '10' Yellow, '11' Black

[3] **LMD1** **RW** Selects if the User Core or the user program controls LED1

0 LED Interface under User Core control

1 Remote control by software

[5..4] **LCOL2** **RW** L2 LED Color Control
'00' Red, '01' Green, '10' Yellow, '11' Black

[7] **LMD2** **RW** Selects if the User Core or the user program controls LED2

0 LED Interface under User Core control

1 Remote control by software

6.13.5 Constraints

- Clock Constraint

This core assumes an **IB_Clk** frequency of 33 MHz. This constraint must be defined for the clock manager component and is automatically propagated throughout the whole design.

- Input/Output Constraint

This core contains four outputs (Ll1_Green, Ll1_Red, Ll2_Green, Ll2_Red) that must be connected via an OBUF to the IO Pad. Please refer to the ac240.ucf file for LOC constraints. The IOSTANDARD attributes for these buffers shall be "LVCMOS33" with default drive strength (12mA) and slew rate (Slow) values.

6.13.6 Resource Utilization

Resource count and relative usage in the target Xilinx Virtex II Pro – XC2VP70-6FF1517:

Resources	Used	Available	Utilization
Global Buffers	0	16	0.00%
Function Generators	54	66176	0.1%
CLB Slices	47	33088	0.1%
Dffs or Latches	93	69068	0.1%

6.13.7 Version History

Date	FDK Version	Comments
April 05	Beta 1	Initial Version
March 06	Beta 7	Description updated.

6.14 PIO Interface

6.14.1 Functional Description

The core **pio_interface** provides control of the 2 front panel I/O P1 and I/O P2 MMCX connectors of the AC/SC2x0 modules, either via software through the Internal Bus or directly from the User firmware through dedicated ports.

Each connector can independently be defined as an input or an output. If configured as output and if the Internal Bus Control is used, each connector can output one of 64 internal FPGA signals either for external control or for debugging.

Half of the multiplexer capability is reserved for signals defined by Agilent to ensure remote testing whereas the other half is left for the user application or debugging task.

6.14.2 Port Description

Port Name	Size	Type	Description
IO_Fct_Usr	32	In	User-defined signals to be multiplexed on the PIO outputs when using register control
IO_Fct_Acq	32	In	Agilent-defined signals to be multiplexed on the PIO outputs when using register control
Io1_dir	1	In	Direction control for the IO1 line if not overridden by register control
Io1_in	1	Out	Input from the IO1 line if not overridden by register control
Io1_out	1	In	Output to the IO1 line if not overridden by register control
Io2_dir	1	In	Direction control for the IO2 line if not overridden by register control
Io2_in	1	Out	Input from IO2 line if not overridden by register control
Io2_out	1	In	Signal to output on IO2 line if not overridden by register control
IB_Customer **IB_Dirsel** **IB_Write** **IB_Rdy** **IB_TimeO** **IB_Addr** **IB_DataW** **IB_DataR**	1 1 1 1 1 32 32 32	In In In Out In In In Out	Must be connected to the IB-BUS signal with the same name. For details please refer to the description of the IB-BUS.
IB_Clk	1	In	Internal Bus Clock
Reset	1	In	Start Up Reset
Pio1_in	1	In	Input line from IO1 bidirectional buffer
Pio1_out	1	Out	Output line to IO1 bidirectional buffer
Pio1_dir	1	Out	Direction control of IO1 bidirectional buffer
Pio2_in	1	In	Input line from IO2 bidirectional buffer
Pio2_out	1	Out	Output line to IO2 bidirectional buffer
Pio2_dir	1	Out	Direction control of IO2 bidirectional buffer

6.14.3 Detailed Description

There are two different ways of controlling the front panel IO interface. Each of the 2 IO lines can be controlled either by the user core (default) or by using the PIO Control Register.

By default the front panel IO lines are controlled by the firmware. It implies that the Pio1_xx and Pio2_xx signals are driven by the Io1_xx and Io2_xx signals if the software does not override it.

If the PIO control is transferred to the PIO Control Register (by setting **IOMDi** to '1', for line i), the output value depends both on the signals connected to the **IO_Fct_Acq** and **IO_Fct_Usr** ports and on

the value of the **IOFi** field. Each port is configured independently but the signals that can be multiplexed are shared between the two PIO lines.

This multiplexer feature is primarily intended to ease firmware debugging and to allow remote support and testing by enabling the output of signals of interest. As the **IOFi** field is 6 bits wide, up to 64 different signals can be connected to the PIO Lines. The first 32 are named **IO_Fct_Acq** and are reserved for use by Agilent. The table below shows the current signal allocation. In the future, it will be completed with signals of interest for the support.

IOF Value	Signals	Comments
0	'0'	For production test
1	'1'	For production test
2	'0'	
3	'0'	
4	Raw trigger	Trigger Signal at pad level
5	Formated_Trigger	Trigger Accepted Signal
6	SP_Trigger	Trigger Signal at User Core Level
7	Trigger_enable	User Trigger Enable
8	Sysclk	System Clock
9	SP_Data_Val_A	MACA Data Valid
10	SP_Data_Val_B	MACB Data Valid
11 - 31	Reserved	

Table 3 : PIO Agilent Predefined Signals

The other 32 entries refer to the **IO_Fct_Usr** ports and can be used to monitor signals of interest from the user's core.

Warning: Do not connect clocks to the **IO_Fct_Usr** port! It could generate mapping errors due to clock placement constraints that cannot be satisfied when trying to output the clock signal on a PIO line.

6.14.4 Register

The PIO Control Register is mapped to a fixed location (0x2280) within the Customer Reserved FPGA Register space and is available whenever the PIO Interface Core is used.

6.14.4.1 PIO Control

Register Space	Register Number	Register Address
Customer	**32**	**0x2280**

31..18	17	16	15	14	13..8	7	6	5..0
	INIO2	INIO1	IOMD2	IODIR2	IOF2	IOMD1	IODIR1	IOF1

[5..0] **IOF1** **RW** IO Function to be sent on PIO1 while configured as an output, i.e. when IOMD1 = 1 and IODIR1 = 1. See Table 3.

[6] **IODIR1** **RW** Defines the direction of PIO1 buffer (0: In / 1: Out)

[7] **IOMD1** **RW** Defines the management mode of PIO1

- **0** PIO1 is managed by the User Core (default)
- **1** PIO1 is managed by this register

[13..8] **IOF2** **RW** IO Function to be sent on PIO2 while configured as an output, i.e. when IOMD2 = 1 and IODIR2 = 1. See Table 3.

[14] **IODIR2** **RW** Defines the direction of PIO2 buffer (0: In / 1: Out)

[15] **IOMD2** **RW** Defines the management mode of PIO2

- **0** PIO2 is managed by the User Core (default)

		1	PIO2 is managed by this register
[16]	**INIO1**	**R**	Current value on the PIO1 line, if IOMD1 = 1.
[17]	**INIO2**	**R**	Current value on the PIO2 line, if IOMD2 = 1.

6.14.5 Instantiation

This core is already instantiated in the base design example.

It only requires adding external pad buffer for the external (from the FPGA point of view) signals. Although there may be several ways to insert I/O buffers, the explicit IOBUF instantiation is the solution preferred by Agilent.

It is highly recommended to connect **IO_Fct_Acq** as specified in Table 3

6.14.6 Constraints

- Clock Constraint

This core assumes an IB_Clk frequency of 33 MHz. This constraint must be defined for the clock manager component and is automatically propagated throughout the whole design.

- Input/Output Constraints

This core contains two outputs (**pio1_dir**, **pio2_dir**) and two bidirectional signals that must be connected to the IO Pad via an OBUF/ IOBUF. The IOSTANDARD attributes for these buffers should be "LVCMOS33" with default drive strength (12mA) and slew rate (Slow) values.

6.14.7 Resource Utilization

Resource count and relative usage in the target Xilinx Virtex II Pro – XC2VP70-6FF1517:

Resources	Used	Available	Utilization
Function Generators	114	66176	0.2%
CLB Slices	57	33088	0.2%
Dffs or Latches	70	69068	0.1%

6.14.8 Version History

Date	FDK Version	Comments
April 05	Beta 1	Initial Version
March 06	Beta 7	Description updated.

6.15 Temperature Interface

6.15.1 Functional Description

The core **acq_tmp_struct** provides control of the temperature monitoring chip that senses the FPGA temperature diode integrated within the FPGA device. When enabled it performs a sense cycle every 8 seconds. It can generate an alarm if the observed temperature is greater than a programmable threshold.

6.15.2 Port Description

Port Name	Size	Type	Description
IB_Customer **IB_Dirsel** **IB_Write** **IB_Rdy** **IB_TimeO** **IB_Addr** **IB_DataW** **IB_DataR**	1 1 1 1 1 32 32 32	In In In Out In In In Out	Must be connected to the IB-BUS signal with the same name. For details please refer to the description of the IB-BUS
IB_Clk	1	In	Internal Bus Clock
Reset	1	In	Start Up Reset
Tmp_Alarm	1	Out	Temperature Alarm
Tmp_Sclk	1	Out	Temperature Monitoring Serial Clock
Tmp_Sel	1	Out	Temperature Monitoring Serial Line Selection
Tmp_Sdata	1	In	Temperature Monitoring Serial Data

6.15.3 Detailed Description

When enabled by writing at 1 to the **TMPE** bit of the TempMonitor register, a new value is automatically read every 8 second. The last value read is available as the lower 13 bits of the TempMonitor register.

The temperature value is formatted as a 13-bit field that contains the sign at the leftmost position and the absolute value on the 12 lower bits. The temperature resolution is 0.0625°C. Thus reading back a value of 0x822F would correspond to an FPGA temperature of 0x22F * 0.0625 = + 34.9375°C.

The temperature monitoring core lets the user define a programmable temperature threshold. If the current temperature exceeds the programmed temperature threshold field and if **ALAE** is set, the **Tmp_alarm** event is triggered.

Temperature monitoring is highly recommended, especially in a processing intensive context, in order to avoid damage to the FPGA. It should be performed either by monitoring the temperature register with software or directly within the firmware by using the **Tmp_Alarm** signal to halt the user core activity.

6.15.4 Register

The TempMonitor Register is mapped to a fixed location (0x221C) within the Customer Reserved FPGA Register space and is available whenever the Temperature Interface Core is used.

It is used to retrieve the internal temperature of the Data Processing Unit and to set the temperature threshold for the temperature alarm.

6.15.4.1 TempMonitor

Register Space	Register Number	Register Address
Customer	**7**	**0x221C**

31	30..29	28..16	15	14..13	12..0
ALAE	--	Tmp_Threshold	TMPE	--	TMP_Monitor

[12..0] **TMP_Monitor** **R** FPGA temperature when monitoring is enabled

[15]	**TMPE**	**RW**	FPGA Temperature Monitoring Enable
[28..16]	**Tmp_Threshold**	**RW**	FPGA Temperature Threshold for user Temp Alarm
[31]	**ALAE**	**RW**	FPGA Temperature Alarm Enable

6.15.5 Constraints

- Clock Constraint

This core assumes an IB_Clk frequency of 33 MHz. This constraint must be defined for the clock manager component and is automatically propagated throughout the whole design.

- Input/Output Constraint

This core contains two outputs (**Tmp_Sclk**, **Tmp_Sel**) and one input (**Tmp_Data**) that must be connected to the IO Pads via an OBUF/ IBUF. The IOSTANDARD attributes for these buffers should be "LVCMOS33" with default drive strength (12mA) and slew rate (Slow) values.

6.15.6 Resource Utilization

Resource count and relative usage in the target Xilinx Virtex II Pro – XC2VP70-6FF1517:

Resources	Used	Available	Utilization
Function Generators	132	66176	0.2%
CLB Slices	76	33088	0.2%
Dffs or Latches	152	69068	0.2%

6.15.7 Version History

Date	FDK Version	Comments
April 05	Beta 1	Initial Version
March 06	Beta 7	Description updated

6.16 DAC Interface

6.16.1 Functional Description

The core **dac_interface** provides control of the 16-bit Digital to Analog Converter (DAC) that drives the analog output (ANL OUT) on the front panel MMCX connector, either via software through the Internal Bus or directly from the User firmware through dedicated ports.

This analog signal can be driven within a [-5V to +5V] range and has rise/fall times faster than 500ns, the DAC settling time being specified at 1us.

There are three predefined test patterns that are already implemented within the DAC Interface Core: a positive square waveform, a full scale square waveform, and a rising ramp over the full scale range.

6.16.2 Port Description

Port Name	Size	Type	Description
DAC_DIN	16	In	16-bit data word to be output to the DAC
DAC_WR	1	In	Data Write Request
DAC_BUSY	1	Out	DAC Interface Availability (='0')
DAC_DONE	1	Out	DAC Interface Conversion Completion ('1')
IB_Customer **IB_Dirsel** **IB_Write** **IB_Rdy** **IB_TimeO** **IB_Addr** **IB_DataW** **IB_DataR**	1 1 1 1 1 32 32 32	In In In Out In In In Out	Must be connected to the IB-BUS signal with the same name. For details please refer to the description of the IB-BUS.
IB_Clk	1	In	Internal Bus Clock
Reset	1	In	Start Up Reset
SCS_N	1	Out	DAC Chip Select
SDATA	1	Out	DAC Serial Data Line
SCLR_N	1	Out	DAC Clear
SCK	1	Out	DAC Serial Clock

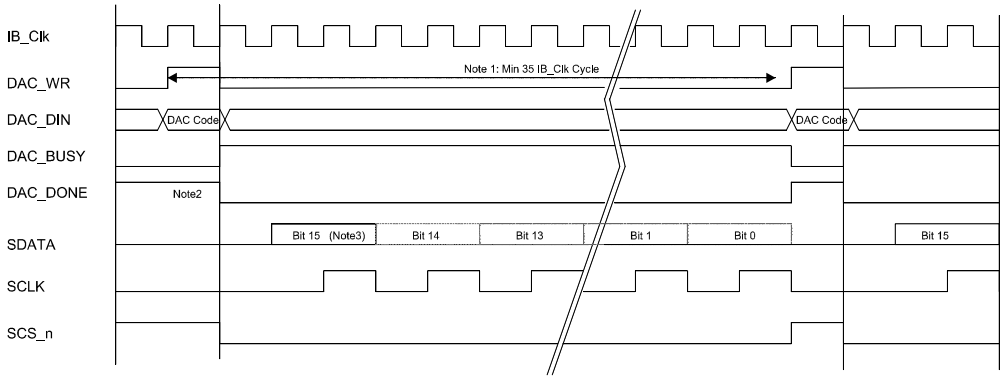
6.16.3 Detailed Description

There are two different ways of controlling the DAC interface. It can be controlled either by the user core (default) or by using the DAC Control Register.

By default, the user core drives the DAC interface by looking at its availability (**DAC_BUSY** should be negated), writing the 16-bit data code to **DAC_DIN,** and asserting the **DAC_WR** request. In response, the DAC Interface asserts the **DAC_BUSY** signal to prevent further write requests and serializes the data onto the DAC serial interface. The DAC Interface announces termination of the serial transfer, i.e. loading of the DAC, by asserting the **DAC_DONE** signal and simultaneously negating the **DAC_BUSY** signal (see the following timing diagram).

The theoretical voltage is defined by

$$Vout = \left(\frac{DAC\_DIN - 32768}{65536} \right) * 10 \cdot V$$



Note 1: Design limitation due to the use of **IB_Clk**/2 to generate **SCLK**.

Note 2: **DAC_DONE** remains high until the next **DAC_WR** request.

Note 3: Data are shifted out starting from the Most Significant Bit.

The next DAC value can be accepted no earlier than 35 **IB_Clk** cycles (1050ns) after the previous write to **DAC_IN**. This limitation derives from the fact that the DAC Interface core uses a serial clock at half the **IB_Clk** frequency (16.67 MHz) because the DAC device cannot sustain serial clocks higher than 25 MHz. The full scale settling time of the DAC is typically 1µs.

The DAC Interface can also be controlled by the DAC Control Register by writing 1 to the **DMOD** field. The 16-bit DAC value is defined by the **DAC_VAL** field and the **DSND** field acts as the **DAC_WR** port. In this mode, the DAC Interface provides three test patterns that can be selected with the **TPAT** field. By default, the DAC Interface uses the **DAC_VAL** value. The table below presents the test patterns that are automatically generated on the DAC output when the **TPAT** field is different from "00" and **DMOD** = 1.

TPAT Value	Test Pattern	Comments
00	Level defined by DAC_VAL	Sent upon write to **DSND** field
01	Positive square signal	0-5V / f= 246 KHz
10	Bipolar square signal	-5V to + 5V, f= 246 KHz
11	Ramp (-5V to 5V)	-5V to +5V, f= 14.96 Hz

Table 4 : DAC Test Patterns

6.16.4 Register

The DAC Control Register is mapped to a fixed location (0x2284) within the Customer Reserved FPGA Register space and is available whenever the DAC Interface Core is used.

6.16.4.1 DAC Control

Register Space	Register Number	Register Address
Customer	**33**	**0x2284**

31..20	21..20	19	18	17	16	15..0
	TPAT	DMOD	DSND	DUSY	DONE	DAC_VAL

[15..0]	**DAC_VAL**	**RW**	DAC value If **DMOD** = '1', it is loaded by the software and sent to the DAC If **DMOD** = '0', it is the last value loaded by the user core
[16]	**DONE**	**R**	'1' = DAC Interface has completed the last conversion
[17]	**BUSY**	**R**	'1' = DAC Interface not available (initializing or still converting)
[18]	**DSND**	**RW**	Send the word defined by **DAC_VAL** to the DAC interface when

			DMOD =’1’.
[19]	**DMOD**	**RW**	Defines the mode used for the DAC interface control: **0** DAC under User Core control **1** Remote control by software
[21..20]	**TPAT**		Defines the test pattern to be generated after configuration to remote control (**DMOD** = ‘1’)

6.16.5 Constraints

- Clock Constraint

This core assumes an IB_Clk frequency of 33 MHz. This constraint must be defined for the clock manager component and is automatically propagated throughout the whole design.

- Input/Output Constraints

This core contains four outputs (**SCS_N, SCLR_N, SDATA, SCLK**) that must be connected via an OBUF to the IO. The IOSTANDARD attributes for these buffers should be "LVCMOS33" with default drive strength (12mA) and slew rate (Slow) values.

6.16.6 Resource Utilization

Resource count and relative usage in the target Xilinx Virtex II Pro – XC2VP70-6FF1517:

Resources	Used	Available	Utilization
Function Generators	169	66176	0.3%
CLB Slices	85	33088	0.3%
Dffs or Latches	120	69068	0.2%

6.16.7 Version History

Date	Version	Comments
April 05	Beta 1	Initial Version
March 06	Beta 7	Description updated.

6.17 Dlink Interface

6.17.1 Functional Description

The core **dlink_interface** is a design example that performs data serialization and de-serialization on up to 7 differential lines connected to the µDB connector (I/O EXT) located on the front panel of AC2x0 modules.

The 15 pin µDB connector offers either 14 closely coupled individual lines or 7 differential pairs that can be configured to any standard supported on banks supplied with a 2.5V Voltage reference. As there is no active logic between the pin connector and the FPGA buffer, the user can independently instantiate any type of buffer (input or output) on each available line.

This core is primary aimed at testing of the µDB connector. It can easily be removed from the user design or replaced by a more straightforward user control of the IO buffers.

6.17.2 Port Description

Port Name	Size	Type	Description
IB_Customer **IB_Dirsel** **IB_Write** **IB_Rdy** **IB_TimeO** **IB_Addr** **IB_DataW** **IB_DataR**	1 1 1 1 1 32 32 32	In In In Out In In In Out	Must be connected to the IB-BUS signal with the same name. For details please refer to the description of the IB-BUS.
IB_Clk	1	In	Internal Bus Clock
Reset	1	In	Start Up Reset
IO_Sel	1	In	Serial Line Select Input Signal
IO_Data	1	In	Serial Data Input Signal
IO_Clk	1	In	Serial Clock Input Signal
S_Clk	1	In	Serial Clock Reference used for Serial transmission
S_CS	1	Out	Serial Line Select Output Signal
S_DATA	1	Out	Serial Data Output Signal
S_CK	1	Out	Serial Clock Output Signal
S_CLR	1	Out	Serial Clear Output Signal
IO_DDO	7	Out	Parallel Data Bus Out
IO_DDI	7	In	Parallel Data Bus In
DZ_CRT	7	Out	Impedance Control for LVPECL Differential Inputs [1]

6.17.3 Detailed Description

The core **dlink_interface** can be used in several ways depending on the configuration of the IO Buffers. The FDK Base Design uses this core as serializer / de-serializer on 3 differential output pairs and 3 differential input pairs. These I/O differential pairs are externally connected with each other in a loop configuration within the tester component in the VHDL Test Bench.

The data to transfer is loaded into the Dlink_Dout register. When enabled, by writing the **DENB** bit of the Dlink_Control register, the **dlink_interface** may shift out all or parts of the Dlink_Dout register contents using a dedicated Serial Clock (**S_Clk**). The **BYE** field of the Dlink_Control register defines the number of bytes of Dlink_Dout that are sent to the serial interface. The Dlink interface also provides a

[1] Note: Impedance Control bits should be asserted for each LVPECL differential pair used as Input. Otherwise, it must be left at its default value '0'.

receiver that is able to de-serialize the data into the Dlink_DIN register and flags the data reception into the **DAVL** bit of the Dlink_Control register.

The **dlink_interface** can also be used as a simple input or output (or any mix thereof) buffer by using the **IO_DDO** and **IO_DDI** ports. These ports are directly connected to the **DDO** and **DDI** fields of the Dlink_Control register and can be used to interface with dedicated IO buffers.

The table below presents the pinout allocation for the µDB connector.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
DP6_p	DP6_n	DP5_p	DP5_n	DP4_p	DP4_n	DP0_p	DP0_n	DP3_p	DP3_n	DP2_p	DP2_n	DP1_p	DP1_n	GND

Table 5 : µDB Connector Pinout

DPx_n refers to the DIO_DPx_n line defined in the Base Test Design.

6.17.4 Registers

The Dlink_Control register controls the data transmission / reception to/from the Dlink. The Dlink_Dout register defines the data to be sent to the Dlink and the Dlink_Din contains the data received from the Dlink. They are mapped to a fixed location (0x2290, 0x2294, and 0x2298) within the Customer Reserved FPGA Register space and are available whenever the Dlink Interface Core is used.

6.17.4.1 Dlink_Control

Register Space	Register Number	Register Address
Customer	**36**	**0x2290**

31	30..24	23..17	16	15	14..12	11	10	9..8	7	6..0
--	DDO	DDI	DAVL	DENB	--	SEND	--	BYE	--	DPDIR

[6..0]	**DPDIR**	**RW**	Input Impedance control for each differential pair
[9..8]	**BYE**	**RW**	Byte Enable: Defines the number of bytes in the Dlink_Dout register that should be shifted out with the SEND Command '0' = 1 byte, '1' = 2 bytes, '2' = 3 bytes, '3' = 4 bytes
[11]	**SEND**	**RW**	Start sending the Dlink_Dout contents to the DLink
[15]	**DENB**	**RW**	Data Enable: '0' = interface disabled, '1' = interface enabled
[16]	**DAVL**	**R**	Data Available: Asserted upon data receipt, reset upon register read
[23..17]	**DDI**	**R**	Digital Data In. Meaningful only if the micro DB link is used with 7 differential inputs
[30..24]	**DDO**	**RW**	Digital Data Out. Meaningful only if the Dlink is used with 7 differential outputs

6.17.4.2 Dlink_Dout Register

Register Space	Register Number	Register Address
Customer	**37**	**0x2294**

31..0
DOUT

[31..0]	**DOUT**	**RW**	Defines the data word to be sent to the Dlink

6.17.4.3 DLink_Din Register

Register Space	Register Number	Register Address
Customer	**38**	**0x2298**

31..0
DIN

[31..0] **DIN** **RW** Last data word received from the Dlink

6.17.5 Instantiation

This core is already instantiated in the base design example.

The Dlink Interface core may be used in several ways depending on the direction and type of the I/O buffers. The Base Design Instantiation enables the "serializer / de-serializer " configuration.

In the Base Design Configuration, **IO_DDO** should be looped into **IO_DDI** to successfully run the test bench. **Sysclk2** should be connected to **S_Clk**. **DDO(6)** is connected to **DIO_DP6** only for test purposes.

Other configurations are possible with the same core.

Please note that even if the Dlink Interface core is not used for controlling the lines to the µDB connector, **DIO_CFG(6:0)** must be driven to '0' to avoid any transmission problems.

6.17.6 Constraints

- Clock Constraint

This core assumes an **IB_Clk** frequency of 33 MHz and an **S_Clk** frequency of 133 MHz. This constraint must be defined for the clock manager component and is automatically propagated throughout the whole design.

In order to save precious BUFG resources the **IO_Clk** should be mapped to long line resources (low skew lines). This is done at the Base design level using the following declarations

```
attribute uselowskewlines: string;
attribute uselowskewlines of IOL_DP2: signal is "yes";
```

- Input/Output Constraints

DIO_DPx_n /_p ports should be connected to external I/O buffers

The IOSTANDARD attributes for these buffers for **DIO_DPx_n /_p** could be either differential (LVDS/LVPECL) or any single ended **2.5V** signaling standard (e.g. LVCMOS25). Buffer type (input or output) is left to the requirements of the user application.

Please refer to the ac240.ucf file for LOC constraints. The IOSTANDARD attributes for these buffers should be "LVCMOS33" with default drive strength (12mA) and slew rate (Slow) values.

6.17.7 Resource Utilization

Resource count and relative usage in the target Xilinx Virtex II Pro – XC2VP70-6FF1517:

Resources	Used	Available	Utilization
Function Generators	345	66176	0.5%
CLB Slices	222	33088	0.7%
Dffs or Latches	443	69068	0.6%

6.17.8 Version History

Date	Version	Comments
April 05	Beta 1	Initial Version
March 06	Beta 7	Description updated.

6.18 Dual Port Memory Interface

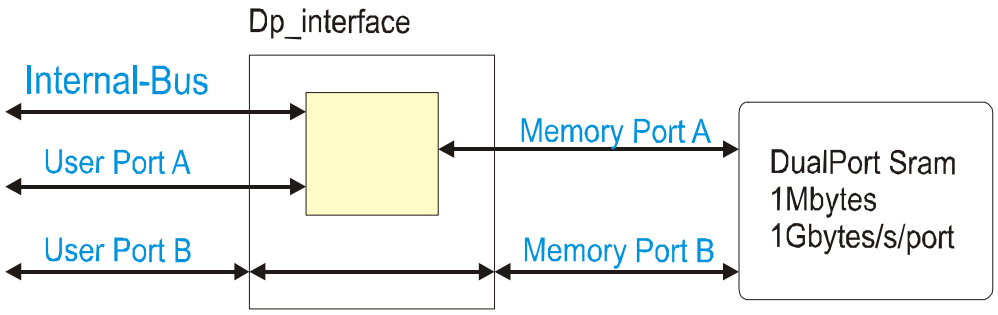
The core **dp_interface** is the interface to the (optional) external dual port SRAM. The size of the dual port memory is 8 Megabits, equivalent to 1 MegaSamples. This is somewhat more than the total of 5 Megabits available within the FPGA block RAMs, and much less compared to the (optional) external dynamic RAM. Its advantage lies in its usage, much simpler as compared to the dynamic RAM.

The Dual Port is functional only on boards of revision C and later.

6.18.1 Functional Description

The dual port SRAM has two 64-bit user ports operating at up to 133 MHz. This is enough to store the data stream at 2 GS/s, using both ports. However, it cannot be read simultaneously at the same speed.

Its versatility makes it easy to implement applications with random addressing, such as histograms.



The dual port RAM can be read or written by the user program through the Internal Bus port. The Internal Bus port and the User port are multiplexed and cannot be used simultaneously. The Internal Bus port runs at 33 MHz. The priority of the Internal Bus over the User port is controlled by the bit **DPM_Local** in the DP_Control register.

Each port handles both single read/write or burst read/write operations.

The memory can be reset with the **DPM_reset** bit of the DP_Control register or with the core input signal **DPM_Reset**. A reset should be executed by the program at least once after the dedicated DCM is enabled.

6.18.1.1 User Port

Each port has independent clock, data, address, read / write, and byte enable control signals. The user may connect the clock to any desired frequency up to 133 MHz. The data bus is 64 bits wide.

On a single port, back-to-back read/write can only be done at half the clock rate. A read can follow a write, but a write can follow a read only after a delay of 2 clock cycles.

If necessary, full speed back-to-back read/write should be implemented with both ports, one for reading and the other for writing. This achieves a throughput of max 2.13 GB/s with a clock frequency of 133 MHz on each port (133 M x 8 x 2 = 2.13 G).

6.18.1.2 Internal Bus Port

Logically, the dual port RAM is accessed as a 32-bit memory of 256 Kwords, using the Indirect Access mode of the Local Bus interface.

The bit 1 of the DP_Control register selects which port, User Port A or the Internal Bus port, takes control of the memory port A. The selection must be done prior to accessing the memory.

The start address is set by writing into the Indirect Address register. Correct values are (0 + N * 4), where N [0…256K] is the requested 32-bit word. Even values of N access the lower 32 bits of the memory while odd values of N access the upper 32 bits of the memory.

6.18.1.3 Self-Testing

Three patterns are implemented:

- Memory positions are set to values that are incremented. The 32-bit data are split in two words, a 17-bit counter and a 15-bit counter. This makes the pattern unique over the entire memory.
- All memory positions are set to the value of the test pattern register.
- All memory positions are set to the inverted value of the test pattern register.

The test can be run in these different modes:

1. No stop on error. This mode tests the entire memory. The **T_Status** bit in the DPStatus register is set '1' if at least one error occurs. The **T_End** bit is set at the end of the test.
2. Stop on error. This mode tests the entire memory. The test is suspended when an error occurs. The test position, the test data, and the test result can be read by software.. The test is continuously repeated at the position where the error occurs (write-read-test). The **T_error** bit in the DP_Status register indicates if the current test is successful or not. Test repetition occurs until the program activates (for at least 2 us) and deactivates the **T_Continue** bit. This is useful for debugging. The test can be aborted by deactivating **T_Start**.
3. Automatic Error. This test simulates the behavior of the test when an error occurs. An error will be automatically generated at memory address 0x8.
4. Short memory. This is to shorten the simulation time. In short memory mode, only the first 16 positions are tested.

6.18.2 Instantiation

The core is already instantiated in the base design example. It is always associated with its companion, **dp_interface_io**, which handles the Xilinx IO primitives.

6.18.3 Port Description

Signal	Size	Type	Short Description
INTERNAL BUS			
IB_Customer **IB_Dirsel** **IB_Write** **IB_Valid** **IB_Rdy** **IB_TimeO** **IB_End** **IB_IndirCtr** **IB_Addr** **IB_DataW** **IB_DataR**	1 1 1 1 1 1 1 1 31 32 32	In In In In Out In Out In In In Out	Must be connected to the IB-BUS signal with the same name. For details please refer to the description of the IB-BUS.
IB_Clk	1	In	Internal Bus clock, must be connected to **lbclkg**, the Local Bus clock
Reset	1	In	Reset, must be connected to the general reset **Dreset**
USER Port: x=A or B, stands for User Port A or User Port B			
UPx_select	2x1	In	Port selection for read and write. It must remain '1' for one clock cycle for single access or multiple times for burst access. On each port, a write can be directly followed by a read or a write access. A read can be directly followed only by a read. A minimum of two 2-clock cycles must be inserted after a read before a write can occur.
UPx_ADS	2x1	In	Address strobe. When '1', the value on **UPx_Address** is stored in the auto increment address register of the memory, otherwise the current value of the auto increment register of the memory is used.
UPx_Write	2x1	In	It must be set to '1' for writing, '0' for reading. Simultaneous

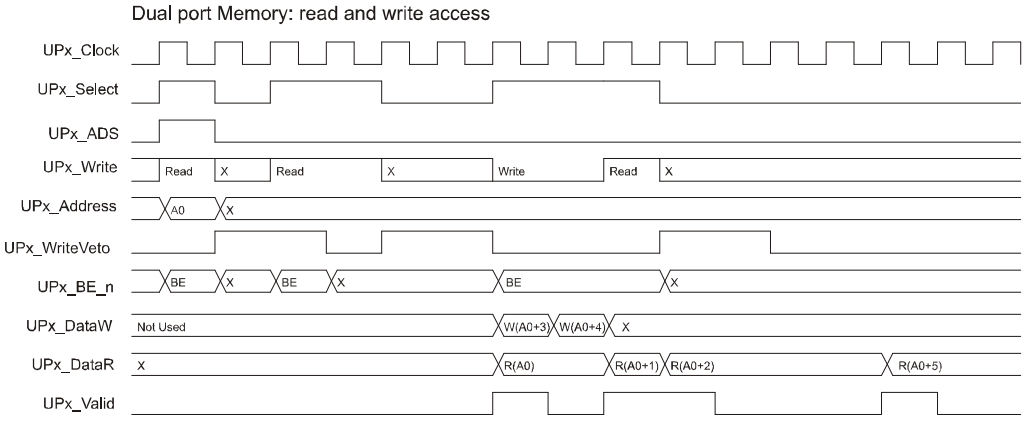
Signal	Size	Type	Short Description
			writing to the same location on memory ports A and B will store unknown data.
UPx_WriteVeto	1	In	When '1', the dual memory port does not accept any write access
UPx_Address	2x20	In	Address. The value is always in bytes, independently of the selected data bus width.
UPx_DataW	2x64	In	Bus for data write
UPx_DataR	2x64	Out	Bus for data read
UPx_Valid	2x1	In	After a read cycle, will be '1' for one cycle when the data is ready. This occurs 6 clock cycles after **UPx_select** has been set.
UPx_clock	2x1	In	Memory port clock. It can be set to any frequency up to 133 MHz.
UPA_Busy	1	Out	User Port A only: '1' when the user port cannot be accessed. This is the case when the program reads or writes the dual port memory.
DPM_Reset	1	In	'1' resets the Dual Port Memory

OUTPUT to Dual Port Memory: x=A or B, stands for Memory Port A or Port B

DPx_Select	2x1	Out	Portx Select.
DPx_ADS_n	2x1	Out	Portx Address Strobe.
DPx_Write_n	2x1	Out	Portx Direction
DPx_Address	2x17	Out	Portx Address
DPx_CNTEN_n	2x1	Out	Portx Address Counter Increment
DPx_BE_n	2x7	Out	Portx Byte Enable. "y" is any value from 0 to 7.
DPx_DataW_hz	2x1	Out	Portx output enable for FPGA Data Bus output
DPx_DataW	2x64	In	Portx Data Bus Output
DPx_DataR	2x64	Out	Portx Data Bus Input
DPM_Reset_n	1	Out	Common reset for port A and port B

6.18.4 User Port Timing Diagram

The access to the memory is enabled when **`Upx_Select`** is active. The address must be defined and stored in the memory by simultaneously activating **`UPx_ADS`** on the first read/write transaction.



Afterwards, the user may continue using the address strobe and supplying an address (which may be random or incremented). Alternatively, the user may stop activating the address strobe, in which case the memory circuit automatically increments the address. It must be remembered that for the User port A only (because the memory port A is shared for the User port A and the Internal Bus port), the address in the memory chip could be modified when the application software accesses the memory (via the Internal Bus port).

6.18.5 Registers

6.18.5.1 DP_Control

Register Space	Register Number	Register Address
Customer	**39**	**0x229C**

31..9	8
	T_short

7	6	5	4	3..2	1	0
T_ForceError	T_StopOnError	T_Continue	T_Start		DPM_Local	DPM_Reset

[0] **`DPM_Reset`** **RW** Reset of dual port memory. It must be set to '1' and back to '0' at least once after the FPGA firmware is reloaded.

[1] **`DPM_Local`** **RW** Select Master of memory port A

- **`0`** Access to the memory port A is controlled by the User Port A port
- **`1`** Access to the memory port A is controlled by the Internal Bus port.

[4] **`T_Start`** **RW** '1' Starts the test sequence. '0' aborts the test sequence

[5] **`T_Continue`** **RW** Continue the test after an error was found in the mode **`StopOnError`**. **`Test_Continue`** must be toggled to '1' and back to '0'. Duration of the '1' state must be greater than 2 µs.

[6] **`T_StopOnError`** **RW** '1' to set the debug **`StopOnError`** mode, '0' to complete the whole test without stopping

[7] **`T_ForceError`** **RW** An error will automatically be generated at address 0x8.

[8] **`T_short`** **RW** All test patterns are executed but only for memory positions 0 to 15. This is useful for simulation only.

6.18.5.2 DP_TestPatternControl

Register Space	Register Number	Register Address
Customer	**40**	**0x22A0**

31..17	16..0
	Test_Pattern

[16..0] **Test_Pattern** **RW** Test pattern

6.18.5.3 DP_Status

Register Space	Register Number	Register Address
Customer	**41**	**0x22A4**

31..24	23	22	21	20	19..17	16..0
	T_Chip	T_Error	T_Status	T_End		T_Address

[16..0] **T_Address** **R** Current test address

[20] **T_End** **R** End of test flag: '1' indicates the test has completed. The flag is reset by the rising edge of the bit **T_Start** in the DP_Control register.

[21] **T_Status** **R** Test status flag: The flag is reset by the rising edge of the bit **T_Start** of the DP_Control register.

- **0** The entire test completed without error
- **1** There was at least one error

[22] **T_Error** **R** Test error flag, is the status of the current test at the current memory address. The flag is reset by the rising edge of the bit **T_Start** of the DP_Control register.

- **0** The current test failed
- **1** The current test passed

[23] **T_Chip** **R** When an error occurred, indicates which memory chip failed. '0' for the instance NDP1A, '1' for the instance NDP2A.

6.18.5.4 DP_TestValue

Register Space	Register Number	Register Address
Customer	**42**	**0x22A8**

31..0
TestValue

[31..0] **TestValue** **RW** The value written at the current/last test address

6.18.5.5 DP_TestResult

Register Space	Register Number	Register Address
Customer	**43**	**0x22AC**

31..0
TestResult

[31..0] **TestResult** **RW** The value read at the current/last test address

6.18.6 Accessing the Dual Port Memory

The Dual Port memory can be read or written in Indirect Access mode using the Indirect Addressing register. The maximum length is 2 Mwords of 32 bits. The Indirect Address Register and the Buffer Identifier Register must be set prior to reading or writing the memory.

6.18.6.1 DPMemory

Register Space	Register Number	Register Address	Buffer Identifier
Customer	**0**	**0x2200**	**0x04**

31..0
RWData

[31..0] **RWData** **RW** Data format depends on the customer application.

6.18.7 Constraints

- Clock Constraint

This core assumes an **IB_Clk** frequency of 33 MHz. This constraint must be defined for the clock manager component and is automatically propagated throughout the whole design.

- User Port Clock Constraints

This core assumes **UPA_clock** and **UPB_clock** frequencies of maximum 133 MHz. This constraint must be defined for the clock manager component and is automatically propagated throughout the whole design.

- Output and Input Signal Constraints

All FPGA inputs / outputs are LVTTL. All must be registered within the IOB. All must have the drive strength set to FAST – 12 mA.

6.18.8 Resource Utilization

Resource count and relative usage in the target Xilinx Virtex II Pro – XC2VP70-6FF1517:

Resources	Used	Available	Utilization
Block RAMs	2	328	0.6%
Function Generators	843	66176	1.3%
CLB Slices	729	33088	2.2%
Dffs or Latches	1457	69068	2.1%

6.18.9 Version History

Date	FDK Version	Comments
February 06	Beta 6	Initial Version
March 06	Beta 7	Description updated.
November 06	1.0	Bit **DPM_Local** of the register **DP_Control** was described the opposite ways it effectively behave. Corrected.

6.19 Dual Port Memory Control Example

The core **dp_ctr_example** is delivered as an example of control for the core **dp_interface**. Its function is to store a portion of the incoming data streams to the dual port memory after a trigger occurred, until the memory is full. Developers should remove it or may adapt it to their own application.

The two incoming data streams, 2x16 bytes at each **sysclk** period are multiplexed and sent to the dual port interface as two new streams of 2x8 bytes with **sysclk2**, which is twice the **sysclk** frequency.

The first stream is sent to the port A of the dual port memory interface. It is written starting at address 0. The second stream is sent to the port B of the dual port memory interface. It is written starting at the middle of the memory.

The control bit **Start** enables storage to the dual port memory. Depending on the state of the control bit **StartOnTrigger**, storage will effectively begin either immediately or after the first trigger occurs. Of course the stream must be enabled prior to activating **Start**. The storage stops if **Start** is set back to '0' or when the entire memory has been filled.

In mode start on trigger, the output signal **TriggerEnable** will be set '1' until the input signal **SP_Trigger** becomes '1' indicating a trigger has been detected.

The status bit **FULL** is set to '1' after the dual port memory has been entirely filled. It is set back to '0' when **START** is again activated.

6.19.1 Port Description

Signal	Size	Type	Short Description
IB_Customer **IB_Dirsel** **IB_Write** **IB_Rdy** **IB_TimeO** **IB_Addr** **IB_DataW** **IB_DataR**	1 1 1 1 1 31 32 32	In In In Out In In In Out	Must be connected to the IB-BUS signal with the same name. For details please refer to the description of the IB-BUS.
IB_Clk	1	In	Internal Bus clock, must be connected to **lbclkg**
Reset	1	In	Reset, must be connected to the general reset **Dreset**
Data Stream Input and Trigger			
SP_Data_A	128	In	Samples from channel A
SP_Data_Val_A	1	In	Data valid from channel A
SP_Data_B	128	In	Samples from channel B (AC/SC240 only)
SP_Data_Val_B	1	In	Data valid from channel B (AC/SC240 only)
SP_Trigger	1	In	Trigger marker
Enable_Trigger	1	Out	Trigger Enable to the Trigger core
OUTPUT to Dual Port Memory: x=A or B, stands for Memory Port A or Port B			
UPx_ADS	2x1	Out	Address strobe. When '1', the value on **UPx_Address** is stored in the auto increment address register of the memory, otherwise the current value of the auto increment register of the memory is used.
UPx_Write	2x1	Out	It must be set to '1' for writing, '0' for reading. Simultaneous writing to the same location on memory ports A and B will store unknown data.
UPx_WriteVeto	1	In	When '1', the dual memory port does not accept any write access
UPx_Address	2x20	Out	Address. The value is always in bytes, independently of the selected data bus width.
UPx_DataW	2x64	Out	Bus for data write
UPx_DataR	2x64	In	Bus for data read
UPx_Ready	2x1	Out	After a read cycle, will be '1' for one cycle when the data is ready. This occurs 6 clock cycles after **UPx_select** has been set.

Signal	Size	Type	Short Description
UPx_clock	2x1	Out	Memory port clock. It can be set to any frequency up to 133 MHz.
UPA_Busy	1	In	User Port A only: '1' when the user port cannot be accessed. This is the case when the program reads or writes the dual port memory.

6.19.2 Registers

6.19.2.1 Control Register

Register Space	Register Number	Register Address
Customer	**66**	**0x2308**

31..2	1	0
--	Full	Start

WRITE

[0] **Start** **W** Write only bit. Set Start to '1' will enable data storage to the dual port memory. Set to '0' will stop data storage.

[1] **StartOnTrigger** **RW** '0': Storage will start immediately after **Start** has been set '1'.

'1': Storage will start after the first trigger occurrence after **Start** has been set '1'.

READ

[0] **Full** **R** Read only bit. '1' indicate the dual port memory has been entirely filled.

6.19.3 Resource Utilization

Resource count and relative usage in the target Xilinx Virtex II Pro – XC2VP70-6FF1517:

Resources	Used	Available	Utilization
Function Generators	185	66176	0.3%
CLB Slices	212	33088	0.6%
Dffs or Latches	424	69068	0.6%

6.19.4 Version History

Date	FDK Version	Comments
February 06	Beta 6	Initial version
March 06	Beta 7	Description updated.
November 06	1.0	Updated port description. Updated operation description Triggered mode is now programmable. BufferFull and stop filling were badly implemented. Corrected.

6.20 Serial Front Panel Data Port Controller

6.20.1 Functional Description

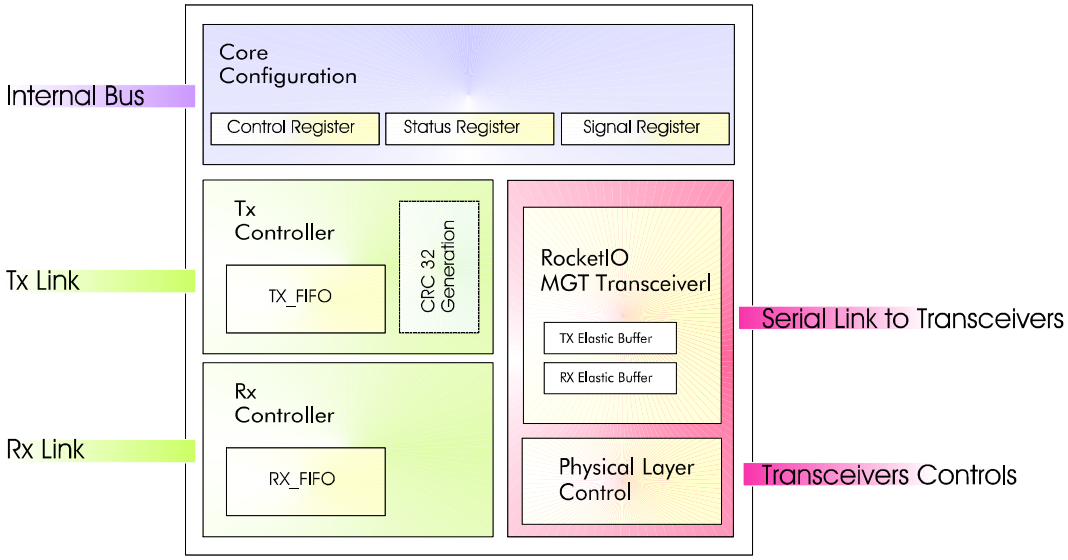
The Serial Front Panel Data Port (sFPDP) core, **`slc_controller`**, implements a *Data Link* layer compliant with the Serial Front Panel Data Specification (ANSI/VITA 17.1-2003). It is intended to be used together with one RocketIO Multi Gigabit Transceiver (Virtex II Pro primitive) that implements the *physical* layer of the *Data Link*. The physical media could be either an optical or a copper link at up to 2.5Gbit/s.

Note: It is assumed that the user of the sFPDP core has a basic knowledge of networking technology and is familiar with the Serial Front Panel Data Port protocol and with the RocketIO primitives. Detailed information can be found in

- [RD1] ANSI/VITA 17.1-2003 Serial Front Panel Data Port
- [RD2] RocketIO Transceivers User Guide- Ug024-V2.5, Xilinx 9.12.2004
- [RD3] LocalLink Interface Specification, DS230, Xilinx 18.10.2002

Convention: The acronym TX points out to anything related to the Transmit path, whereas the RX one to the Receive path.

The figure hereafter presents a functional block diagram of the sFPDP Controller.



Configuration and control of the core are performed through dedicated registers that are accessed with the Agilent Internal Bus. Data transfers are performed using the Local Link standard used by Xilinx for packet transmission (See [RD3]).

The sFPDP core performs all the tasks related to the framing of user data into the Fiber Frame (TX), including the CRC32 generation and insertion, from its internal 16 KB TX FIFO. It also performs all the tasks related to retrieving the user data from its internal 2 KB RX FIFO. The sFPDP core instantiates one RocketIO MGT primitive and handles all the physical layer control tasks, including the transceiver control and initialization. The physical link is directly managed by the RocketIO transceiver, which implements all the functions involved in the 8B/10B coding and decoding including clock recovery.

6.20.2 Port Description

Port Name	Size	Type	Description
`SLC_Base_Add`	12	In	Twelve rightmost bits of the Controller Base Address
`IB_Customer` **`IB_Dirsel`** **`IB_Write`** **`IB_Rdy`** **`IB_TimeO`**	1 1 1 1 1	In In In Out In	Must be connected to the IB-BUS signal with the same name. For details please refer to the description of the IB-BUS.

Port Name	Size	Type	Description
IB_Addr **IB_DataW** **IB_DataR**	32 32 32	In In Out	
IB_Clk	1	In	Internal Bus Clock – 33 MHz
Reset	1	In	Start Up Reset
Tx_Data_Usr	128	In	TX Data Bus. It contains the data of the frame to be transmitted.
Tx_Rem_Usr	4	In	TX Data Remainder: Indicates the number of valid bytes on given transfers.
Tx_Sof_n_Usr	1	In	TX Start of Frame: Indicates the first transfer for a given frame.
Tx_Eof_n_Usr	1	In	TX End of Frame: Indicates the last transfer for a given frame.
Tx_Src_rdy_n_ Usr	1	In	TX Source ready : Indicates that the source is ready to transfer data
Tx_Dst_rdy_n_ Usr	1	Out	TX Destination ready : Indicates that the core is ready to accept data
Rx_Data	40	Out	RX Data Bus. It contains the data and the error bus of the received frame.
Rx_Rem	4	Out	RX Data Remainder: Indicates the number of valid bytes on given transfers.
Rx_Sof_n	1	Out	RX Start of Frame: Indicates the first transfer for a given frame.
Rx_Eof_n	1	Out	RX End of Frame: Indicates the last transfer for a given frame.
Rx_Src_rdy_n	1	Out	RX Source ready : Indicates that the core is ready to transfer data
Rx_Dst_rdy_n	1	In	RX Destination ready : Indicates that the user is ready to accept data
Rx_Pio1	1	Out	PIO1 – Received sFPDP Signal Status
Rx_Pio2	1	Out	PIO2 – Received sFPDP Signal Status
Rx_Dir	1	Out	DIR – Received sFPDP Signal Status
Tx_Nrdy	1	Out	NRDY – Received sFPDP Signal Status
Rx_Loop	1	In	Loop command for dynamic loop configuration
Tx_Empty	1	Out	TX FIFO Status: A ‘1’ indicates that the TX FIFO is empty.
Link_Err	1	Out	Data Link Error: it indicates that the data link has encountered an error condition
Link_Rdy	1	Out	Data Link Ready: it indicates that the data link is initialized and ready for the transmission or reception of data.
Sysclk	1	In	User Clock used to store and retrieve User Frame
Refclk	1	In	Reference clock used by the MGT for serial transmission
Usrclk	1	In	User Clock used for reading/writing the data buffer
Usrclk2	1	In	Clock used to transfer data and status within the FPGA fabric
Usrclk_lock	1	In	Lock Status of the DCM that drives Usrclk/Usrclk2
TXN	1	Out	Transmit differential port (FPGA external)
TXP	1	Out	Transmit differential port (FPGA external)
RXN	1	Out	Receive differential port (FPGA external)
RXP	1	Out	Receive differential port (FPGA external)
Phy_Rx_Loss	1	In	Physical Layer Loss of RX signal
Phy_Tx_Fault	1	In	Physical Layer TX Fault
Phy_Tx_Enable	1	Out	Physical Layer Enable

6.20.3 Detailed Description

Basically, the core architecture is divided into two main blocks – the TX Controller Block which implements all the operations involved in the data transmission and the RX Block which implements all the operations involved in the data reception. Each controller shares the same Control, Configuration and Status registers. Both controllers may interact with each other, especially when the core is configured to run in copy or copy loop mode.

The core's behavior is highly dependent on its configuration which is mainly static. The core must first be configured into one of the operating modes using the SLC Control register. The table hereafter shows the different Serial Front Panel Data Port running modes that are currently supported and the associated software configuration.

Serial FDPD Mode	Supported	TXE	RXE	CPY	FWC	MST	RFC	CRC
Tx Only Mode	✓	1	0	X	0	X	0	X
Rx Only Mode	✓	0	1	0	0	0	0	X[2]
Bidirectional Mode	✓	1	1	X	X	X	X	X
Copy Mode	✓	1	X[3]	1	0	X	0	X
Copy Loop Mode	✓[4]	1	X	1	1	X	1	X
Copy Master option	✓	1	X	1	X	1	X	X
Flow Control option	✓[5]	1	1	X	1	X	1	X
CRC option	✓	X	X	X	X	X	X	1

Note: X means "don't care".

Note: Do not attempt to use the core until the **`Usrclock`**/ **`Usrclock2`** are configured and ready (DCM locked). Otherwise access to these registers will be neither relevant in reading nor effective in writing.

Note: Copy Loop Mode may be dynamically controlled by the use of the RX_loop port. This feature can be removed by wiring this input port to '1'.

6.20.3.1 TX Controller

The TX Controller is the sequencer used to frame the user data into Serial FPDP *Normal Data Fiber* Frames and to send them to the associated RocketIO that performs all the tasks relative to the *Physical* layer. It handles the overall framing process (type of the frame, frame length, control characters to be inserted, flow control and CRC encoding if requested).

The TX Controller is enabled by setting the **`TXE`** bit to 1 in the SLC Control register. This action initiates the physical layer initialization process. It enables the optical transceiver by driving the **`Phy_tx_enable`** control line to 1 and waits for the **`phy_tx_fault`** status signal to be negated. The logical state of that control line could be read by the **`TXF`** bit of the SLC Status register.

Then, it sends IDLE characters (TX_INIT_DLY) for approximately 800 ms to enable the receiver's physical layer to lock onto the serial data stream. Upon completion of this physical layer initialization, the **`TXR`** status bit of the SLC Status register is asserted. Then, it sends **`TX_IFRX_VAL`** Idle Data Fiber Frames before reaching the "initialized" state (Link_Ready = 1). Upon completion of this link layer initialization, the **`TXR`** status bit of the SLC Status register is asserted.

Both **`TX_IFRX_VAL`** and **`TX_INIT_DLY`** are generic parameters that are different between the simulation model and the synthesizable model.

After the TX Controller is started, it maintains link synchronization by continuously sending Idle characters and *Idle Data Fiber* Frames until the TX_FIFO memory contains enough data to start the framing of a user Data Fiber Frame. As long as the TX controller has no user data to frame, it maintains the link synchronization and the Serial FPDP signals transmission.

[2] CRC checking not supported in this core version. If CRC is used by TX, it must be set to 1
[3] If RXE is not set the data are not available at the RX side of the link
[4] Dynamic Copy Loop Mode available using Rx_Loop mode
[5] RX Flow Control and TX Flow Control are independently configurable. RX Flow control is limited to the RX buffer size and by the user core ability to retrieve enough data.

The Serial FPDP protocol defines 6 status and control signals that may be exchanged between two nodes of a Serial FPDP link (DIR, PIO1, PIO2, NRDY, TX FIFO Overflow, and STOP/GO). These status and control signals are embedded within the control Ordered Sets that frames any serial FPDP Fiber frames (See [RD1] for more details). The STOP/GO signal is internally used by the core for the flow control management and is not available at the User interface. The core guarantees by design that no TX FIFO Overflow event will ever occur. The remaining control and status signals are managed by the SLC Signal register. Local signals (i.e. those defined at the Serial FPDP core level) are driven by the corresponding bit values (**`TX_PIO1`**, **`TX_PIO2`**, **`TX_DIR`**, **`RX_NRDY`**) whereas remotely driven signals are available as read only bits (**`RX_PIO1`**, **`RX_PIO2`**, **`RX_DIR`**, **`TX_NRDY`**, **`RX_FOVF`**).

Note: Remotely driven signals are also available as dedicated ports in this version of the sFPDP controller.

The update rate of these control signals depends on the current fiber frames and on the selected running mode. While the TX Controller has no user data to frame, it periodically inserts *Normal Idle Fiber Frames* that still update the values of these control signals (one *Normal Idle Fiber* Frame after TX_IDLE_MAX (16) Idle characters). In this case any signal changes are propagated within the next 320 ns. When the TX Controller sends user data, the update rate depends on the size of the *Normal Data Fiber Frame* and on the idle time between two consecutives *Normal Data Fiber Frames*. In this case any signal changes are propagated within the next 8320 ns.

The user provides the data frame to the core by means of the TX link which is implemented as a subset of the Local Link Specification (See [RD3] and section 6.20.3.3 ***TX AND RX LOCAL LINKS***). The TX link data are directly stored in the TX FIFO using the **`Sysclk`** clock. The size of the TX buffer is set to 16K by the TX_BUFFER_SIZE generic parameter. It must be a multiple of 8K due to the memory buffer layout which is organized with a programmable depth of 4 Block RAM primitives.

The TX Controller can generate two types of Serial Front Panel Data Port frames:

1. *Normal Data Fiber Frame*
2. *Sync Without Data Fiber Frame*

The *Sync With Data Fiber Frame* cannot be generated but is supported and decoded as a valid frame.

Normal Idle FiberFrames are *Normal Data FiberFrames* with no data.

As soon as data are available in the TX FIFO, the TX controller will try to send them as fast as possible using *Normal Data Fiber Frame* with the maximum length. The frame length is 512 data words of 32 bits unless there is not enough data to complete the frame. If the size of the user frame is greater than one *Normal Data Fiber Frame* it is split into multiple *Normal Data Fiber Frames*. The last one is shorter or equal to the maximum length of a *Normal Data Fiber Frame*.

Note: In order to maximize the data bandwidth it is highly recommended to wait for the TX buffer to be empty (Tx_Empty = '1') and then fill it at the maximum TX link bandwidth. This avoids generating short incomplete *Normal Data Fiber Frame* that reduce the link performance by adding the protocol overhead.

If the **`FWC`** bit of the SLC Control register is asserted, the TX Controller takes into account the flow control information issued from the RX controller to pause the data emission. Once the STOP Ordered Set is decoded on the RX Controller side, the TX Controller interrupts its ongoing transmission by an early completion of its current Normal Data Fiber Frame. Then it will restart transmission upon the receipt of a GO Ordered Set.

If the **SYF** bit of the SLC Control Register is set, the TX Controller will mark the end of each TX Frame by sending an additional *Sync Without Data Fiber Frame* after the last *Normal Data Fiber Frame*. It might be a convenient way to mark the user frame boundaries at the receiver side.

The user could also request the sending of one *Sync Without Data Fiber Frame* by writing the **FSY** bit of the SLC register at '1'. This bit must be set back to '0' in order to enable the next *Sync Without Data Fiber Frame* request.

6.20.3.2 RX Controller

The RX Controller is the sequencer used to optionally decode the arriving Data Fiber frames. It handles the overall decoding process and discards all the protocol overhead characters. It stores the useful data and the associated status bits in the RX_FIFO memory.

The RX Controller is enabled by writing 1 into the **`RXE`** bit of the SLC Control register. This action initiates the physical layer initialization process. First it waits for the **`Phy_rx_loss`** control line to be negated indicating that the optical transceiver has detected a useful signal. The logical state of that control line can be read via the **RXL** bit of the SLC Status register.

Then, it must successfully decode RX_INIT_DLY (0x02000000) IDLE characters to complete the physical layer which is flagged by the assertion of the **RXR** status bit of the SLC Status register. Finally it must correctly decode RX_IFRX_VAL *Idle Data Fiber Frames* before reaching the "initialized" state where the **RXK** status bit is asserted**.**

Both RX_IFRX_VAL and RX_INIT_DLY are generic parameters that are different between the simulation model and the synthesizable model.

Note: At this point it is highly recommended to clear all status bits by writing '1' into the **RSF** bit of the SLC Control register.

Once enabled, the RX Controller performs continuous checking of the arriving data. It could detect either a 8B/10B decoding error (**DER**), a Running Disparity Error (**RDE**), a loss of synchronization error (**SYE**) or a format error (**FTE**) that would assert the corresponding bit in the SLC Status register. Any error is memorized until it is explicitly cleared by the writing '1' into the **RSF** bit of the SLC Control register.

The RX controller's behavior is closely related to the sFPDP running mode. The Copy modes do not require that the controller be configured in the Enable state. Data are just decoded and passed through the RX controller to the TX controller. Data decoding is still performed in order to determine Fiber Frame boundaries and allow dynamic switching upon the RX_LOOP port. If this port is connected to '0' and the CPY bit is asserted then the core is running in TX or Bidirectional mode. On the other hand, if this port is connected to '1', the core is running in Copy Mode. Dynamic switching is always performed once IDLE characters are sent or received. In order to compensate for clock drift, the RocketIO instance is configured to add or remove one IDLE character whenever it receives or sends an IDLE character.

The RX controller stores the incoming RX frames in its RX FIFO buffer, in the Bidirectional or RX modes only. The size of the RX buffer is set to 4K by the RX_BUFFER_SIZE generic parameter. It must be a multiple of 2K due to the memory buffer layout which is organized with a programmable depth of one Block RAM primitive.

The RX FIFO should be read using the RX local link clocked by the **Sysclk** user clock. The **Rx_Dst_rdy_n** input port of the RX local link is used to enable the core to output its incoming data. When this port is driven by the user to '0', the core outputs all the data contained in its RX_FIFO onto a 40-bit bus. Received data are contained on the 32 lowest bits. The 8 upper bits should be discarded. They are reserved for future use.

If the **RFC** bit is set to '1' in the SLC Control Register, the RX Controller can request the source to pause its transmission by asking the TX Controller to send a STOP Ordered Set. This request occurs if at least one of the two following conditions is satisfied :

- The user asserts **Rx_Dst_rdy_n** to '1'.
- The RX FIFO filling exceeds the filling threshold defined by the RX FIFO Threshold field **RXTHR** of the SLC Control Register. A value of 0xFF corresponds to a "RX_FIFO full" threshold whereas a value of 0x80 is the "RX_FIFO half full" threshold.

6.20.3.3 TX and RX Local Links

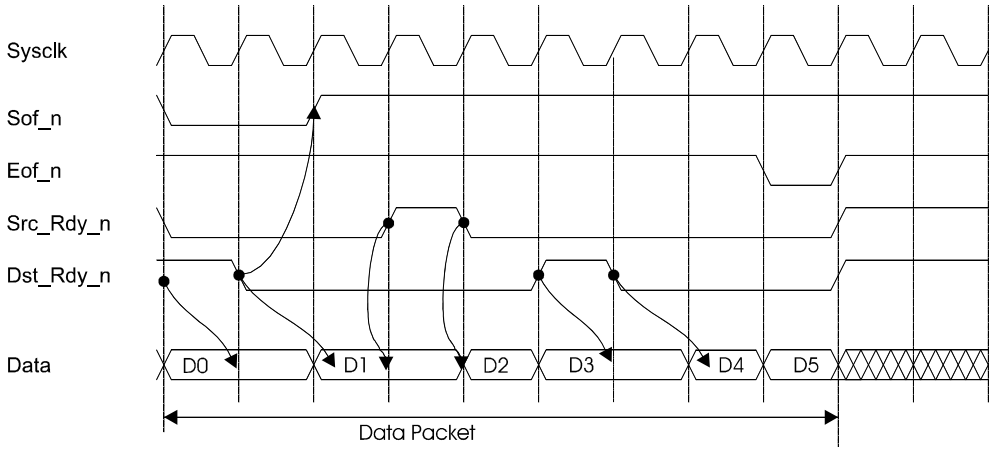
The TX and RX Local links are implemented as subsets of the LocalLink specification. LocalLink is a high performance, synchronous, point-to-point interface, designed to serve as user interface to Xilinx's system interfaces intellectual property (IP) solutions. The interface defines a set of protocol agnostic signals that allow the transfer of protocol data units (PDUs).

LocalLink allows the source and destination interfaces to control data flow with a simple handshake protocol: when the signals **Src_rdy_n** and **Dst_rdy_n** are both valid, data is transferred. Source ready (**Src_rdy_n**) is asserted by the source, when it is ready to transfer data and is presenting data on the data bus.

At the start of a PDU transfer, the source asserts start-of-frame (**Sof_n**) together with source ready (**Src_rdy_n**). If the source temporarily runs out of data during the PDU transfer, it can de-assert source ready.

Destination ready (**Dst_rdy_n**) is asserted when the destination is ready to accept data. This may be before or after it has detected the source interface assert source ready (**Src_rdy_n**). The destination can de-assert **Dst_rdy_n** if it temporarily cannot accept data.

A LocalLink frame transfer with source and destination flow control is shown in the figure hereafter.



Transfer starts when the source interface presents data and asserts **Sof_n** and **Src_Rdy_n**. The destination interface is not ready and holds **Dst_Rdy_n** de-asserted. The source interface presents the next set of data bytes after the designation asserts **Dst_Rdy_n**. Next, the source interface de-asserts **Src_Rdy_n**, which means it is unable to present any new data at the clock cycle. Transfer starts again when the source interface asserts **Src_Rdy_n** and presents the next data set. Transfer ends when the source interface presents data and asserts **Eof_n** and **Src_Rdy_n** and when the destination is ready to accept these data. Further details can be found in [RD3].

The present sFPDP core implementation does not use the remainder field and thus assumes that when data are valid the whole data bus width carries valid data. It implies that the TX frame length granularity is 128 bits or 16 bytes and that the RX frame length granularity is 32 bits (4 bytes).

Channelization, parity, source, and destination discontinuation options are not implemented in order to keep the implementation simple and obtain a user-friendly interface.

6.20.3.4 Clocking

The sFPDP core requires the 5 clock domains **IB_Clk**, **Sysclk**, **Refclk**, **Usrclk,** and **Usrclk2**.

IB_Clk is used for the Internal Bus interface. **Sysclk** is used for data transfer to the TX FIFO and from the RX FIFO.

Refclk is used by the RocketIO MGT primitive to generate its serial clock at the desired bit rate (up to 2.5GHz)

Usrclk and **Usrclk2** are used by the RocketIO MGT primitive and by the sFPDP core to perform all controls related to TX and RX controllers. **Usrclk** and **Usrclk2** have both a phase and a frequency relationship that is derived from the way the RocketIO MGT primitive is configured (especially the width of the data path which is set to 32 bits. See [RD2] for more details).

The **Refclk**, **Usrclk,** and **Usrclk2** frequencies are dependent on the serial link bit rate. Although this core was tested and qualified for a 125 MHz reference clock leading to a 2.5 Gbit/s bit rate, it is possible to use other bit rate values. The reference clock is fed by an external dedicated clocking chip which provides a low jitter LVDS reference clock. The frequency of that reference clock must be $1/20^{th}$ of the desired bit rate. It is set using a dedicated attribute named oldTxbitrate that can take the values 2.5G, 2.125G or 1.063G, leading to bit rates of 2.5 Gbit/s, 2.125 Gbit/s or 1.063575 Gbit/s.

6.20.3.5 Throughput Monitoring

The sFPDP core implements a convenient way to monitor the effective data throughput of the TX controller. After the TX Controller is enabled, it counts during 65536 **Usrclk2** periods (roughly during more than 1ms) the number of periods it has spent sending user data. When this counting period is elapsed, it stores the counting result into the **TXTGH** field of the SLC Status register and restarts a new counting cycle. Thus, the effective data throughput can be computed as follows

$$TX_{Throughput} = TX_{Bandwidth} \frac{TXTGH}{65536}$$

$TX_{Throughput}$ is the effective user data throughput and $TX_{Bandwidth}$ the raw sFPDP core TX Bandwidth.

$$TX_{Bandwidth} = \frac{2}{Trefclk}[MByte/s]$$

T_{refclk} is the period, expressed in ns, of the reference clock used for the sFPDP core.

6.20.3.6 Generic Parameters

Although these parameters should not be modified by the developer, the table hereafter presents the generic values which are used for both simulation and synthesizable models of this sFPDP core version.

Parameter	Simulation	Synthesis	Comments
TX_INIT_DLY	x"00000100"	x"08000000"	Number of IDLE Words sent for Link Initialization
TX_IFRX_VAL	x"0010"	x"2000"	Number of IDLE frames sent for Link Initialization
RX_FIFO_SIZE	16384	16384	Size [Bytes] of the TX Buffer
TX_FIFO_DPH	9	9	Memory Depth of the TX Buffer
TX_IDLE_MAX	x"0F"	x"0F"	Max number of IDLE characters before IDLE Frame Insertion
TX_FIFO_THR	"000000001"	"000000001"	Defines the number of words before starting a Data Frame
PHY_INIT	"00001F"	"3D0900"	Defines the start up time for PAROLI transceivers
RX_INIT_DLY	x"00000080"	x"02000000"	Number of IDLE words to be correctly received for Link Initialization
RX_IFRX_VAL	x"0008"	x"1000"	Number of IDLE frames to be correctly received for Link Initialization
RX_BUFFER_SIZE	4093	4096	Size [Bytes] of the RX Buffer
RX_PDUI_TMO	x"03"	x"03"	Number of PDU Idle before a frame is aborted on the RX side

6.20.4 Register

SLC Registers refers to a set of three registers, which are embedded within each Serial FPDP Core. The address of these registers is defined by a base address (which may be mapped anywhere into the Customer-free address space) and an offset. The address offset is frozen within the core whereas the base address is defined by the SLC_BASE_ADD port value.

☛ **Note:** It is up to the user to carefully assign the SLC_BASE_Add value so that there will not be any address overlaps or other conflicts.

6.20.4.1 SLC Control Register

The SLC Control register defines the configuration and running modes of the Serial Front Panel Data Port Controller. It should not be accessed until the associated **`Usrclk`**/**`Usrclk2`** clocks are available and stable.

Register Space	Register Number	Register Address
Customer	**User_Defined(*)**	**User_Defined(*)**

(*) Register Address = SLC_Base_Add + 0x00

31	30	29..26	25	24	23..16
RSF	RRX	Reserved	RXP	TXP	RX FIFO Threshold

15..14	13	12	11..9	8
Reserved	FSY	SYF	Reserved	RFC

7	6	5..4	3	2	1	0
CRC	MST	*Reserved*	FWC	CPY	RXE	TXE

[0]	**TXE**	**RW**	TX Controller Enable. ‘0’: Disabled , ‘1’: Enabled
[1]	**RXE**	**RW**	RX Controller Enable. ‘0’: Disabled , ‘1’: Enabled
[2]	**CPY**	**RW**	TX Copy Mode. *TXE must be asserted.* ‘0’: Disabled , ‘1’: Enabled
[3]	**FWC**	**RW**	TX Flow Control. *TXE and RXE must be asserted.* 0’: Disabled, ‘1’: Enabled
[5..4]	**Reserved**	**RW**	Reserved: Do not use these bits in either reading or writing.
[6]	**MST**	**RW**	TX Copy Master Mode. *TXE and CPY must be asserted.* ‘0’: Disabled , ‘1’: Enabled
[7]	**CRC**	**RW**	CRC Encoding Enable. ‘0’: Disabled, ‘1’: Enabled
[8]	**RFC**	**RW**	RX Flow Control. *TXE and RXE must be asserted.* ‘0 : Disabled , ‘1’: Enabled
[11.. 9]	**Reserved**	**RW**	Reserved: Do not use these bits in either reading or writing
[12]	**SYF**	**RW**	Mark TX Frame with SYNC without data. ‘0 : Disabled, ‘1’: Enabled
[13]	**FSY**	**W**	Send a SYNC without data frame. ‘0’: Disabled, ‘1’: Enabled
[23..16]	**RXTHR**	**RW**	RX FIFO Threshold for Flow Control
[24]	**TXP**	**RW**	TX Polarity. ‘0’: Default Polarity, ‘1’: Invert Polarity
[25]	**RXP**	**RW**	RX Polarity. ‘0’: Default Polarity, ‘1’: Invert Polarity
[29-26]	**Debug**	**RW**	Reserved: Do not use these bits in either reading or writing.
[30]	**RRX**	**W**	Reset RX Controller. ’0 : Reset Flags, ’0’: Default
[31]	**RSF**	**RW**	Reset Status Flags. ’1’: Reset Flags, ’0’: Default

6.20.4.2 SLC Status Register

Register Space	Register Number	Register Address
Customer	**User Defined (*)**	**User Defined (*)**

(*) Register Address = SLC_Base_Add + 0x04

31..16
TXTGH

15	14	13	12	11	10	9	8
RES	CRE	RDE	DER	FTE	SYE	-	EER

7	6	5	4	3	2	1	0
-	RXK	-	TXK	RXR	TXR	RXL	TXF

[0]	**TXF**	**R**	TX Fault (Optical Transceiver Not Ready or Faulty).
[1]	**RXL**	**R**	RX Loss of Signal (Optical Transceiver Not Ready or Faulty).
[2]	**TXR**	**R**	TX Physical Layer Ready. ‘0’: Not initialized or faulty, ‘1’: Ready
[3]	**RXR**	**R**	RX Physical Layer Ready. ‘0’: Not initialized or faulty, ‘1’: Ready
[4]	**TXK**	**R**	TX Link Layer Ready. ‘0’: Not initialized or faulty, ‘1’: Ready
[6]	**RXK**	**R**	RX Link Layer Ready. ‘0’: Not initialized or faulty, ‘1’: Ready
[8]	**EER**	**RW**	TX Encoding Error. ‘0’: No error, ‘1’: Error Detected
[10]	**SYE**	**RW**	RX Loss of Sync Error. ‘0’: No error, ‘1’: Error Detected
[11]	**FTE**	**RW**	RX Format Error. ‘0’: No error, ‘1’: Error Detected

[12]	**DER**	**RW**	RX Data Encoding Error. ‘0’: No error, ‘1’: Error Detected
[13]	**RDE**	**RW**	RX Running Disparity Error. ‘0’: No error, ‘1’: Error Detected
[14]	**CRE**	**RW**	RX CRC Checking Error if relevant. ‘0’: No error, ‘1’: Error Detected
[15]	**RES**	**RW**	Reserved
[31..16]	**TXTGH**	**R**	TX Effective Throughput (Relevant only if TX is Enabled)

6.20.4.3 SLC Signal Register

Register Space	Register Number	Register Address
Customer	**User Defined (*)**	**User Defined (*)**

(*) Register Address = SLC_Base_Add + 0x08

31..9	8
-	RX_FOVF

7	6	5	4	3	2	1	0
RX_NRDY	RX_DIR	RX_PIO2	RX_PIO1	TX_NRDY	TX_DIR	TX_PIO2	TX_PIO1

[0]	**TX_PIO1**	**RW**	Value of the PIO1 signal to be transmitted over the Serial Link
[1]	**TX_PIO2**	**RW**	Value of the PIO2 signal to be transmitted over the Serial Link
[2]	**TX_DIR**	**RW**	Value of the DIR signal to be transmitted over the Serial Link
[3]	**TX_NRDY**	**RO**	Value of the NRDY signal received from the Serial Link
[4]	**RX_PIO1**	**RO**	Value of the PIO1 signal received from the Serial Link
[5]	**RX_PIO2**	**RO**	Value of the PIO2 signal received from the Serial Link
[6]	**RX_DIR**	**RO**	Value of the DIRY signal received from the Serial Link
[7]	**RX_NRDY**	**RW**	Value of the NRDY signal to be transmitted over the Serial Link
[8]	**RX_FOVF**	**RO**	Value of the RX_FOVF signal received from the Serial Link

6.20.5 Instantiation

This core is not instantiated in the base design example.

The TX_Data_ IF signals (refer to the Logic Symbol groups) should be connected to the User Core that is expected to transmit data packets.

The RX_Data_ IF signals (refer to the Logic Symbol groups) should be connected to the User Core that is expected to receive data packets. Even if the RX LocalLink is not used, it is highly recommended to wire the RX_DST_RDY_N port to ‘0’.

The PIO_IF could be left open with the exception of specific applications that need to control firmware operation by means of Serial FPDP Control Signals. These status signals are also available in the SLC Signal register.

The IB_IF signals should be connected to the Internal Bus.

The Clocks signals should be carefully wired depending on the location of the associated RocketIO MGT primitives and on the physical media the user wants to address (See Constraints sections).

Miscellaneous signals could be used for the User core to control the link availability for the TX path.

RocketIO IF signals are directly connected to the pad of the RocketIO MGT instance. No additional OBUF instantiation are required for those pins (TXP, TXN, RXP, and RXN).

Transceiver Control Signals only require the addition of an external pad buffer. Although there may be several ways inserting I/O buffers, the explicit IOBUF instantiation is the recommended solution.

Generic parameters are not available in this core versionsare frozen at the value described in the dedicated generic section.

6.20.6 Constraints

- Clock Constraint

The sFPDP core assumes an **IB_Clk** frequency of 33 MHz, a **Sysclk** frequency of 66 MHz, **Refclk** and **Usrclk** frequencies of 125 MHz, and a **Usrclk2** frequency of 62.5 MHz. These constraints must be defined for the clock manager core and are automatically propagated throughout the whole design.

Refclk must be allocated to a dedicated routing resource that provides a low jitter path from a clock buffer directly to the RocketIO MGT primitive. It must be directly wired without any BUFG primitive. The "Place and Route" tool automatically detects that connection and routes it accordingly.

Depending on the instantiated RocketIO MGT primitive, the user should connect the **Usrclk**/ **Usrclk2** and **Refclk** either to **Usrclka/Usrclk2a** and **RefclkA** ports of the clock manager core or on the **Usrclkb**/**Usrclk2b** and **RefclkB**.

- Input/Output Constraints

RocketIO instances	Transceiver	Clocking Edge	TXP	TXN	RXP	RXN
GT_X0Y1	SFP ODLA	RefclkA	A35	A36	A34	A33
GT_X2Y1	SFP ODLB	RefclkA	A31	A32	A30	A29
GT_X6Y1	PAROLI_L0	RefclkA	A14	A15	A13	A12
GT_X7Y1	PAROLI_L1	RefclkA	A10	A11	A9	A8
GT_X9Y1	PAROLI_L2	RefclkA	A6	A7	A5	A4
GT_X5Y1	PAROLI_L3	RefclkA	A18	A19	A17	A16
GT_X4Y1	PAROLI_L4	RefclkA	A23	A24	A22	A21
GT_X3Y1	PAROLI_L5	RefclkA	A27	A28	A26	A25
GT_X0Y0	PAROLI_L6	RefclkB	AW35	AW36	AW34	AW33
GT_X2Y0	PAROLI_L7	RefclkB	AW31	AW32	AW30	AW29
GT_X3Y0	PAROLI_L8	RefclkB	AW27	AW28	AW26	AW25
GT_X6Y0	PAROLI_L9	RefclkB	AW14	AW15	AW13	AW12
GT_X5Y0	PAROLI_L10	RefclkB	AW18	AW19	AW17	AW16
GT_X4Y0	PAROLI_L11	RefclkB	AW23	AW24	AW22	AW21

The sFPDP core contains one output (**PHY_TX_ENABLE**) and two inputs (**PHY_TX_FAULT**, **PHY_RX_LOSS**) that can be used to control directly a SFP optical transceiver. PAROLI transceivers should be controlled by an additional component that directly interfaces with these ports.

Please note that **PHY_TX_ENABLE** must be first inverted before being connected to the external pad.

6.20.7 Resource Utilization

Resource count and relative usage in the target Xilinx Virtex II Pro – XC2VP70-6FF1517:

Resources	Used	Available	Utilization
Function Generators	2722	66176	4.11%
CLB Slices	1361	33088	4.11 %
Dffs or Latches	2001	69068	2.90 %
Block RAMs	10	328	3.05 %

Note: Please note that the figures above are for an optimized version of the core that does not implement CRC Decoding on the RX path.

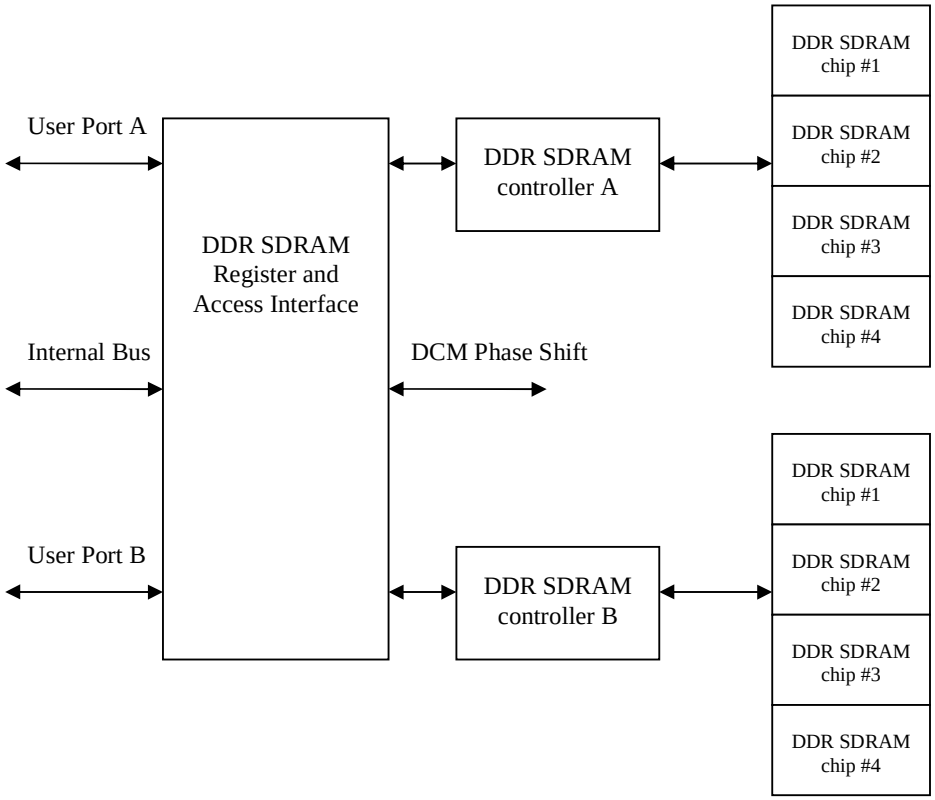
6.20.8 Version History

Date	FDK Version	Comments
July 05	Beta 3	Initial Version
March 06	Beta 7	Description updated.

6.21 DDR Memory Interface

The P512MB Memory Option provides two independent DDR SDRAM (“Dual-Data Rate, Synchronous Dynamic Random Access Memory”) memory banks that are accessed through two independent interfaces: DDRA and DDRB. Each memory bank is defined by four DDR SDRAM devices each with a 16 bits wide data bus. Thus the data bus of each bank is 64 bits wide. The DDR SDRAM clock runs at 166 MHz and provides a usable bandwidth of 2 GB/s.

The size of each DDR SDRAM device is 512 Mbits, leading to a total of 256 MB per bank.



6.21.1 Functional Description

After a bit file is loaded, the DDR SDRAM memories must first be initialized. Once the initialization is completed, the access to a DDR SDRAM bank can be performed either from the Internal Bus (for access to/from the PCI bus) or from the User Port.

The User Port and the Internal Bus port of a bank cannot be simultaneously active. The active port can be set independently for each bank. It is not possible to simultaneously access both memory banks through the Internal Bus.

The User Port is optimized for high throughput transfers using Burst mode accesses. Single access is also available but will be less efficient than single access to the Dual Port SRAM. The DDR memory interface core contains multiple read/write buffers in order to sustain the 2 GB/s throughput. The buffer handling is automatically performed by the core. In the same way the DDR memory refresh cycles are automatically managed by the core.

The Internal Bus port enables DMA readout through the PCI backplane at up to 132 MB/s.

The DDR memory interface core includes a self-test. This self-test covers the entire memory depth and is performed at full speed with different test patterns. This test can be run through the AcqirisAnalyzer application if the base design is loaded in the FPGA.

6.21.1.1 Initialization

After the bit file is loaded the DDR memory initialization begins automatically. It will end several microseconds after the user program releases the **`SGReset_n`** bit of the control register of the core **`acq_ctr_reg`**. When the initialization is completed, the **`Init_DDR_Done`** bit of the DDRStatus register is set to ‘1’. The user program should wait for it before attempting to access the DDR memory.

6.21.1.2 Minimum Number of Transfers

The core **ddr_interface** configures the memory devices to use burst4 transfers. This means that a minimum of four words of 64 bits are transferred to or from the memory when accessing the memory. It therefore requires a minimum of two transfers on the User Port (2 x 128 bits) and a minimum of 8 transfers on the Internal Bus port (8 x 32 bits).

6.21.1.3 Read Access Time

The core **ddr_interface** always reads the memory starting at an address aligned to 0 (modulo 128) and reads entire buffer blocks of 128 x 128 bits, even if the start address is not aligned to 0. The address always refers to words of 128 bits. The time to access the first data is then directly dependent on its alignment in the read buffer. As soon as the required data is available, the user is informed with a data valid signal.

6.21.1.4 Port Selection

The bit **LB_Get_Ctrl** of the DDRControl register selects which port will be the active port. Of course the selection must be done prior to accessing the memory bank.

6.21.1.5 User Port

Each User Port has independent clock, data, address, read / write control ports. The data bus width is 128 bits. The user can connect the clock to any required frequency up to 133 MHz as long as the write rate does not exceed the maximum of 2 GB/s, or 128 bits at 125 MHz. This average value must not be exceeded when completely filling a single buffer of 128 x 128 bits. For example, if 128 words of 128 bits were written at a rate of 133 MHz, there should be no attempt to write data for at least 8 clock cycles.

The User Port provides two kinds of accesses:

Single Access: It corresponds to a single memory access (burst4) which always results into a double transfer at the User Port (2 x 128 bits).

Burst Access: It corresponds to *n* consecutive memory accesses (burst4) which always results into *2n* transfers at the User Port (n x 2 x 128 bits).

Any User Port access begins by activating the signal **UPx_ADS** for one clock cycle and ends by activating the signal **UPx_End** for one clock cycle. The transfer direction, type, and start address are sampled when **UPx_ADS** is active.

***NOTE*: The UPX_Address address is defined on a basis of 128-bit words, which differs from the internal bus address.**

The start address on the **UPX_Address** bus must always be 0 modulo 2 (i.e. be a multiple of 2).

In the case of write access, the signal **UPx_WriteEn** should be activated when valid data are present on the **UPx_DataW** bus.

In the case of read access, the signal **UPx_DataR_Valid** is activated when data are available. The user should then activate the signal **UPx_DataR_En** to get the data available on **UPx_DataR** with a latency of one clock.

6.21.1.6 Internal Bus Port

The Internal Bus port provides access to the control register, status register, and to the memory. Access to the register uses the direct access mode of the Local Bus interface. Access to the memories uses indirect addressing with DMA capability for fast readout.

Each memory bank is seen through the Internal Bus as a memory space of 64M words of 32 bits.

The start address is the content of the indirect address register of the Local Bus interface.
***NOTE*: The address is in bytes, which differs from the user port address.**
Valid address are (0 + n * 4), where n [0..64M] is the targeted 32-bit word.

Reading the DDR memory can start at any valid address.

The DDR core writes data to the memory with a minimum burst of four 64-bit words. Therefore the DDR core accumulates eight 32-bit words into an intermediate buffer of 256 bits prior to writing. In case an access ends before 8 words are accumulated, the last intermediate buffer remains in the DDR core and is not written to the memory. Any write access to an address 0x0 modulo 0x20 (8x32 bits) overwrites the intermediate buffer even if it was previously partially filled.

6.21.1.7 Internal Bus Port Address versus User Port Address

Internal Bus port address [bytes]	User Port Address[128 bits]
0x00	0x0, Data(bits 31 to 0)
0x04	0x0, Data(bits 63 to 32)
0x08	0x0, Data(bits 95 to 64)
0x0C	0x0, Data(bits 127 to 96)
0x10	0x1, Data(bits 31 to 0)
0x14	0x1, Data(bits 63 to 32)
0x18	0x1, Data(bits 95 to 64)
0x1C	0x1, Data(bits 127 to 96)

6.21.1.8 DDR SDRAM Clock Structure

The clock speed of the DDR SDRAM devices is 166 MHz. The DDR SDRAM controller uses three different 166 MHz clocks generated from 3 different DCMs to ensure the DDR SDRAM device timing.

- **SYS_DDR_CLK**: is used to generate the DDR command and the DDR data output.
- **STB_CLK**: generates the DDR SDRAM clock and the DQS strobes.
- **SMP_CLK**: is internally used, to set the sampling point of the DDR read data properly

All these clocks are shared among the two DDR-SDRAM controllers.

The **STB_CLK** and **SMP_CLK** can be individually shifted. This allows exact matching of the timing of the DDR SDRAM devices to compensate the board layout delays.

The **STB_CLK** can be shifted by using the phase shift option **DCM_EXT_PS**. The **SMP_CLK** can be shifted by using the phase shift option **DCM_SMP_PS**. The phase shift affects always both DDR SDRAM controllers. It can't be set individually.

NOTE: **The phase shift is fixed by design. Developers should not modify the default value**.

6.21.1.9 Self-Test

The DDR memory interface core also implements a self-test that is used in production for testing the DDR-SDRAM memory devices. The goal of this test is to fill the whole memory with four types of patterns: all '0', all '1', '0' → '1' → '0', and ramps on 16 bits (for 160 bits, 10 ramps of 16 bits) and then to read back the memory to check the proper behavior.

The test is started by writing the bit **Start_Self_Test** in the DDRTestControl register. This test can be run in different modes. The simplest mode performs the entire test and counts the number of errors. This mode is selected by setting the bit **Self_Test_Autocont** in the DDRTestControl register before running the test. When the test is complete, the bit **DDR_Ready** of the DDRStatus register is set to '1'. The bit **Self_Test_Ok** of the DDRTestStatus register is '1' if the test was successful, otherwise it is '0'.

***NOTE*: Developers need not know more details. The tests can be executed by running Acqiris Analyzer with the Base Design firmware.**

6.21.2 Instantiation

The DDR memory interface core is already instantiated in the base design example. It is always associated with its companion, **ddr_interface_buffer**, which handles the Xilinx IO primitives.

6.21.3 Port Description

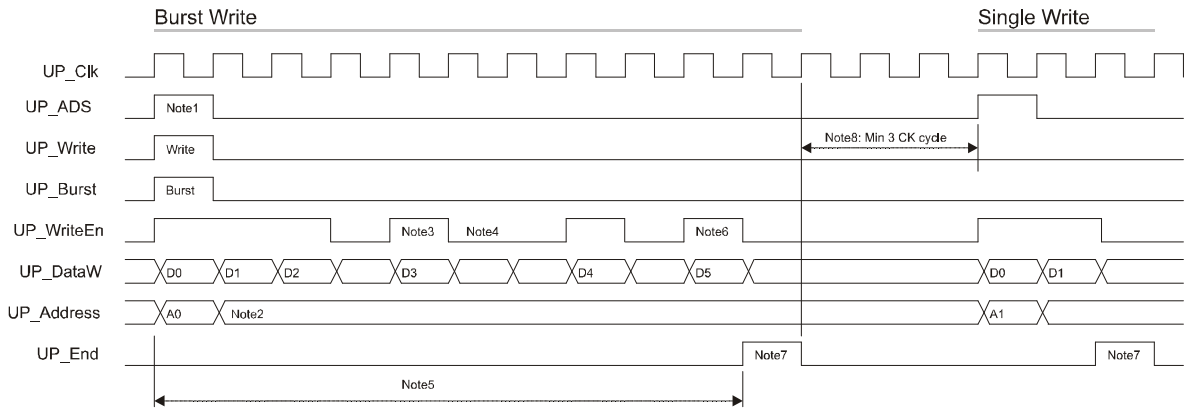
Signal	Size	Type	Short Description
INTERNAL BUS			

Signal	Size	Type	Short Description
IB_Customer **IB_Dirsel** **IB_Write** **IB_Rdy** **IB_TimeO** **IB_Addr** **IB_DataW** **IB_DataR** **IB_IndirCtr** **IB_Valid** **IB_End**	1 1 1 1 1 32 32 32 1 1 1	In In In Out In In In Out In In In	Must be connected to the IB-BUS signal with the same name. For details please refer to the description of the IB-BUS.
IB_Clk	1	In	Internal Bus clock, must be connected to **lbclkg**, the Local Bus clock.
Reset	1	In	Reset, must be connected to the general reset **Dreset**.
USER Port: x=A or B stand for User Port A or User Port B			
UP_Clk	1	In	Memory port clock. It can be set to any frequency up to 125 MHz. Up to 133 MHz is also allowed, but only with the maximum sampling rate of 2 GS/s
UPx_ADS	2x1	In	Address strobe. When '1', the value on **UPx_Address** is used. It is stored in the auto increment address register and the access will be started.
UPx_Write	2x1	In	Transfer direction: '1' for writing to the memory, '0' for reading.
UPx_Address	2x26	In	Start Address. The address value is in units of 128-bit words.
UPx_Burst	2x1	In	It must be set to '1' for burst transfer, '0' for single transfer.
UPx_End	2x1	In	Each access must terminate by setting **UPx_End** to '1' for one clock period.
UPx_DataW	2x127	In	Bus for data write
UPx_WriteEn	2x1	In	DataW write enable
UPx_DataR	2x127	Out	Bus for data read
UPx_DataR_Valid	2x1	Out	Data read valid signal. '1' indicates that the controller has valid data in the read buffer.
UPx_DataR_En	2x1	In	Data read request. Read a next valid data from the read buffer. The user must first wait for **UPx_DataR_Valid**.
OUTPUT to DDR SDRAM MEMORY: x=A or B stand for Port A or Port B			
DDRx_CK	2x1	Out	Portx DDR SDRAM Clock Output
DDRx_CK_N	2x1	Out	Portx DDR SDRAM negative Clock Output
DDRx_CKE	2x1	Out	Portx DDR SDRAM Clock enable (is fixed '1')
DDRx_ADDR	2x13	Out	Portx DDR SDRAM Address
DDRx_BA	2x2	Out	Portx DDR SDRAM Bank Address
DDRx_CS_N	2x1	Out	Portx DDR SDRAM Chip Select
DDRx_WE_N	2x1	Out	Portx DDR SDRAM Write Enable
DDRx_RAS_N	2x1	Out	Portx DDR SDRAM Row Address Strobe
DDRx_CAS_N	2x1	Out	Portx DDR SDRAM Column Address Strobe
DDRx_LDM	2x1	Out	Portx DDR SDRAM Lower Data Mask (is fixed to '0')
DDRx_UDM	2x1	Out	Portx DDR SDRAM Upper Data Mask
DDRx_DQ	2x127	InOut	Portx DDR SDRAM bidirectional Data Bus
DDRx_LDQS	2x4	Out	Portx DDR SDRAM Lower Data Strobe
DDRx_UDQS	2x4	Out	Portx DDR SDRAM Upper Data Strobe
DDRx_CS_N_RDBACK	2x1	In	Portx DDR SDRAM Chip Select Readback Signal

Signal	Size	Type	Short Description
DDRx_WE_N_RDBACK	2x1	In	Portx DDR SDRAM Write Enable Readback Signal
DDRx_RAS_N_RDBACK	2x1	In	Portx DDR SDRAM Row Address Strobe Readback Signal
DDRx_CAS_N_RDBACK	2x1	In	Portx DDR SDRAM Column Address Strobe Readback Signal

6.21.4 User Port Timing Diagrams: Burst write and Single write

Note1: **UP_ADS** initiates the transfer. **UP_Write**, **UP_Burst,** and **UP_Address** must be valid



simultaneously to **UP_ADS**.

Note2: The start address **UP_Address** must be 0 modulo 2.

Note3: **UP_WriteEn** enables the transfer of the actual value of **UP_DataW**.

Note4: The average writing rate must not exceed 2 GB/s for longer than 128 transfers.

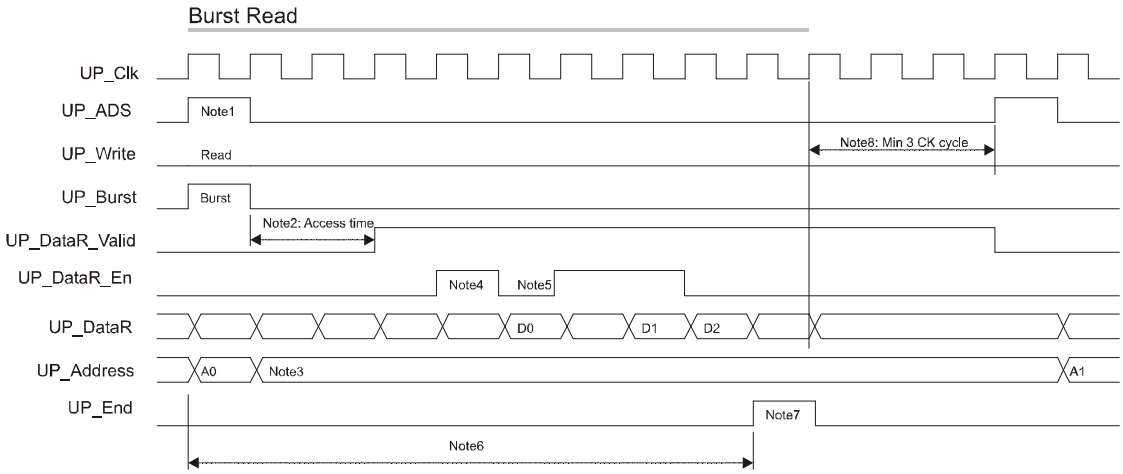
Note5: The number of transfers in a burst is not limited.

Note6: The number of transfers in a burst must be an even value.

Note7: **UP_End** must be '1' for one clock cycle in order to terminate the transfer.

Note8: Any sequence of read or write, burst or single must be separated by at least 3 clock cycles.

6.21.5 User Port Timing Diagrams: Burst read



Note1: **UP_ADS** initiates the transfer. **UP_Write**, **UP_Burst,** and **UP_Address** must be valid simultaneously with **UP_ADS**.

Note2: The access time depends on the alignment of the start address **UP_Address** with respect to the read buffer.

Note3: The start address **UP_Address** must be 0 modulo 2.

Note4: **UP_DataR_En** enables reading. It must be activated only after valid data are present in the read buffer of the DDR core. This is true when the signal **UP_DataR_Valid** becomes '1'. The data will be present on the signal **UP_DataR** with a latency of one clock cycle.

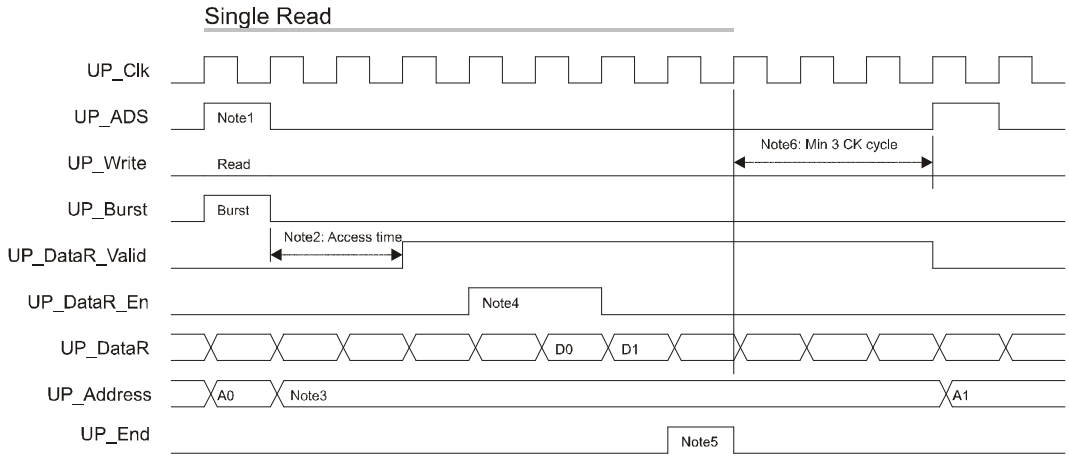
Note5: The average reading rate must not exceed 2 GB/s for longer than 128 transfers.

Note6: The number of transfers in a burst is not limited.

Note7: **UP_End** must be '1' for one clock cycle in order to terminate the transfer.

Note8: Any sequence of read or write, burst or single must be separated by at least 3 clock cycles.

6.21.6 User Port Timing Diagrams: Single Read



Note1: **UP_ADS** initiates the transfer. **UP_Write**, **UP_Burst,** and **UP_Address** must be valid simultaneously with **UP_ADS**.

Note2: The access time depends on the alignment of the start address **UP_Address** with respect to the read buffer.

Note3: The start address **UP_Address** must be 0 modulo 2.

Note4: **UP_DataR_En** enables reading. It must be activated only after valid data are present in the read buffer of the DDR core. This is true when the signal **UP_DataR_Valid** becomes '1'. The data will be present on the signal **UP_DataR** with a latency of one clock cycle.

Note5: **UP_End** must be '1' for one clock cycle in order to terminate the transfer.

Note6: Any sequence of read or write, burst or single must be separated by at least 3 clock cycles.

6.21.7 Registers

Each register exists both for the DDR BankA and the DDR BankB. The first value in the column "Register Number" and in the column "Register Address" is for the register of BankA, the second value is for the register of BankB.

Each bit has a different meaning depending on the access direction.

6.21.7.1 DDRControl

Register Space	Register Number	Register Address
Customer	**44/52 (*)**	**0x22B0/0x22D0 (*)**

Note (*) : the two values refer to the register of BankA and BankB (A/B)

31..9	8
--	LB_Get_Ctrl

7	6	5..4	3..2	1	0
--	DDR_SDRAM_Sim	DDR_SDRAM_Size	--	Clr_Overflow_Flag	--

[1]	**Clr_Overflow_Flag**	**W**	Clears the DDR write buffer overflow flag. Overflows can occur when the write speed is more than 2 GB/s.
[5..4]	**DDR_SDRAM_Size**	**RW**	It must be set to '0': 256 MB (32M x 64 bits)
[6]	**DDR_SDRAM_Sim**	**RW**	Select Simulation mode: - in order to reduce the simulation time - to perform automatic tests on reduced memory size **0** Normal operation **1** Simulation
[8]	**LB_Get_Ctrl**	**RW**	Select the port for subsequent DRAM memory access **0** User port (UPx port) **1** Internal bus port (indirect address)

6.21.7.2 DDRStatus

Register Space	Register Number	Register Address
Customer	**44/52 (*)**	**0x22B0/0x22D0 (*)**

Note (*) : the two values refer to the register of BankA and BankB (A/B)

31..16	15..13	12
--	--	Stop_Rd_Flag

11..9	8	7	6	5..4
	LB_Get_Ctrl	--	DDR_SDRAM_Sim	DDR_SDRAM_Size

3	2	1	0
--	Wr_Buffer_Overrun	DDR_Ready	Init_DDR_Done

[0]	**Init_DDR_Done**	**R**	After power up the DDR SDRAM devices have to be initialized first. **Init_DDR_Done** goes to '1' as soon as the initialization process is finished
[1]	**DDR_Ready**	**R**	Self-test status. **0** State after power up **1** Self-test process is finished
[2]	**Wr_Buffer_Overrun**	**R**	Write Buffer Overrun. Overruns can occur when the data write speed is more than 2 GB/s
[5..4]	**DDR_SDRAM_Size**	**RW**	DRAM memory size **01** 256 MB (32M x 64 bits) **others** Not available
[6]	**DDR_SDRAM_Sim**	**RW**	Select Simulation mode: - in order to reduce the simulation time - to perform automatic tests on reduced memory size **0** Normal operation. **1** Simulation.
[8]	**LB_Get_Ctrl**	**RW**	Select port for subsequent DRAM memory access **0** User port (UPx port)

1 Internal bus port (indirect address)

[12] **Stop_Rd_Flag** **R** Status for read access:

0 A read access is active.

1 A read access is stopped. The DDR core is ready for another access

6.21.7.3 DDRTestControl

Register Space	Register Number	Register Address
Customer	**45/53 (*)**	**0x22B4/0x22D4 (*)**

Note (*) : the two values refer to the register of BankA and BankB (A/B)

31	30-8	7..4
Self_Test_Short		Self_Test_NB_Active

3	2	1	0
Self_Test_Autocont	Self_Test_Stop	Self_Test_Continue	Start_Self_Test

[0] **Start_Self_Test** **W** Writing a '1' starts a self-test

[1] **Self_Test_ Continue** **W** Writing a '1' continues a self-test. In case of a DDR SDRAM access error detected by the self-test process, the test is paused and can be continued by this bit.

[2] **Self_Test_Stop** **W** Writing a '1' stops a self-test and the DDR SDRAM controller goes into normal operation mode. In case of a DDR SDRAM access error detected by the self-test process, the test is paused and can be stopped by this bit.

[3] **Self_Test_ Autocont** **W** Writing a '1' puts the DDR controller into auto-continue mode. In auto-continue mode, the self-tests are executed on the whole memory. They are not paused in case of an access error. At the end of a self-test, the number of errors is written in the self-test error counter register (50/58).

[7..4] **Self_Test_NB_ Active** **W** Test pattern number (multiple selections possible). The pattern number will be processed one after another.

Bit4 Data pattern '0'

Bit5 Data pattern '1'

Bit6 Data pattern '0' – '1' – '0' – '1' – .....

Bit7 Data pattern ramp

[31] **Self_Test_Short** **W** Writing a '1' force the self-test to be executed on 64K memory position word instead of on the whole memory.

6.21.7.4 DDRTestStatus

Register Space	Register Number	Register Address
Customer	**45/53 (*)**	**0x22B4/0x22D4 (*)**

Note (*) : the two values refer to the register of BankA and BankB (A/B)

31	30	29	28
--	Self_Test_Timeout	Self_Test_Busy	Self_Test_Ok

27..25	24..0
--	Self_Test_Addr_KO

[24..0] **Self_Test_Addr_KO** **R** Address location in the DDR SDRAM

memory where an access mismatch was found. Check this register when **Self_Test_Busy** goes to '0' and **Self_Test_Ok** indicates '0'.

[28]	**Self_Test_Ok**	**R**	Self-test pass / fail status **0** Self-test was not successful **1** Self-test was successful
[29]	**Self_Test_Busy**	**R**	Self-test busy status **0** Self-test is finished. When finished, check **Self_Test_Ok**. **1** Indicates a running self-test
[30]	**Self_Test_Timeout**	**R**	When equal to '1', this bit indicates that the current self-test did not terminate. This is normally due to an incorrect phase shift value for the DCM Ext or the DCM Smp. When internal timeout occurs, the **Self_Test_Ok** and **Self_Test_Busy** bits are deactivated

6.21.7.5 DDRTestData0

Register Space	Register Number	Register Address
Customer	**46/54 (*)**	**0x22B8/0x22D8 (*)**

Note (*) : the two values refer to the register of BankA and BankB (A/B)

31..0
Self_Test_Data_KO_1(31:0)

[31..0]	**Self_Test_Data_KO_1**	**R**	Data bits (31:0) at the DDR SDRAM address location where a mismatch was found

6.21.7.6 DDRTestData1

Register Space	Register Number	Register Address
Customer	**47/55 (*)**	**0x22BC/0x22DC (*)**

Note (*) : the two values refer to the register of BankA and BankB (A/B)

31..0
Self_Test_Data_KO_2 (63:32)

[31..0]	**Self_Test_Data_KO_2**	**R**	Data bits (63:32) at the DDR SDRAM address location where a mismatch was found

6.21.7.7 DDRTestData2

Register Space	Register Number	Register Address
Customer	**48/56 (*)**	**0x22C0/0x22E0 (*)**

Note (*) : the two values refer to the register of BankA and BankB (A/B)

31..0

Self_Test_Data_KO_3(95:64)

[31..0] **Self_Test_Data_KO_3** **R** Data bits (95:64) at the DDR SDRAM address location where a mismatch was found

6.21.7.8 DDRTestData3

Register Space	Register Number	Register Address
Customer	**49/57 (*)**	**0x22C4/0x22E4 (*)**

Note (*) : the two values refer to the register of BankA and BankB (A/B)

31..0
Self_Test_Data_KO_4(127:96)

[31..0] **Self_Test_Data_KO_4** **R** Data bits (127:96) at the DDR SDRAM address location where a mismatch was found

6.21.7.9 DDRTestCounter

Register Space	Register Number	Register Address
Customer	**50/58 (*)**	**0x22C8/0x22E8 (*)**

Note (*) : the two values refer to the register of BankA and BankB (A/B)

31..25	24..0
--	Self_Test_Error_Counter

[24-0] **Self_Test_Error_Counter** **R** In auto-continue mode, these bits indicate the number of errors during the last self-test

6.21.7.10 DDRClockControl

Developers should not modify nor write to this register. The proper behavior of the DDR memory could be altered. This register is common for both DDR Banks.

Register Space	Register Number	Register Address
Customer	**60**	**0x22F0**

31..26	25	24	23..16
--	DCM_Smp_Change_Start	DCM_Smp_PS_IncDec	DCM_Smp_PS_Offset

15..10	9	8	7..0
--	DCM_Ext_Change_Start	DCM_Ext_PS_IncDec	DCM_Ext_PS_Offset

[7..0] **DCM_Ext_PS_Offset** **W** Phase shift offset of the DCM Ext
Phase shift width is: t_{clk_ext} * (DCM phase shift / 256)

[8] **DCM_Ext_PS_IncDec** **W** Phase shift is decremented (‘0’) or incremented (‘1’) by the value of **DCM_Ext_PS_Offset**

[9] **DCM_Ext_Change_Start** **W** Start DCM Ext Phase shift

- **0** no action
- **1** Phase shift process of the DCM Ext is started

[23..16] **DCM_Smp_PS_Offset** **W** Phase shift offset of the DCM Smp
Phase shift width is: t_{clk_smp} * (DCM phase shift / 256)

[24] **DCM_Smp_PS_IncDec** **W** Phase shift is decremented (‘0’) or incremented (‘1’) by the value of **DCM_Smp_PS_Offset**

[25] **DCM_Smp_Change_Start** **W** Start DCM Smp Phase shift

0 no action

1 Phase shift process of the DCM Smp is started

6.21.7.11 DDRClockStatus

Register Space	Register Number	Register Address
Customer	**60**	**0x22F0**

31..24	23..18	17	16
DCM_Smp_Done_cnt		DCM_Smp_Overflow	DCM_Smp_Busy

15..8	7..2	1	0
DCM_Ext_Done_cnt	--	DCM_Ext_Overflow	DCM_Ext_Busy

[0] **DCM_Ext_Busy** **R**
- **0** Phase shift of DCM Ext is idle or finished
- **1** Phase shift of DCM Ext is in progress

[1] **DCM_Ext_Overflow** **R** When equal to '1', this bit indicates the overflow of the DCM Ext phase shift

[15..8] **DCM_Ext_Done_cnt** **R** Phase shift counter of DCM Ext. Counts the actual number of **DCM_Ext_PS_Value**, which are executed

[16] **DCM_Smp_Busy**
- **0** Phase shift of DCM Smp is idle or finished
- **1** Phase shift of DCM Smp is in progress

[17] **DCM_Smp_Overflow** **W** When equal to '1', this bit indicates the overflow of the DCM Smp phase shift

[31..24] **DCM_Smp_Done_cnt** **W** Phase shift counter of DCM Smp. Counts the actual number of **DCM_Smp_PS_Value**, which are executed.

6.21.8 Accessing the DDR SDRAM Memory

DDR memory content could be read or write by the user program using the Indirect Addressing register. The Indirect Address Register and Buffer Identifier Register should be set prior read or write the memory.

6.21.8.1 DDR A -memory

Register Space	Register Number	Register Address	Buffer Identifier
Customer	**0**	**0x2200**	**0x00**

31..0
RWData

[31..0] **RWData** **RW** Data format depends on the customer application.

6.21.8.2 DDR B –memory

Register Space	Register Number	Register Address	Buffer Identifier
Customer	**0**	**0x2200**	**0x01**

31..0
RWData

[31..0] **RWData** **RW** Data format depends on the customer application.

6.21.9 Constraints

- Clock Constraint

This core assumes an **IB_Clk** frequency of 33 MHz. This constraint must be defined for the clock manager component and is automatically propagated throughout the whole design.

- User Port Clock Constraints

This core assumes a **Sysclk** frequency of maximum 133 MHz. This constraint must be defined for the clock manager component and is automatically propagated throughout the whole design.

- DDR A&B Port Clock Constraint

This core assumes a DDR clock frequency of 166 MHz. This constraint must be defined for the clock manager component and is automatically propagated throughout the whole design.

- Output and Input Signal constraints

They are set in the design, using Xilinx primitives.

6.21.10Resource Utilization

Resource count and relative usage in the target Xilinx Virtex II Pro – XC2VP70-6FF1517:

Resources	Used	Available	Utilization
Block RAMs	20	328	6.1%
Function Generators	4120	66176	6.2%
CLB Slices	3167	33088	9.6%
Dffs or Latches	6334	69068	9.2%

6.21.11Version History

Date	FDK Version	Comments
February 06	Beta 6	New core
March 06	Beta 7	Description updated.

6.22 DDR Memory Control Example

The block **ddr_ctr_test_only** is delivered as an example of interfacing the **ddr_interface** core. Developers should remove it or adapt it for their own purpose. Its main function is to verify the correct behavior of the two user ports of the **ddr_interface** block. This is done by exercising fixed pattern with short burst (2x128 bits) and long burst (256x128 bits).

6.22.1 Port Description

Signal	Size	Type	Short Description
IB_Customer **IB_Dirsel** **IB_Write** **IB_Rdy** **IB_TimeO** **IB_Addr** **IB_DataW** **IB_DataR**	1 1 1 1 1 31 32 32	In In In Out In In In Out	Must be connected to the IB-BUS signal with the same name. For details please refer to the description of the IB-BUS.
IB_Clk	1	In	Internal Bus clock, must be connected to **lbclkg**, the Local Bus clock.
DReset	1	In	Reset, must be connected to the general reset **Dreset**.
UPx_ADS **UPx_Write** **UPx_Address** **UPx_Burst** **UPx_End** **UPx_DataR_En** **UPx_DataW** **UPx_WriteEn** **UPx_DataR** **UPx_DataR_Valid**	2x1 2x1 2x20 2x1 2x1 2x1 2x128 2x1 2x128 2x1	Out Out Out Out Out Out Out Out In In	Must be connected to the corresponding pins of the core **ddr_interface**. For details please refer to the description of the core **ddr_interface**.
Sysclk	1	In	It must be connected to the system clock **Sysclk**

6.22.2 Registers

6.22.2.1 DDREControl

Register Space	Register Number	Register Address
Customer	**67**	**0x230C**

31..6	5	4	3..2	1	0
--	Burst2	StartTest	--	SmallMem	--

[1] **SmallMem** **W** It must be set '1' to perform user port tests on reduced memory size in order to reduce the simulation time.

- **0** Normal operation
- **1** Simulation

[4] **StartTest** **W** Setting to '1' starts the built-in test. The **ddr_interface** core must be set in the user port mode before starting the test.

[5] **Burst2** **W** Access type for the built-in test:

- **0** burst256 access (256 x 128 bits).
- **1** burst2 access (2 x 128 bits)

6.22.2.2 DDREStatus

Register Space	Register Number	Register Address
Customer	**67**	**0x230C**

31..12	11	10	9	8
--	--	TestErrorB	TestFailB	TestEndB

7	6	5	4	3..1	0
--	TestErrorA	TestFailA	TestEndA	--	--

[4]	**TestEndA**	**R**	Set to '1' when the test of the DDR bank A is complete. Reset to '0' when the test is started.
[5]	**TestFailA**	**R**	Latches to '1' when at least one error occurs. Reset to '0' when the test is started.
[6]	**TestErrorA**	**R**	Set to '1' for a single clk period when an individual test fails.
[8]	**TestEndB**	**R**	Set to '1' when the test of the DDR bank B is complete. Reset to '0' when the test is started.
[9]	**TestFailB**	**R**	Latches to '1' when at least one error occurs. Reset to '0' when the test is started.
[10]	**TestErrorB**	**R**	Set to '1' for a single clk period when an individual test fails.

6.22.3 Resource Utilization

Resource count and relative usage in the target Xilinx Virtex II Pro – XC2VP70-6FF1517:

Resources	Used	Available	Utilization
Function Generators	1020	66176	1.5%
CLB Slices	510	33088	1.5%
Dffs or Latches	661	69068	1.0%

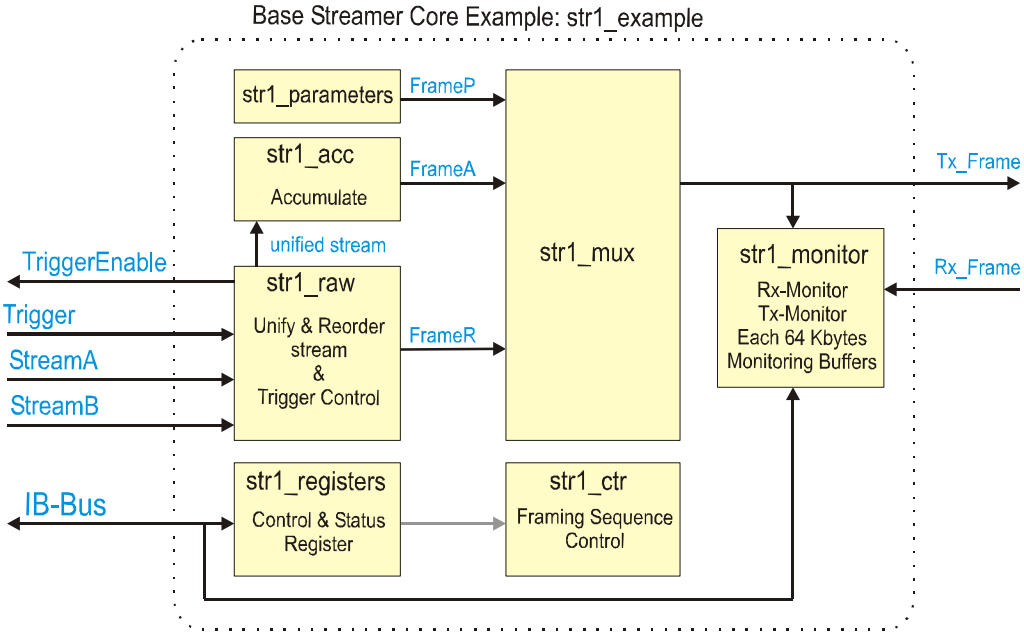
6.22.4 Version History

Date	FDK Version	Comments
February 06	Beta 6	New core
March 06	Beta 7	Description updated.

6.23 Base Streamer Example

The block **`str1_example`** is instantiated in the Base Streaming Base Design described in the paragraph 5.5. It is an example of how to build different types of data frame and how to send the frames to the serial front panel data port controller **`slc_controller`** that is instantiated in the block **`slc1_interface.`**

This core also implements the two monitoring buffers, the TX-Monitor buffer and the RX-Monitor buffer. Both can be read by program and are useful for verification purpose.



The block **`str1_raw`** mixes the stream A and B to a single unified stream in an interleaved fashion. The unified stream could also be reordered in order to align the trigger position to the 16 samples data blocks. Once enabled and after each trigger, the block **`str1_raw`** generates and sends a Raw Data frame to the FrameR port of the Frame multiplexer. The Raw Data frame has a 128 bits header followed by a programmable number of raw samples [or bytes].

The unified stream is also connected to the block **`str1_acc`**. This block perform the store accumulate function. The number of acquisition to accumulate is programmable from 1 to 256. When the accumulation completes, the block **`str1_acc`** generates and sends an Accumulated Data frame to the FrameA port of the frame multiplexer. The Accumulated Data frame has a 128 bits header followed by a number of accumulated data equal to the number of programmed of Raw Data samples. In bytes, this corresponds to double the number of programmed samples because the accumulated data is now on 16 bits instead of the 8 bits initial raw Data.

After the Accumulated Data frame has been sent to the frame multiplexer, the block **`str1_parameters`** generates and sends a Parameter Data frame to the FrameP port of the Frame multiplexer. The Parameter Data Frame has a 128 bits header followed by a fixed number of parameters. In the example, there is 256 parameters, each parameters is 128 bits wide.

The Frame Multiplexer simply multiplexes one of the input frames to TX-Frame output. The TX-Monitor Buffer is a spy on the TX-Frame and can be read by the user program.

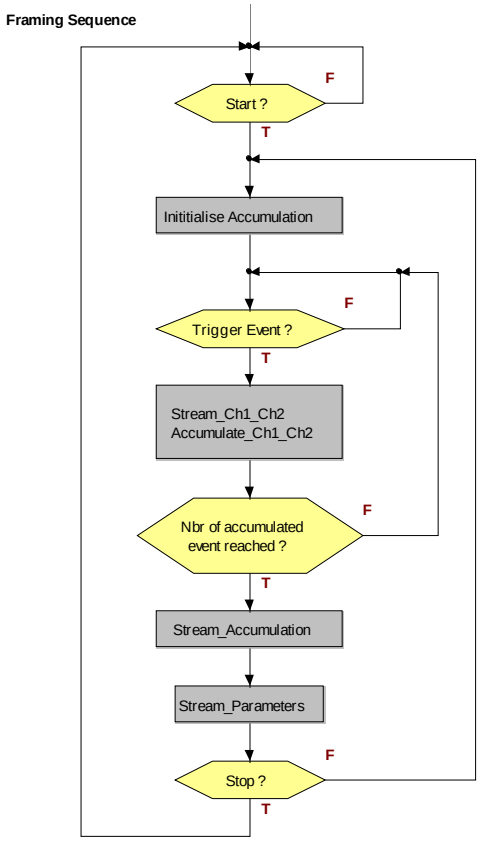
For verification, the front-panel TX output of the optical link could be looped back to the RX input of the optical link and the received data could be monitored with the RX-Monitor Buffer and readout by the user program. This makes possible to verify received data are strictly identical to the transmitted data.

Once enabled, the TX- and the RX-Monitor buffer will store one Raw Data frame, one Accumulated Data frame and one Parameter Data frame and will stop until restarted. Two readable status bits, one for RX and one for TX buffer, indicate when the monitor buffers are ready for readout.

Monitoring can be enabled or disabled at any time. Monitoring has no effect on the streaming process.

6.23.1 Framing Sequence Flow Chart

The streaming or framing sequence is controlled by the main state machine **`str1_ctr`**.



6.23.2 Raw Data Frame

The Raw Data Frame consists of a 128 bits header followed by a programmable number of raw data. Each raw data is 8 bits.

Header, 1x 128 bits

127..120	119..64	63..0
`FrameType=x'00'`	`TimeStamp`	`Fixed to 0`

Raw Data

7..0	7..0	…..	7..0
`RawData#1`	`RawData#2`	…	`RawData#n`

[127..120]	**`FrameType`**	Fixed to X'00', indicates the frame is a Raw Data frame
[119..64]	**`TimeStamp`**	The TimeStamp value indicate the arrival time of the trigger
[7..0]	**`RawData`**	Raw data. The number of raw data is programmable with the bits **SFS** of the Base Streamer Configuration Register. The values are signed 8 bits values.

6.23.3 Accumulated Data Frame

The Accumulated Data Frame consists of a 128 bits header followed by a programmable number of accumulated data. Each accumulated data is 16 bits.

Header

127..120	119..64	63..0
`FrameType=x'01'`	`TimeStamp`	`Fixed to 0`

Accumulated Data

15..0	15..0	…..	15..0
`AccData#1`	`AccData#2`	…	`AccData#n`

[127..120]	**FrameType**	Fixed to X'01', indicates the frame is an Accumulated Data frame
[119..64]	**TimeStamp**	The TimeStamp value indicates the arrival time of the trigger of the last acquisition that participates to the accumulation.
[7..0]	**AccData**	The number of Accumulated data is equal to the number of programmed raw data. The values are signed 16 bits values.

6.23.4 Parameter Data Frame

The Parameter Data Frame consists of a 128 bits header followed by a 256 parameters. Each parameter data is 128 bits.

Header

127..120	119..64	63..0
FrameType=x'02'	TimeStamp	Fixed to x'0123456789ABCDEF'

Parameters Data

128..0	128..0	…..	128..0
ParamData#1	ParamData#2	…	ParamData#256

[127..120]	**FrameType**	Fixed to X'02', indicates the frame is a Parameter Data frame
[119..64]	**TimeStamp**	The TimeStamp value indicate the arrival time of the trigger of the last acquisition that participate to the accumulation.
[7..0]	**ParamData**	The value of each Parameter is fixed to x'0123456789ABCDEF0123456789ABCDEF'.

6.23.5 Port Description

Signal	Size	Type	Short Description
Internal Bus			
IB_Customer **IB_Dirsel** **IB_Write** **IB_IndirCtr** **IB_Rdy** **IB_TimeO** **IB_End** **IB_Addr** **IB_DataW** **IB_DataR**	1 1 1 31 1 1 1 32 32 32	In In In In Out In Out In In Out	Must be connected to the IB-BUS signal with the same name. For details please refer to the description of the IB-BUS.
IB_Clk	1	In	Internal Bus clock, must be connected to **lbclkg**, the Local Bus clock.
Reset	1	In	Reset, must be connected to the general reset **Dreset**.
Sysclk	1	In	It must be connected to the system clock **Sysclk2**
Data Stream and Trigger			
SP_Data_A	128	In	Samples from channel A
SP_Data_Val_A	1	In	Data valid from channel A
Sp_first_A	4	In	Position of the trigger in the data block
SP_Data_B	128	In	Samples from channel B (AC/SC240 only)
SP_Data_Val_B	1	In	Data valid from channel B (AC/SC240 only)
SP_Trigger	1	In	Trigger marker
SP_Trigger_Reorder	1	In	Trigger reorder control input

Signal	Size	Type	Short Description
TimeStamp	56	In	Trigger time stamp value from the trigger core
Enable_Trigger	1	Out	Trigger Enable to the Trigger core
Transmit port			
Tx_Rem	4	out	TX Data Remainder: Indicates the number of valid bytes on given transfers.
Tx_Data	128	out	TX Data Bus. It contains the data of the frame to be transmitted.
Tx_Sof_n	1	out	TX Start of Frame: Indicates the first transfer for a given frame.
Tx_Eof_n	1	out	TX End of Frame: Indicates the last transfer for a given frame.
Tx_Src_Rdy_n	1	out	TX Source ready : Indicates that the source is ready to transfer data
Tx_Dst_Rdy_n	1	in	TX Destination ready : Indicates that the core is ready to accept data
Tx_Empty	1	in	TX Destination ready : Indicates that the core is ready to accept data
Tx_NRDY	1	in	TX Destination ready : Indicates that the core is ready to accept data
Receive port			
Rx_Rem	4	In	RX Data Remainder: Indicates the number of valid bytes on given transfers.
Rx_Data	39	In	RX Data Bus. It contains the data and the error bus of the received frame.
Rx_Sof_n	1	In	RX Start of Frame: Indicates the first transfer for a given frame.
Rx_Eof_n	1	In	RX End of Frame: Indicates the last transfer for a given frame.
Rx_Src_rdy_n	1	In	RX Source ready : Indicates that the core is ready to transfer data
Other			
Link_Err	1	Out	Data Link Error: it indicates that the data link has encountered an error condition
Link_Rdy	1	Out	Data Link Ready: it indicates that the data link is initialized and ready for the transmission or reception of data.
Led1	2	Out	To control the front panel led L1: Link 0 Status : • Red: ODL faulty. • Green: ODL successfully initialized and active in TX Mode. • Orange: ODL successfully initialized.
Led2	2	Out	To control the front panel led L2: Acquisition Status : • Orange: transfer disabled • Green: Transfer & Trigger Enabled • Red: Transfer Enabled & Trigger received

6.23.6 Registers

6.23.6.1 Main Control Register

This register defines the main control of the Base Streamer Application.

Register Space	Register Number	Register Address
Customer	**64**	**0x2300**

31..24	23..16
ACCN	-

15	14	13	12	11..10	9	8	7..6	5..4	3..0
BIDIR	CPRB	CPTB		-		STFX	-	-	-

[8] **STFX** **RW** Start/Stop framing process according to the Streamer Mode

0 Stop
1 Start

[13] **CPTB** **RW** Setting **CPTB** to '1' will start capture a new TX-Monitor buffer according to the configured Capture Mode. **CPTB** shall remain '1' until the TX-Monitor buffer is full or until the programmer wants to abort the capture by setting **CPTB** back to '0'.

[14] **CPRB** **RW** Setting **CPRB** to '1' will start capture a new RX-Monitor buffer according to the configured Capture Mode. **CPRB** shall remain '1' until the RX-Monitor buffer is full or until the programmer wants to abort the capture by setting **CPRB** back to '0'.

[15] **BIDIR** **RW** This bit shall be set '1' when the RX-Monitor buffer is used for monitoring.

[31..24] **ACCN** **RW** Number of Accumulation before to send accumulated data. Value 0 to 255 for number of accumulation 1 to 256.

To restart another capture, the start capture bits, CPTB and CPRB shall be set to '0' and then set '1' again.

6.23.6.2 TX-Monitor Buffer Control and Status

Register Space	Register Number	Register Address
Customer	**66**	**0x2308**

31	30	29..28	27	26	25	24	23	22	21..20	19..16
RDY		-	-	-	-	-	-	-	-	-

15..0
-

[31] **RDY** **R** Buffer Ready: After a capture is started, **RDY** is set '1' when the TX-Monitor buffer is ready for readout. **RDY** is set '0' when the capture bit **CPTB** of the Main Control Register is set '0'.

6.23.6.3 RX-Monitor Buffer Control and Status

Register Space	Register Number	Register Address
Customer	**67**	**0x230C**

31	30	29..28	27	26	25	24	23	22	21..20	19..16
RDY		-	-	-	-	-	-	-	-	-

15..0
-

[31] **RDY** **R** Buffer Ready: After a capture is started, **RDY** is set '1' when the RX-

Monitor buffer is ready for readout. **RDY** is set '0' when the capture bit **CPTB** of the Main Control Register is set '0'.

6.23.6.4 Base Streamer Configuration Register

Register 73 defines the length of the Stripe Frame for Stream.

Register Space	Register Number	Register Address
Customer	**73**	**0x2324**

31..16
-

15..0
SFS

[15..0] **SFS** **RW** Size of a Stripe Frame in units of 16-sample blocks. **SFS** does not take into account the size of the Header. The number of samples transmitted with a Stripe Frame is **SFS** x 16 plus the Header size. The valid range is 256 samples to 64K samples (**SFS** = 0xF to 0xFFF)

6.23.7 Resource Utilization

Resource count and relative usage in the target Xilinx Virtex II Pro – XC2VP70-6FF1517:

Resources	Used	Available	Utilization
Function Generators	1020	66176	1.5%
CLB Slices	510	33088	1.5%
Dffs or Latches	661	69068	1.0%

6.23.8 Version History

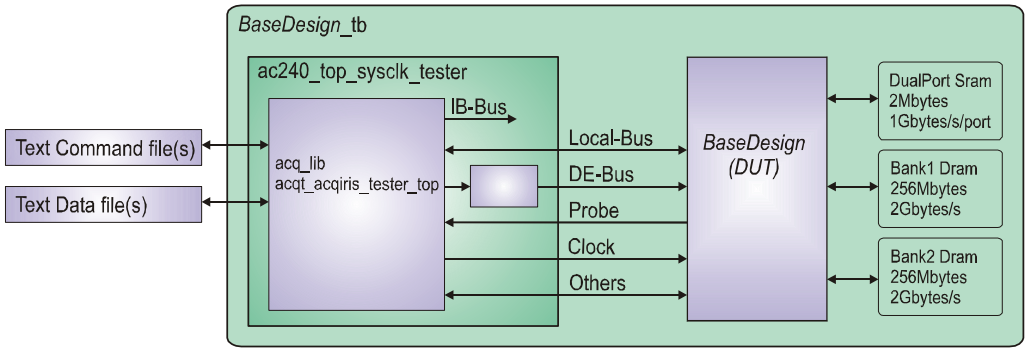
Date	FDK Version	Comments
February 06	Beta 6	New core
March 06	Beta 7	Description updated.
January 07	1.0	New SC240 Base Design

7. VHDL Test Bench

7.1 Overview

The Agilent Acqiris-supplied Test Bench is a simulation environment that simplifies the simulated functional verification of new or existing designs.

There is one test bench for each base design. The test bench component name is the name of the base design with the adjunction of **"_tb"**. The test bench usually contains a tester component, the base design itself, and the attached memories when required. There is a single tester for all ac2x0 base designs, **`ac240_top_sysclk_tester`** and one tester for the Base streamer: **`sc240_top_sysclk_str1_tester`**.



The tester has been designed with the requirement that implementing new tests should not require recompilation for simulation. It is also based on the concept of executing the self verifying tests while the simulation is running. This is achieved by reading and executing commands from a text script file, configuring the FPGA, running some simulated operations, and then reading the resulting data with the ability to compare them to a reference and to report the result of the comparison. The report feature can be configured to list only the errors, all test results or all commands.

The reference data can be either intrinsic data or data read from a text file. The test results are reported to the Modelsim transcript window with a basic "end of simulation" summary.

Developers need not understand the tester in detail, but should be aware of the existing commands to be able to write their own test bench scripts. The table below is a summary of the existing commands. They are described in detail later in this chapter.

Command	Comment
LL	Writes the text argument to the Modelsim transcript window.
SHOWAC	Enables locally to display all commands to the Modelsim transcript window.
HIDEAC	Resets the SHOWAC command.
DC	Declares and sets a value to a numeric constant.
DF	Declares and sets a value to a string constant.
EF	Executes lower level script file.
BG	Begin group
EG	End Group
RUN	Makes the simulation run when not automatically started by another command.
CWx	Writes to the FPGA (emulates driver register level write operations). Burst available.
CRx	Reads from the FPGA (emulates driver register level read operations). Burst & Test available.
IWx	Writes to the FPGA internal bus. Needs a specific test bench. Burst available.
IRx	Reads from the FPGA internal bus. Needs a specific test bench. Burst & Test available.
CKx	Defines clock frequency, start / stop clocks.
WP	Signal Probe with capability to wait for a specific pattern.
MACF	Generation of de-multiplexed data stream. Emulates the ADC and demux ASIC.

7.2 VHDL Generic of the Tester Component

Several simulation parameters can be set through VHDL generics of the tester component.

Generic	type	Short Description
DefaultScriptFile	String	Pathname of the script file that Modelsim will attempt to open at the beginning of the simulation. This could be a path relative to the Modelsim working directory.
TrigLevel	Std_logic_vector(7..0)	Trigger level
ShowAllTest	Boolean	It must be set True to list all tests to the Modelsim transcript window. False to list only failing test.
ShowAllCmd	Boolean	It must be set True to list all commands to the Modelsim transcript window.

7.3 Script Command Syntax

A script consists of a collection of files including commands. Each command contains first a command name keyword followed by a series of arguments.

The syntax of this language is divided into two grammars. The first of is the lexical grammar which define the tokens of the language. The second one is the syntaxical grammar which defines the correct sequences of tokens to write commands. The two grammars are as follows in the EBNF notation.

In the syntaxical grammar, all the italic upper-case words refer to tokens defined by the lexical grammar.

7.3.1 Lexical Grammar

```
input         =   { inputelement }
inputelement  =   whitespace
                  | token

whitespace    =   ' ' | '\t' | '\f' | '\r'
token         =   command_name
                  | number
                  | string
                  | constant
                  | switch
                  | semicolon
                  | comment
                  | eol

command_name  =   letter_up {letter_up | decdigit | '_'}
number        =   '0' 'd' decdigit {decdigit}
                  | '0' 'x' hexdigit {decdigit}
                  | '0' 'b' bindigit {bindigit}
string        =   '"' { schar } '"'
constant      =   '_' (letter | decdigit ) {letter | decdigit | '_'}
switch        =   '\' (letter | decdigit )
semicolon     =   ';'
comment       =   '-' '-' { cchar }
eol           =   '\n'


letter        =   letter_up | letter_down
```

```
letter_up     =    'A' | ... | 'Z'
letter_down   =    'a' | ... | 'z'
Decdigit      =    '0' | ... | '9'
Bindigit      =    '0' | '1'
Hexdigit      =    '0' | ... | '9' | 'a' | ... | 'f' | 'A' | ... | 'F'
Cchar         =    all characters except  '\n'
Schar         =    all characters except  '"' and '\n'
```

7.3.2 Syntaxical Grammar

```
SCRIPT        =    { LINE }
LINE          =    [ EXPRESSION ] [ comment ] eol
EXPRESSION    =    command_name {ARG} (switch ARG} semicolon
ARG           =    string
                   | number
                   | constant
```

7.3.3 Description of the two grammars

A script contains a list of commands and some comments. Comments begin with a double dash and are valid until the end of a line (as in VHDL). Except for empty or comment lines, a line contains exactly one command which consists in a command name followed by arguments. An argument is either a number, a string, a constant or a switch. A switch is special argument type composed of two parts: the first one is a backslash follows by a character while the second one is either a number, a string or a constant. Each command must be followed by a semicolon. The arguments of a command are either mandatory or optional. The optional arguments always follow the mandatory ones, but the switches, which are optional by definition, always follow the optional arguments. In short, the arguments are in the following order: mandatory parameters, optional parameters, and switches.

Only 32-bit unsigned values are supported. The following syntax must be observed within the script files:

- 0d must precede any decimal value (0d11)
- 0x must precede any hexadecimal value (0xB)
- 0b must precede any binary value (0b1011)

7.3.4 Special rules

Some rules are not expressed within the grammars. They are as follows:

- the maximum number of commands is 1000
- the maximum number of tokens is 20’000
- the maximum number of declared constants is 400
- the maximum size of a token is 120 characters
- a constant is valid from the line after its declaration to the end of the script independently of the files structure of the script. If a constant is declared in a file F2 called by the file F1, it is valid from the line after its declaration in the file F2 to the end of the file F1 including all files called from F1 after F2. In other words, the file hierarchy is first flatten before checking the scope of the constants
- a group can be opened in a file and closed in another one. As for constants, the file hierarchy is first flatten before checking for groups

If needed, these values can be changed in the file **type_def_pkg.vhd** in the ACQ_LIB library.

7.3.5 Data files

Some commands (e.g. CWF and CRFB) need a string argument which indicates a file containing data to write on the bus or data to use for comparison. Such a file must respect the following syntaxical grammar.

```
DATAFILE      =    { LINE }
LINE          =    [ number ] [ comment ] eol
```

An example of such a file is given above:

```
-- generated by the command $ perl ./WfGen.pl -b 10 -l 50 -o 512
0d512
0d576
0d639
0d700
0d758
```

7.4 Script Commands

This section details all the available commands. For each command, all the arguments are described. Note that an argument in square brackets indicates an optional argument that can be omitted.

Note that all arguments can be set explicitly or with predefined constants, using the DC or DF command, i.e. a number can be replaced by a constant declared with the DC command and a string by a constant declared with the DF command.

All the commands are blocking in the sense that the execution of the script is paused until the command is finished. They are two exceptions to this rule: D2RF and D2WF.

7.4.1 Creating Groups: BG / EG

In order to facilitate the simulation of a complex design, the concept of group of commands was introduced. It is therefore possible to run the whole script including all files or to run only some groups.

Each group is defined by a label (a string) and is delimited by the command BG (Begin Group) and EG (End Group).

Begin group.
BG NAME

End group
EG NAME
The entire text is displayed

NAME *string* Name of the group

Example:

BG “test”; Defines the group “test”
…
EG “test”;

7.4.2 Displaying Comments: LL

Displays comments in the Modelsim transcript window.
LL [TEXT]

The entire text is displayed

TEXT *string* This text is displayed

Example:

LL “TEST9 FIFO Read”;	The text “TEST9 FIFO Read” is printed to the Modelsim transcript window.

7.4.3 Report Control Command

This command is useful to modify the reporting behavior without the need of recompiling the design. It allows reducing the amount of text being printed.

The command **SHOWAC** configures the test bench to report all subsequent commands to the Modelsim transcript window, independently of the setting of the generic parameters **ShowAllCmd** or **ShowAllTest** of the tester component.

The command **HIDEAC** configures the test bench to revert to the reporting as defined by the generic **ShowAllCmd** or **ShowAllTest** of the tester component. In other words, it cancels the effect of the **SHOWAC** command.

Forces the reporting of all commands to the Modelsim transcript window.
SHOWAC
All subsequent commands of the test sequence are reported, until the **HIDEAC** command is issued.

Forces the reporting to the rules defined by the tester generic.
HIDEAC

Example:

SHOWAC;

Other commands…

HIDEAC;

7.4.4 Defining a Numeric Constant: DC

The DC command defines a numeric constant of type unsigned integer. This constant can replace any numeric parameter of the same type in any command. Remember that the name of a constant must begin with an underscore.

Defines a numeric constant of type unsigned integer		
DC	NAME	VALUE

NAME - Name of the constant.

VALUE *number* The value must be a positive integer, limited to 32 bits binary.

Example:

DC _Tvalue 0x12345678;	Assign the value 0x12345678 to the constant Tvalue.
DC _Reg0 0x2210;	Assign the value 0x2210 to the constant Reg0. Intended to be the address of a register.

CW _Reg0 _Tvalue; Write Tvalue to the register Reg0

7.4.5 Defining a String Constant: DF

The DF command defines a string constant. This constant can replace any string parameter in any command. Remember that the name of a constant must begin with an underscore. This type of constant is usually used to define paths to script files or data files.

Defines a string constant, name, and value limited to 120 characters (including quotes)		
DF	NAME	VALUE

NAME	-	Name of the constant, use only alphabetic characters.
VALUE	*string*	Any valid sequence of characters.

Example:

DF _wave1 “D:/test/wave1.txt”;	Absolute path
DF _wave2 “wave2.txt”;	Within the Modelsim directory.
DF _wave3 “../testfiles/wave3.txt”;	Relative to the Modelsim directory

7.4.6 Executing a Script: EF

The EF command enables execution of a test bench script from a script file. This enables to split a number of tests into several files, making them more readable and easier to handle. No limitation exists on the hierarchy depth, i.e. a script can call another one which can call a third one, etc.

Defines a string constant, name, and value limited to 120 characters (including quotes)	
EF	FILENAME

FILENAME	*string*	Name of the script file to be executed

Example:

DF _TESTR “D:/test/RegisterTest.txt”;	Define the script path
EF _TESTR;	Execute a script file
EF “D:/test/RegisterMem.txt”;	Execute a script file

7.4.7 Run the Simulator: RUN

The RUN command serves two purposes: It always runs the test bench for the amount of time specified by the first parameter value. It also can delay the processing of the following script commands for a certain amount of time if the second parameter value is omitted or null. Otherwise, the subsequent script commands are immediately processed by the test bench.

When a command is initiated, for example when writing a bit into a register that triggers a complete acquisition (issued from a CW command), the simulation time advances until the CW command is complete. But afterwards, the simulation does not continue unless there is another command to be executed. If you want to run for a defined time before executing another command you can use the RUN command.

Runs the simulation for a defined period of time.		
RUN	TIME	[MODE]

TIME	*number*	Time in ns.

MODE	*number*	When missing or when set to 0d0, the simulator runs for a period of time equal to TIME before reading the next script command. When set to 0d1, the simulator runs for a period of time equal to TIME, but also reads and executes immediately the next script command. When omitted, this parameter is set to 0d0

Example:

RUN 0d1000;	Run simulation for 1000 ns (1 us), then execute next script command.
RUN 0d1000 0d1;	Run simulation for 1000 ns (1 us) and simultaneously execute subsequent script commands.

7.4.8 Writing to Local Bus: CWx

This command emulates writing to the module, and more specifically to the FPGA, which is the area of interest for firmware developers. It allows single or repeated single writes as well as burst writes. This command can be used to configure and control processes within the FPGA. A CWx command produces a write access (or several write accesses) on the Local Bus.

Single or repeated write, explicit data.

CW ADDRESS WDATA [INCR] [REPEAT]

This command can be used for Direct as well as for Indirect Access.
In the case of Indirect Access, it executes a number of single word transfers.

Burst write, explicit data.

CWB ADDRESS WDATA INCR BURSTLEN

This command only allows Indirect Access.

Single or multiple write, data read from file.

CWF ADDRESS RFILE OFFSET REPEAT

This command can be used for Direct as well as for Indirect Access.
In the case of Indirect Access, it executes a number of single word transfers.

ADDRESS	*number*	Destination address
WDATA	*number*	Data to write to the destination
INCR	*number*	Data increment for repeated or burst operations (next WDATA <= WDATA + INCR). When omitted in the CW command, this parameter is set to 0d0
REPEAT	*number*	Number of times to repeat. When omitted in the CW command, this parameter is set to 0d1
BURSTLEN	*number*	Number of data in the burst
RFILE	*string*	File path for data to be written to destination
OFFSET	*number*	Number of data discarded at the beginning of the file

Example:

CW 0x2200 0x5;	Single indirect write to the processing FPGA
CW 0x2210 0xAA;	Single direct write
CWF 0x2200 "../testdata.txt" 0d2 0d5;	5 x indirect write, data from file, starting from the third data

7.4.9 Reading from Local Bus: CRx

This command emulates reading the module, and more specifically reading the FPGA, which is the area of interest for firmware developers. It allows single or repeated single reads as well as burst reads. This command can be used to configure and control processes within the FPGA. A CRx command produces a read access (or several read accesses) on the Local Bus.

Single or repeated read, explicit data.

CR ADDRESS RDATA [INCR] [REPEAT]

This command can be used for Direct as well as for Indirect Access.
In the case of Indirect Access, it executes a number of single word transfers.

Burst read, explicit data.

CRB ADDRESS RDATA INCR BURSTLEN

This command only allows Indirect Access.

Burst read, Data read from file.

CRFB ADDRESS RFILE OFFSET REPEAT

This command only allows Indirect Access.

ADDRESS	*number*	Destination address
RDATA	*number*	Reference data for comparison with read data
INCR	*number*	Data increment for repeated or burst operations (next RDATA <= RDATA + INCR) When omitted in the CR command, this parameter is set to 0d0
REPEAT	*number*	Number of times to repeat. When omitted in the CR command, this parameter is set to 0d0
BURSTLEN	*number*	Number of data in the burst
RFILE	*string*	File path to data for comparison with read data
OFFSET	*number*	Number of data discarded at the beginning of the file

Available switches:

\M MASK	*number*	Mask for the comparison. A '1' at a given position indicates that this bit position will be taken into account. When omitted, the mask is put to 0xFFFFFFFF

Example:

CR 0x2200 0x5;	Single indirect read to the processing FPGA
CR 0x2210 0xAA \M 0xFF;	Single direct read. Only the 8 LSB are taken into account for the comparison to the 0xAA value
CRFB 0x2200 "../testdata.txt" 0d2 0d5;	5 x indirect read, reference data from file, starting from the third data

7.4.10 Writing to Internal Bus: IWx

This command partially emulates writing to the internal bus port (IB-BUS) of the tester component **`acqt_acqiris_tester_top`** of the library **`acq_lib`**. This port is a 'simulation' port and is not identical to the IB-BUS inside the FPGA. The command **IWx** therefore cannot be used to communicate

directly with the internal bus within the FPGA. The IB-BUS port of the tester component is usually not forwarded to the test bench component of the Base Designs.

Developers can use this command to implement test benches for sub-parts which communicate with the internal bus port as defined by Agilent. By connecting such sub-parts directly to the IB-BUS of the tester component, the test bench complexity for non-system test benches is reduced.

These "Ixxx" commands have a corresponding "Cxxx" command. This simplifies the conversion of internal bus test benches to system test benches, by simply replacing the "I" with a "C".

Only a single Internal Bus can be emulated with tester component.

Single or repeated write, explicit data.

IW ADDRESS WDATA [INCR] [REPEAT]

This command can be used for Direct as well as for Indirect Access.
In case of Indirect Access, it executes a number of single word transfers.

The address is any valid FPGA address except the Indirect Data Register.

Burst write, explicit data.

IWB ADDRESS WDATA INCR BURSTLEN

This command should be used only in the case of Indirect Access.
The address is the Indirect Data Register.

Single or multiple write, data read from file.

IWF ADDRESS WFILE OFFSET REPEAT

This command can be used for Direct as well as for Indirect Access.
In the case of Indirect Access, it executes a number of single word transfers.

ADDRESS	*number*	Is the full address as if the register was addressed through the Local Bus interface. The upper bits are not used. This facilitates the translation of the internal bus command to a Local Bus command or vice-versa. **Command IW**: any valid FPGA address except the Indirect Data Register. **Command IWB**: The address is the Indirect Data Register, 0x2200 for the processing FPGA and 0x2600 for the communication FPGA.
WDATA	*number*	Data to write to the destination
INCR	*number*	Data increment for repeated or burst operations (next WDATA <= WDATA + INCR)
REPEAT	*number*	Number of times to repeat
BURSTLEN	*number*	Number of data in the burst
WFILE	*string*	File path for data to be written to destination
OFFSET	*number*	Number of data discarded at the beginning of the file

Example:

IWB 0x2200 0x5 0x1 0d1;	Indirect write, this case is a burst of 1 data.
IW 0x2210 0xAA;	Single direct write.

7.4.11 Reading From Internal Bus: IRx

This command partially emulates reading from the internal bus port (IB-BUS) of the tester component **`acqt_acqiris_tester_top`** of the library **`acq_lib`**. Please refer to the comments of the previous section for more explanations.

Single or repeated read, explicit data.

IR ADDRESS RDATA [INCR] [REPEAT]

This command can be used for Direct as well as for Indirect Access.
In case of Indirect Access, it executes a number of single word transfers.

The address is any valid FPGA address except the Indirect Data Register

Burst read, explicit data.

IRB ADDRESS RDATA INCR BURSTLEN

This command should be used only in the case of Indirect Access.

The address is the Indirect Data Register.

Burst read, Data read from file.

IRFB ADDRESS RFILE OFFSET REPEAT

This command only allows Indirect Access.

ADDRESS	*number*	Is the full address as if the register were addressed through the Local Bus interface. The upper bits are not used. This facilitates the translation of the internal bus command to a Local Bus command or vice-versa.. **Command IR**: any valid FPGA address except the Indirect Data Register. **Command IRB**: The address is the Indirect Data Register, 0x2200.
RDATA	*number*	Reference data for comparison with read data
INCR	*number*	Data increment for repeated or burst operations (next RDATA <= RDATA + INCR)
REPEAT	*number*	Number of times to repeats
BURSTLEN	*number*	Number of data in the burst
RFILE	*string*	File path to data for comparison with read data
OFFSET	*number*	Number of data discarded at the beginning of the file

Available switches:

\M MASK	*number*	Mask for the comparison. A '1' at a given position indicates that this bit position will be taken into account. When omitted, the mask is put to 0xFFFFFFFF

Example:

IRB 0x2200 0x5 0x1 0d1;	Indirect read, this case is a burst of 1 data word
IR 0x2210 0xAA \M 0xFF;	Single direct read. Only the 8 LSB are taken into account for the comparison to the 0xAA value

7.4.12 Clock Generation: CKx

Up to 8 different clocks can be defined for test benches. The clocks are generated on the outputs **`ExtClk(7:0)`** of the tester component: **`acqt_acqiris_tester_top`** of the library **`acq_lib`**. Each one can be enabled or disabled separately. These clock signals are usually passed to the test bench component of the Base Designs.

Setting clock period		
CKSET	CKNUMBER	CKPERIOD

Enabling clocks	
CKEN	ENMASK

CKNUMBER	*number*	A number from 0 to 7 referencing the clock signal **`ExtClk.`**
CKPERIOD	*number*	Clock period in picoseconds, must be a positive integer.
ENMASK	*number*	An 8-bit pattern for enabling the clocks. If the bit i (with i = 0 to 7) of ENMASK is set to '1', the clock signal **`ExtClk(i)`** is enabled. If the bit i is set to '0', the corresponding clock signal is disabled.

Examples:

CKSET 0d0 0d1000;	Period of clk0 is 1 ns.
CKSET 0d2 0d8000;	Period of clk2 is 8 ns.
CKEN 0b101;	The two clocks ExtClk(0) and ExtClk(2) are enabled. All other clocks are disabled.

⚠ In the previous version of this command, the base unit was a nanosecond, not a picosecond. All the custom scripts using the previous "ns" units must therefore be modified to the new "ps" units.

7.4.13 Probe Interface: WP

Up to 8 different probes can be used for test benches. The signals to probe must be connected to the inputs **`ExtProbe(7:0)`** of the tester component **`acqt_acqiris_tester_top`** of the library **`acq_lib`**. Each probe can be masked individually. The probe signals are usually connected to the test bench component of the Base Designs.

The implemented function simply waits until the defined masked pattern exists on the signal **`ExtProbe`**. This is useful to wait for some process to end, before continuing the simulation (example: wait for an interrupt signal).

Wait for probe pattern		
WP	MASK	PATTERN

MASK	*number*	An 8-bit mask where the bit i, in the range 0 to 7 will mask the value of the signal ExtProbe(i), setting the probe value to be '0' if the bit is set '0' or to the value of the connected signal if the bit is set '1'.

PATTERN *number* An 8-bit pattern

Example:

WP 0b1011 0b11; Wait until signal 3=0 AND signal 1 = 1 AND signal 0 = 1

7.4.14 Data Stream Generation: MACF

This command generates a digital data stream intended to emulate the acquisition and demultiplexer function that supplies ADC data to the Data Processing Unit.

The data are read from a file and demultiplexed to the signal **DE_DATA** which is an output from the component **acqt_acqiris_tester_top** of the library **acq_lib**. The command also generates the clock **DECLK** that must be used to store the data. The output signal **DE_DATA** is 16 samples wide. The negative edge of **DECLK** must be used for clocking it into subsequent registers.

This is a single channel command. For multiple channel versions there some additional circuitry is implemented within the tester component of the Base Designs. For dual channel versions, we usually use the MACF command at twice the effective sampling rate, the data being de-multiplexed by 2 and the clock period multiplied by two.

The data stream can start in two different ways, either immediately after the command is executed or triggered by the input signal **TRACPT** of the component **acqt_acqiris_tester_top**. Please read the description of the base design tester component to see if it is available and how it has been connected.

Generate demultiplexed data stream				
MACF	FILE	PERIOD	REPEAT	[STARTMODE]

FILE Filename of source data

PERIOD **declk** period in ns

REPEAT Number of 16-sample blocks to send on the DE bus

STARTMODE 0d0 or omitted to start the data stream immediately
0d1 to start the data stream at the next rising edge of TRACPT

Example:

MACF ../src/aaa_debug 0d8 d10; Generate immediately 10 blocks of 16 data values from file, **declk** period is 8 ns.

7.5 Version History

Date	FDK Version	Comments
September 05	Beta 4	Added Internal Bus command
May06	Beta 7	Add new command: SHOWAC, HIDEAC. Overall Description updated.
Dec 06	1.0	New syntax
January 07	1.0	New SC240 Base Design

8. Design Flow

The purpose of this chapter is to describe the supported design flows and tools. You will find a description on how the tools shall be configured, on what is specific to Agilent and on where to find the information developers will need.

There are multiple design flows that are different from one to the next because the design entry tool and/or the synthesizer are different. The design entry tool can be either HdlDesigner from Mentor Graphics or a simple text editor while the synthesizer can be either Precision Synthesis from Mentor Graphics or XST from Xilinx.

The VHDL simulator is Modelsim, there is no other choice. The "Place and Route" tool is ISE from Xilinx.

It is responsibility of the developer to attend to specific course in order to get used to VHDL design and tools usage.

The design flow can be :

Flow	Design phase	Tool for Design Entry	Tool for Simulation and Synthesis	TOOL FOR P&R
With HdlDesigner	Design & Verification	HdlDesigner	Modelsim	
	Implementation		Precision Synthesis or XST (ISE)	ISE
Without HdlDesigner	Design &Verification	Text Editor / other	Modelsim	
	Implementation		Precision Synthesis or XST (ISE)	ISE

The design entry tool for the flows without HdlDesigner can be either a simple text editor or another higher level design entry tool.

The installer program will copy all FDK design files to the directory at the location defined by the environment variable AcqirisFdkRoot. This directory is referenced hereafter as the **FDKdirectory** whereas **$AcqirisFdkRoot** is used in path names and **%AcqirisFdkRoot%** is used within the value of an environment variable.

The FDK directory structure is described hereafter in the paragraph about the HdlDesigner flow while the key files are listed in the next chapter in the description of the developer library.

8.1 Standard Tools

Developing a new firmware will be faster in the early stage for developers using HdlDesigner. This is why we strongly recommend that our customers buy it. Customers will have the advantage of a fully integrated tool. Agilent Acqiris Technical support will be able to react much faster to any inquiry.

Tool Name	Actual Version	Features / Comment	Approx. Cost in US$
HDL Designer Author graphics Mentor Graphics	2004.1b	Graphical entry: • State diagram. • Block diagram. • Integrated flow for synthesis and simulation. • Facilitate work for documentation. • Good overview of the whole design. • 2nd party tool integration for Version control.	12.500
Modelsim PE / SE Mentor Graphics	6.1b /d	Best simulation tool on the market.	5600
Precision Synthesis Mentor Graphics	2005c.79	New generation of mentor synthesis technology.	15000
ISE Foundation	8.1.03i	Xilinx Synthesis Floorplanning and Place & Route.	3000

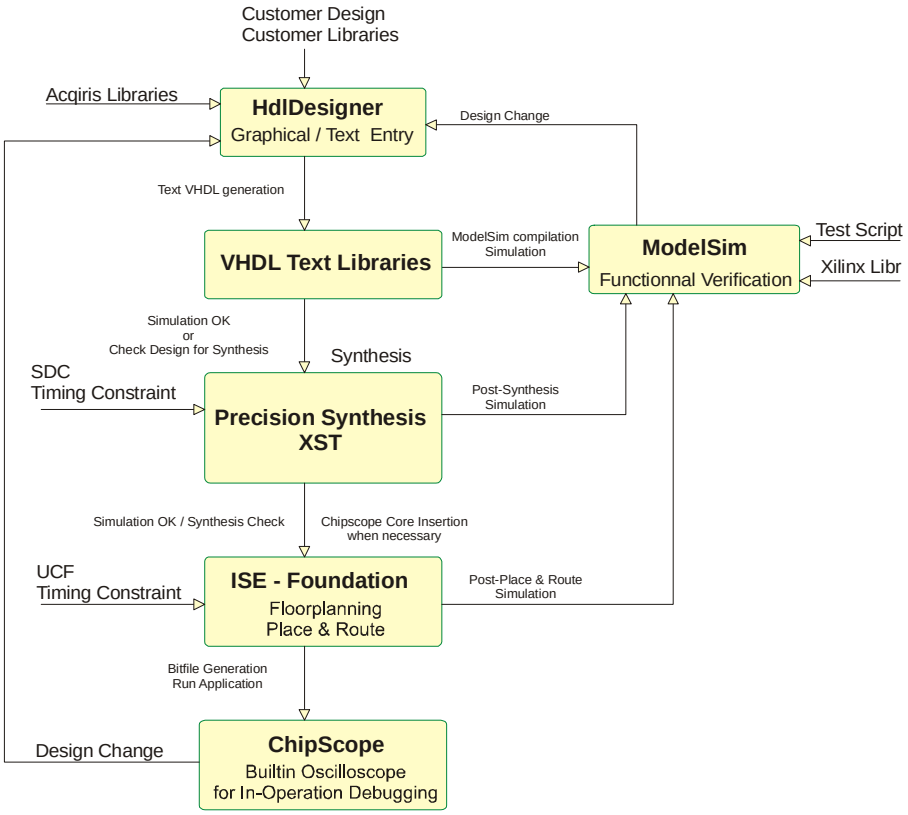
Edif or VHDL flow Xilinx			

8.2 Design Flow with HdlDesigner

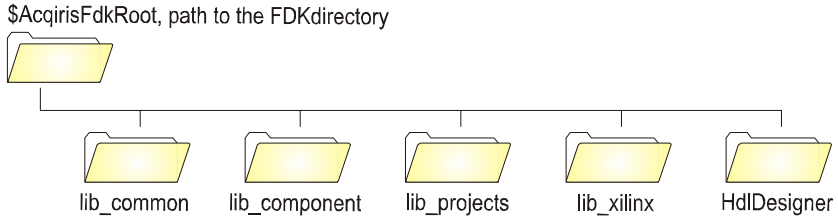
HdlDesigner is our main tool for design management and to run synthesis or simulation. HdlDesigner is a complex program with the capability of doing almost anything for FPGA firmware designers. We make minor changes to the default mentor graphic HdlDesigner configuration. These changes can be set by firmware developers by loading the default Acqiris Team and User configuration.

The directory structure is strongly oriented by the usage of HdlDesigner. It is based on a set of directories containing design libraries. At the upper level, all supported design flows share the same directory structure. So it is also useful for non HdlDesigner users to read this paragraph entirely.

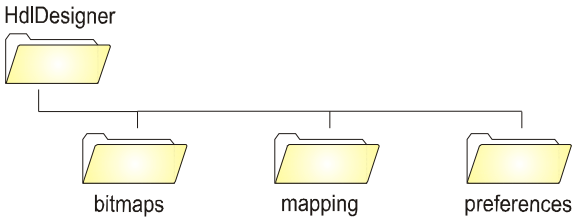
8.2.1 Block Diagram



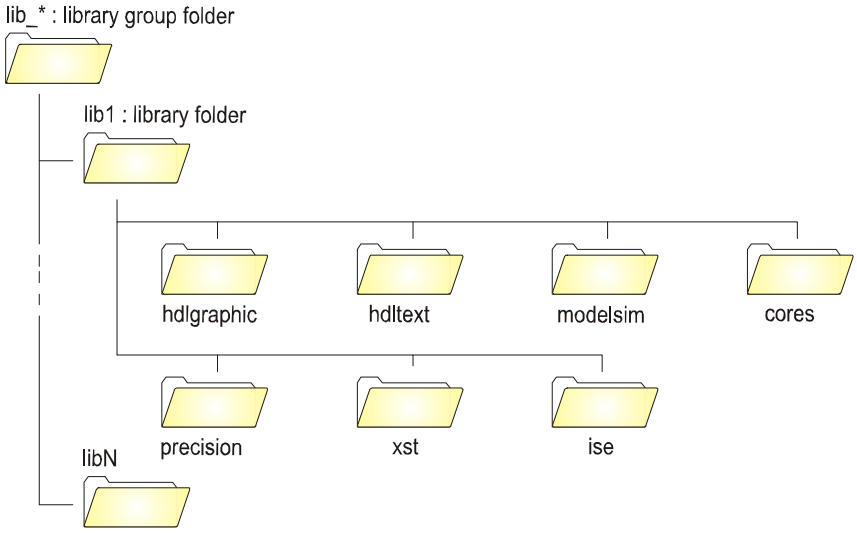
8.2.2 Directory Structure of the FDK Installation



$AcqirisFdkRoot /*	**Purpose**
lib_common	Library group folder for Acqiris standard libraries.
lib_component	Library group folder for third party libraries.
lib_projects	Library group folder for Acqiris projects and base design(s).
lib_xilinx	Library group folder for Xilinx libraries: unisim, simprim, xilinxcorelib
HdlDesigner	Directory for HdlDesigner configuration and preferences



HdlDesigner/*	**Purpose**
bitmaps	Several icons bitmap for the HdlDesigner gui
mapping	HdlDesigner Project file and shared project file for library mapping
preferences	User and Team preferences for HdlDesigner



Directories of the library folder	**Purpose**
hdlgraphic	HdlDesigner Graphical Design files. Non HdlDesigner users should not take care of these files. HdlDesigner will generate a vhdl text version of these

	designs in the hdltext folder.
hdltext	The vhdl source files for simulation and synthesis.
modelsim	ModelSim compiled files & working directory. It includes the file modelsim.ini that should be used to define the libraries and preferences for modelsim.
cores	ISE source folder for sub-components with an edif or ngc description.
precision*	Working directory for Precision Synthesis
ise*	ISE implementation directory for the ISE EDIF flow
xst*	ISE implementation folder for the XST design flow

* These directories are present only for the library ac240_developer_lib.

8.2.2.1 HdlDesigner SideData

Side Data are supplementary source design data (such as EDIF, SDF, and document header files) or user data (such as design documents or text files) which are saved with a design unit view and can be viewed using the Side Data browser. (design unit view: state machine, Block diagram, VHDL text file,…).

8.2.2.2 HdlDesigner SideData Directory

There are two Side Data directories: The design data Side Data directory and the user data Side Data directory. We only use the design data directory. Starting HdlDesigner 2004, the pathname of this directory depends on the type of design unit view it is associated with.

Design unit type	design data SideData Directory
Block diagram	hdlgraphic/*design_unit_name*/struct.bd.info
State machine	hdlgraphic/*design_unit_name*/fsm.sm.info
Text VHDL	hdlgraphic/.hdlsidedata/*design_unit_name*.vhd.info

There is Side Data for each component with a non-VHDL description. For example when there is an EDIF or NGC description, which is the case for cores or components created with the Xilinx core generator. The necessary EDIF or NGC view will be copied to the directory `cores` of the developer library.

There is Side Data for each top level base designs. It includes script files and configuration files for downstream tools.

8.2.3 Configuring HdlDesigner

This paragraph describes how developers should configure HdlDesigner. All necessary files are copied to the directory **$AcqirisFdkRoot\HdlDesigner**.

8.2.3.1 Acqiris Team and User Preferences

The User and Team preferences as defined by Agilent Acqiris must be used in order to compile successfully the designs. This can be simply done by setting two environment variables:

HDS_USER_HOME to %AcqirisFdkRoot%\HdlDesigner\Preferences\hds_user

HDS_TEAM_HOME to %AcqirisFdkRoot%\HdlDesigner\Preferences\hds_team

The Team and User preferences are set to recognize the programs listed hereafter. These programs run automatically when necessary from within HdlDesigner. This is true only if the path for the executable is added to the window environment variable PATH.

Microsoft WORD, EXCEL, and POWERPOINT as well as ACROBAT, CORELDRAW, and ULTRA-EDIT.

8.2.3.2 Project File / Library Mapping

HdlDesigner has the concept of Project File and Shared Project File. The Project File defines all project specific libraries while the Shared Project File defines libraries that are shared by multiple projects. A good example of a specific library is the library ac240_fdk while the best example of a shared library is the test bench library acq_lib.

The library mapping is included in the project files. The mapping is a list of paths indicating to Hdl Designer where the files associated with a library are. This includes paths for design files and paths for the downstream tools working directories.

As the path for the Shared Project File is defined within the Project File, we only need to load the Project File to get the settings for all Design libraries.

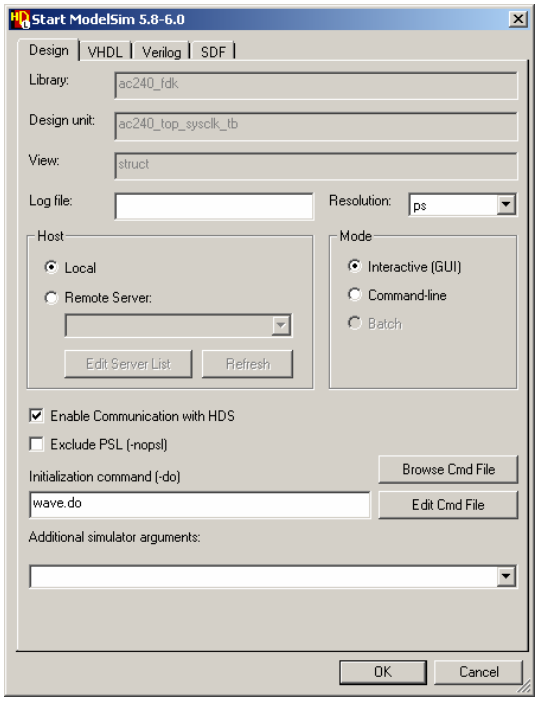
Library Mapping	File Location
Shared Project File	$AcqirisFdkRoot/HdlDesigner/Mapping/***FDK_Name***_shared.hdp
Project File	$AcqirisFdkRoot/HdlDesigner/Mapping/***FDK_Name***.hdp

FDK_Name stands for the main fdk library. For the ac240 FDK, this will be "ac240_fdk".

8.2.4 Simulation with Modelsim

The FDK installation includes the Agilent Acqiris libraries already compiled for the currently supported version of Modelsim. As Agilent does not issue a CD for each new Modelsim version, the customer shall recompile the libraries when using a newer Modelsim version.

Modelsim can be started from the HdlDesigner gui. The simulation resolution shall be set to 1ps.

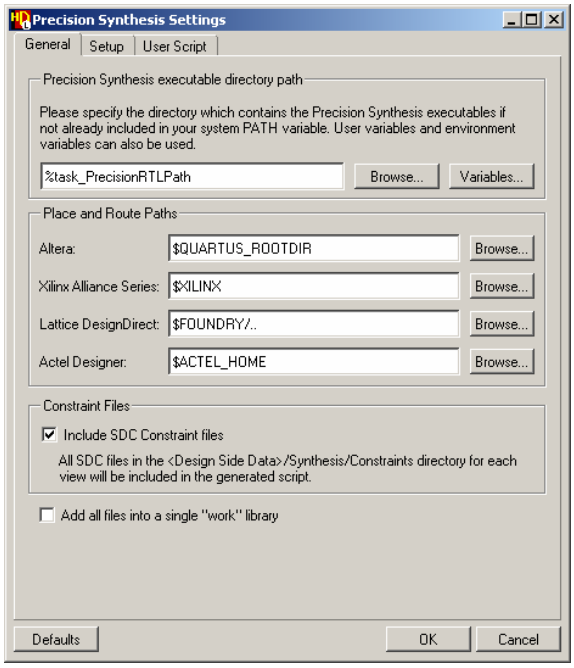


Once the design is loaded, you shall start the simulator by running it for the desired time: "run 100 us". You might then follow the queries of the Acqiris Test Bench. The test results are displayed in the transcript window. Please refer to the description of the Acqiris Test Bench for more information.

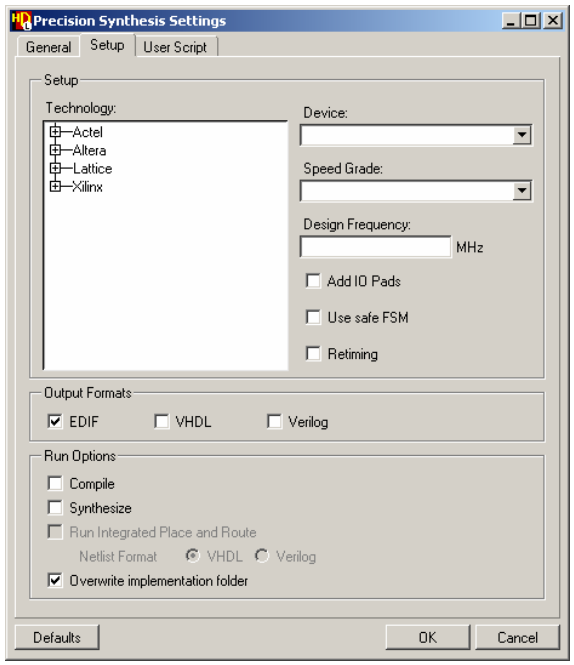
8.2.5 Synthesis with Precision Synthesis

The FDK installation has the Base Design already synthesized for the currently supported version of Precision Synthesis. Precision Synthesis should be started from HdlDesigner as parameterized below.

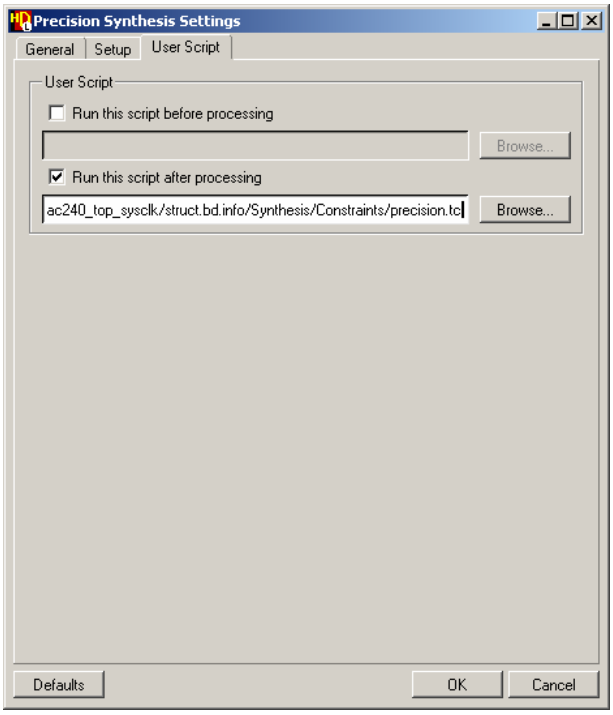
In the General tab of the Precision Synthesis Settings interface, you must activate the option "Include SDC Constraint files". The synthesis constraint file in the Design Side Data will be used for synthesis.



In the Setup tab of the Precision Synthesis Settings interface, you should activate the option “Overwrite implementation folder”. We select this option in order to have the edif and ucf files created by Precision Synthesis always at the same location. This is convenient when ISE is configured to point to these two files.



In the User Script tab of the Precision Synthesis Settings interface, you must activate the option “Run this script after processing” and you must select the correct script file. The script file depends on which Base Design will be synthesized. It is located in the Design Side Data directory of the Base Design, in the folder “/Synthesis/Constraints”.



The synthesis will automatically run and save the EDIF and UCF file for ISE. While the EDIF file is always the design entry for ISE, the ucf file to use could be different than the ucf file generated by Precision Synthesis. Details are indicated in the chapter VHDL libraries.

8.2.6 Synthesis with XST (ISE)

XST could be launch directly from HdlDesigner. This flow is not supplied with the FDK installation.

8.2.7 Implementation with ISE

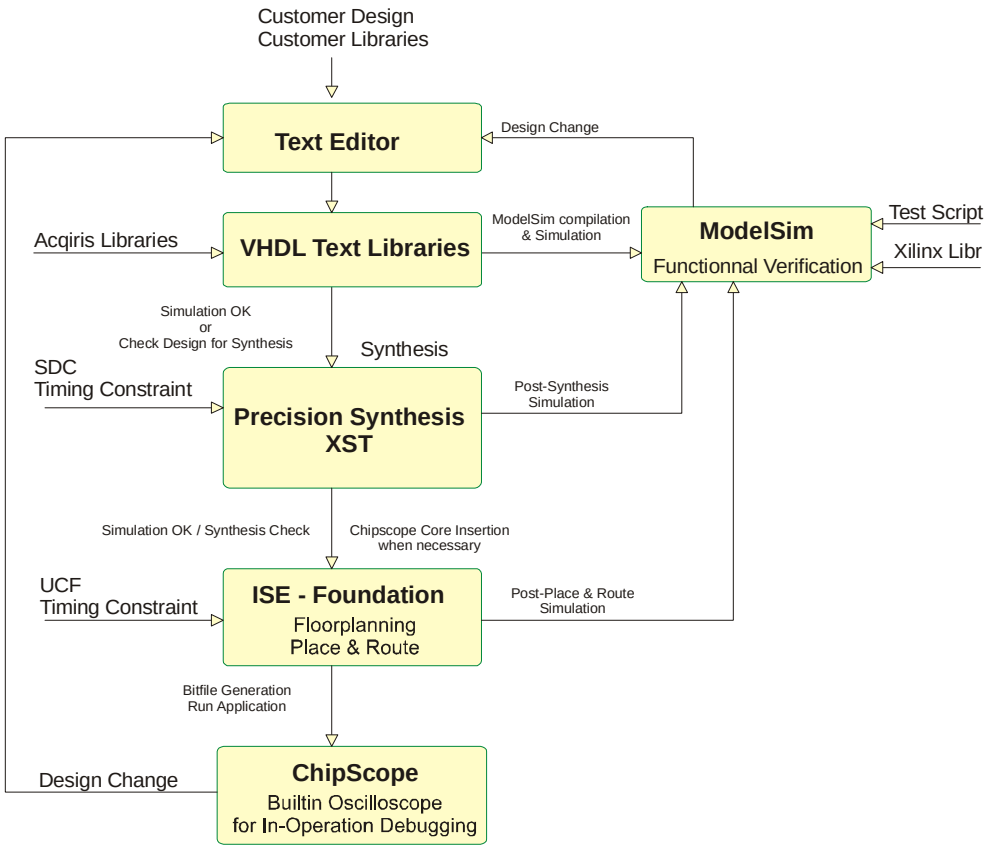
The ISE-EDIF implementation shall be used if the design was synthesized with Precision Synthesis. This is described hereafter in the paragraph specific to the design implementation with ISE.

8.3 Design Flow without HdlDesigner

The directory structure is designed for the use of HdlDesigner. It is recommended that non HdlDesigner users read section 8.2 ***DESIGN FLOW WITH HDLDESIGNER*** prior to read this one.

8.3.1 Block Diagram

The more simple design flow will uses a text editor, Modelsim, and ISE Foundation with the XST synthesizer. Synthesis with Precision Synthesis is also possible.



8.3.2 Simulation with Modelsim

The file modelsim/modelsim.ini should be used to configure Modelsim. This includes the preferences and the library definitions.

8.3.3 Synthesis with Precision Synthesis

Developers should use the two files below to configure Precision Synthesis:

$AcqirisFdkRoot/lib_projects/***Developer_Library***/precision/***Base_Design***_struct/hds/add_files.tcl

$AcqirisFdkRoot/lib_projects/***Developer_Library***/precision/***Base_Design***_struct/hds/precision.tcl

Developer_Library stands for the delivered developer library. In case of the ac240 FDK this is the library **`ac240_developer_lib`**.

Base_Design stands for the delivered Base Designs. In case of the ac240 FDK this can be either **ac240**, **ac210** or **ac240_ddr**.

The script add_files.tcl is executed from the script precision.tcl. These two scripts include references to design files using absolute paths. You should modify them prior of use. The absolute path E:/FPGA shall be replaced by the value you set for the variable $AcqirisFdkRoot.

8.4 Design Implementation with ISE

We distinguish two ISE implementation paths: the **EDIF path** for which the synthesis is done with an external synthesizer like Precision Synthesis and the **VHDL path** for which the synthesis is done with XST, the Xilinx synthesizer.

As the input files for the **EDIF path** and the **VHDL path** are different, we created two separate ISE projects. The ISE project files for the **EDIF path** are stored in the directory **ise** while the ISE project files for the **VHDL path** are stored in the directory **`xst`**. We created one ISE project for each Base Design, this leads to 6 ISE projects (3 base designs x 2 paths). Using these project files ensures that you get all the correct settings.

The configuration for the ISE **Translate**, **Map**, **Place-and-Route**, and **Generate-Programming-File** processes are identical for both the EDIF and the VHDL flows.

8.4.1 Cores Directory

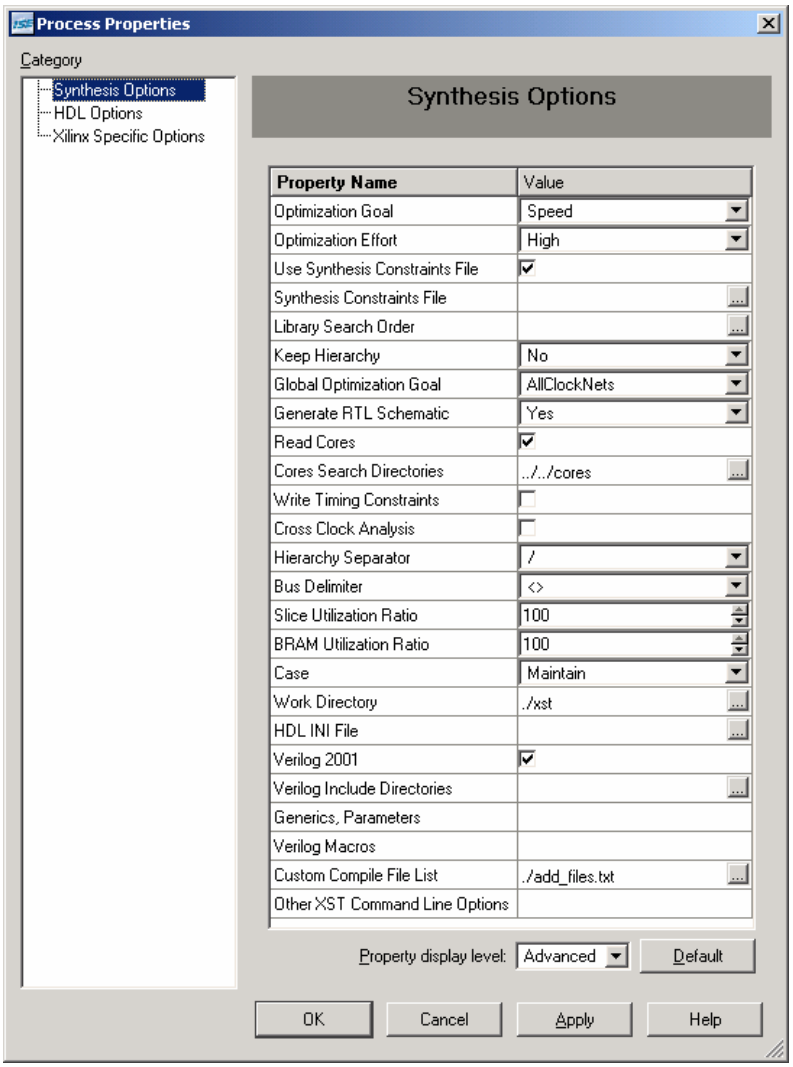
There is Side Data for each component with a non-VHDL description like the EDIF and/or the NGC format. This is the case for cores or components generated with the Xilinx core generator. These EDIF and/or NGC files are copied to a common directory, the directory **`cores`** of the developer library. The delivered ISE projects are configured to search for files in this directory.

8.4.2 Synthesis with XST (ISE)

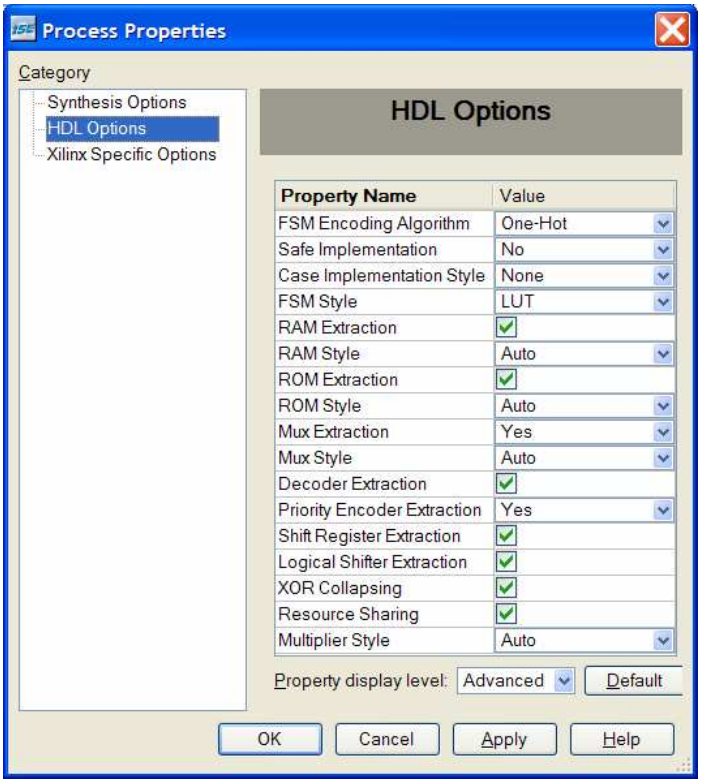
The ISE project file points to the vhdl files of the fdk libraries (vhdl files located in the directories hdltext of the fdk libraries). Any change in these files is detected, requiring the flow to be re-run.

8.4.3 Property Settings for ISE-XST

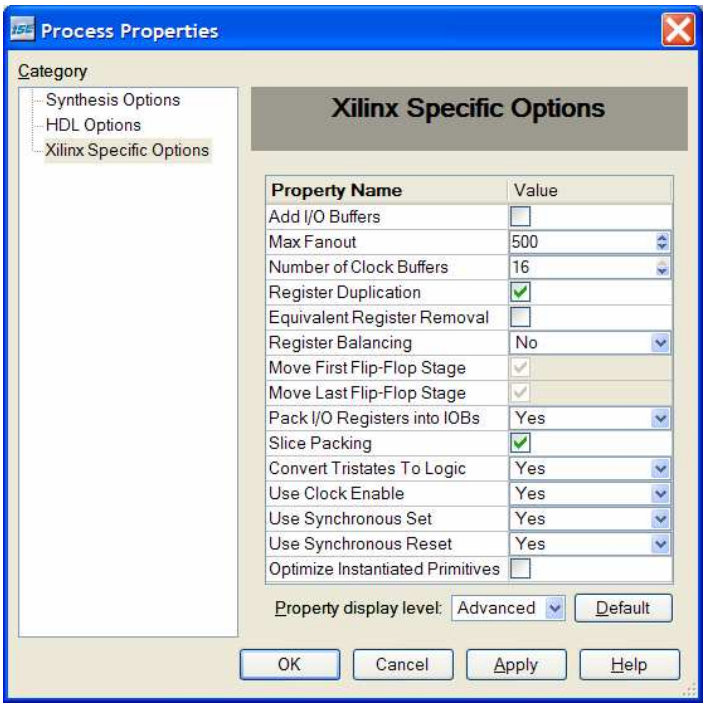
The Optimization Goal property must be set to speed and the Optimization Effort property to High. The Cores Search Directories property must be set to the **`cores`** directory. The custom compile file list add_files.txt must be used.



The FSM Encoding Algorithm property must be set to One-Hot.

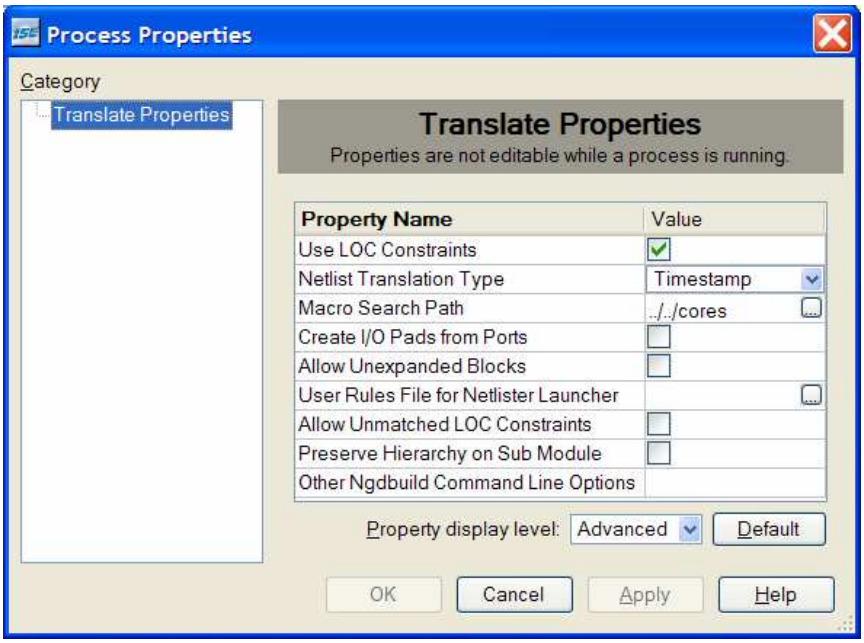


The property Add I/O Buffers must be unset. The Max Fanout property must be set to 500.



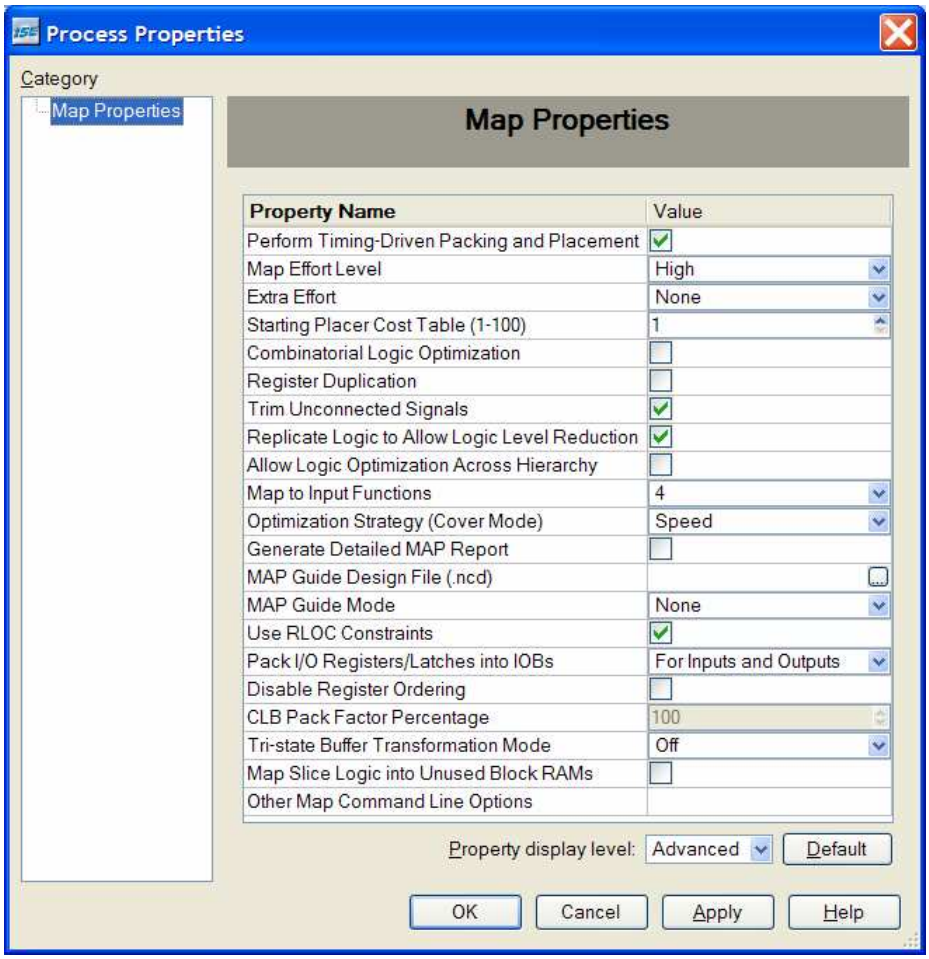
8.4.4 Properties Settings for ISE-Translate

The Macro Search Path property must be set to the **`cores`** directory.

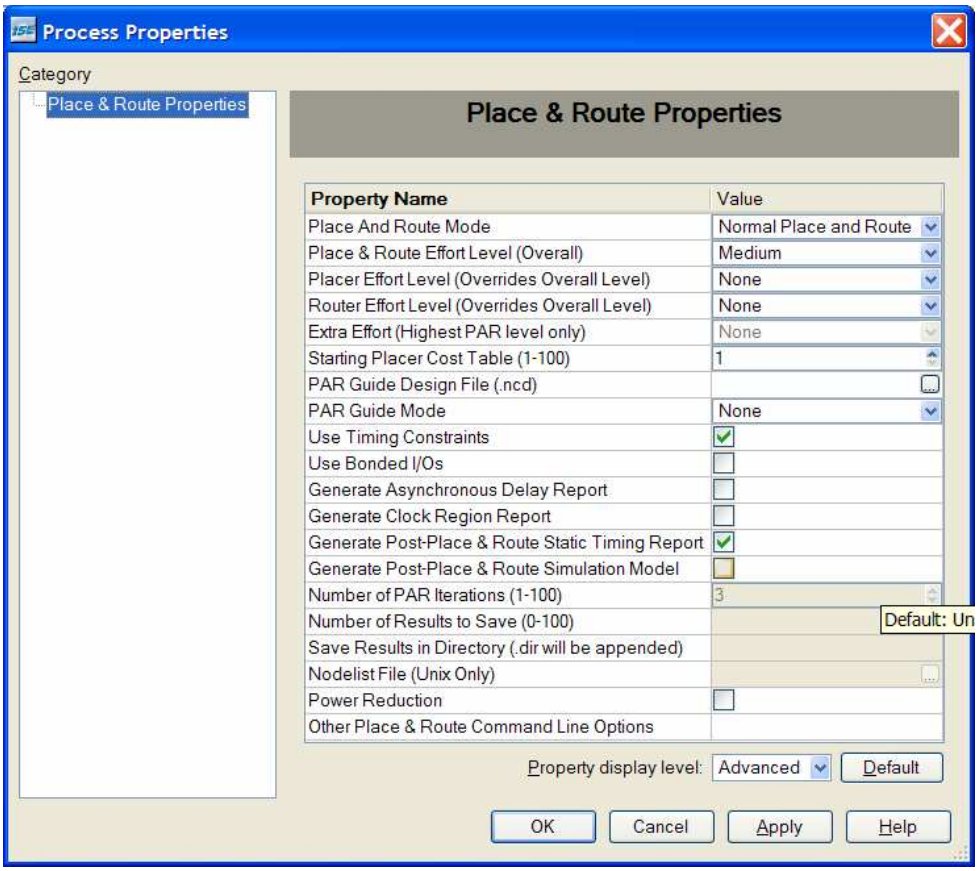


8.4.5 Properties Settings for ISE-Map

Mapping the design must be run with the Timing-Driven property set and with an effort level set to high. The Optimization Strategy property must be set for speed.

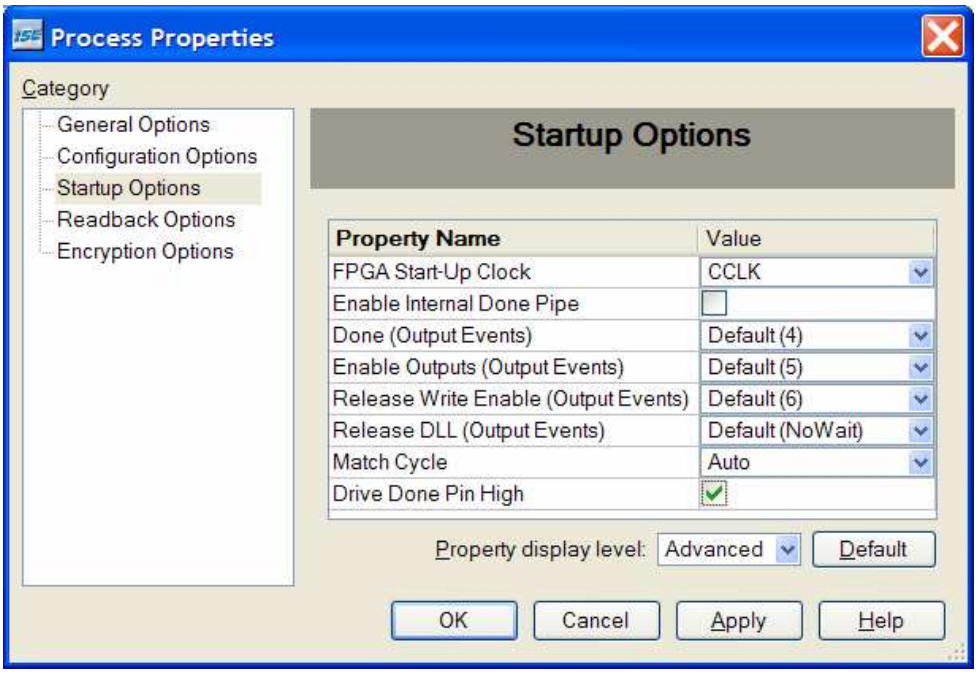


8.4.6 Properties Settings for ISE-Place and Route



8.4.7 Properties Settings for ISE-Bit file Generation

We use all default properties except for the “Drive Done Pin High“ property which must be set.

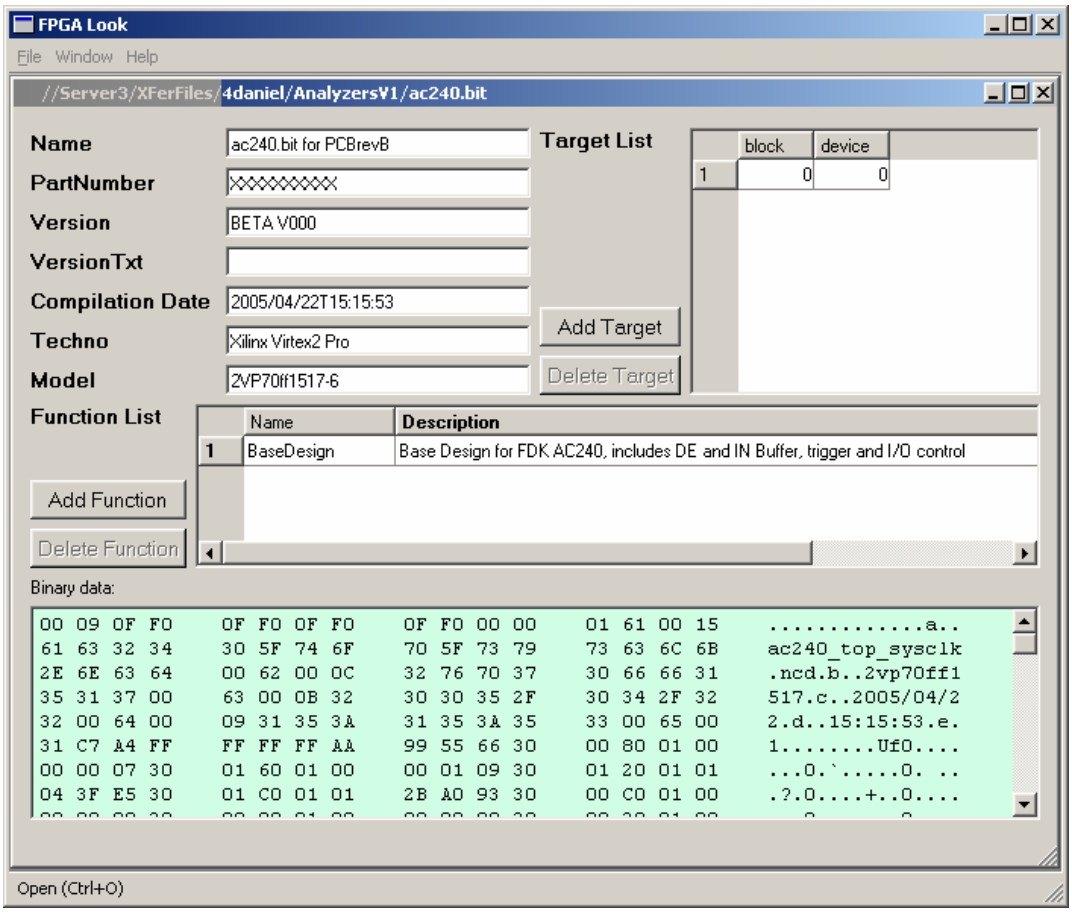


8.5 ChipScope

The module could be connected to Xilinx ChipScope using the Xilinx **Platform Cable USB** and an adapter delivered by Agilent.

8.6 FPGALook

FPGALook is a program used to add a header at the beginning of a bit file. This header has predefined fields that can be modified with FPGALook. These fields can be read with the Acqiris Driver function **Acqrs_getInstrument Info**.



The following fields must be filled as indicated below:

Compilation Date: This field is automatically inserted and read from other portions of the bit file.

Techno: Name of the target technology

Model: reference of the target

Target List: block shall be 0, device shall be 0.

Contents of the others fields are up to the Firmware Developer.

8.7 Version History

Date	FDK Version	Comments
September 05	Beta 4	Modify path for modelsim compiled files of Xilinx libraries, uses the $XILINX environment variable and the default path defined by ise.
May06	Beta7	Reviewed the entire chapter, add information for the XST flow.
January 07	1.0	New SC240 Base Design

9. VHDL libraries

The FDK uses components from Acqiris or Xilinx standard libraries

The purpose here is to briefly describe the libraries and their key components.

9.1 Delivered Libraries

The FDK installer will install the VHDL libraries listed in the table below.

Library Name	Purpose
AcqirisFdkRoot/lib_projects/…	
ac240_developer_lib	Library for developing new firmware for the AC2x0 / SC2x0 products
ac240_fdk	Acqiris library: components specific to the FDK of the AC2x0 / SC2x0 products. This library shall not be modified
cpld_sc240	Acqiris library: On board Local Bus control and digitizer control
ddr_ctrl_virtex2	Acqiris library: DDR interface core
AcqirisFdkRoot/lib_common/…	
std_lib	Acqiris library: common components for basic functions
std_xilinx	Acqiris library: common Xilinx components
acq_lib	Acqiris library: common test bench components
fdk_lib	Acqiris library: common FDK components
fdk_lib_h	Acqiris library: common FDK cores
AcqirisFdkRoot/lib_component/…	
cypress	Acqiris library: SRAM simulation model based on the cypress model
samsung_ddr	Acqiris library: DRAM simulation model based on the Samsung model

9.2 Xilinx Libraries

The Xilinx libraries are not installed by the install program because of their size (>~100 MB). Developers have to generate them with ISE. The installation procedure is described hereafter.

The HdlDesigner mapping should be modified if the simulator is the modelsim SE version.

9.2.1 HdlDesigner Library Mapping

The HdlDesigner mapping uses the default path defined by ISE for modelsim PE and modelsim SE. In case developers use modelsimSE, the HdlDesigner mapping should be modified to point to the file for modelsim se.

Libraries	Purpose	Location	ModelSim
Xilinx Compiled libraries			
unisim	Xilinx library for functional simulation	$Xilinx/vhdl/mti_pe/unisim	PE
		$Xilinx/vhdl/mti_se/unisim	SE
XilinxCoreLib	Xilinx Cores	$Xilinx/vhdl/mti_pe/xilinxcorelib	PE
		$Xilinx/vhdl/mti_se/xilinxcorelib	SE
simprim	Xilinx primitives for delay annotated simulation.	$Xilinx/vhdl/mti_pe/simprim	PE
		$Xilinx/vhdl/mti_se/simprim	SE

9.2.2 Installing the Xilinx Libraries

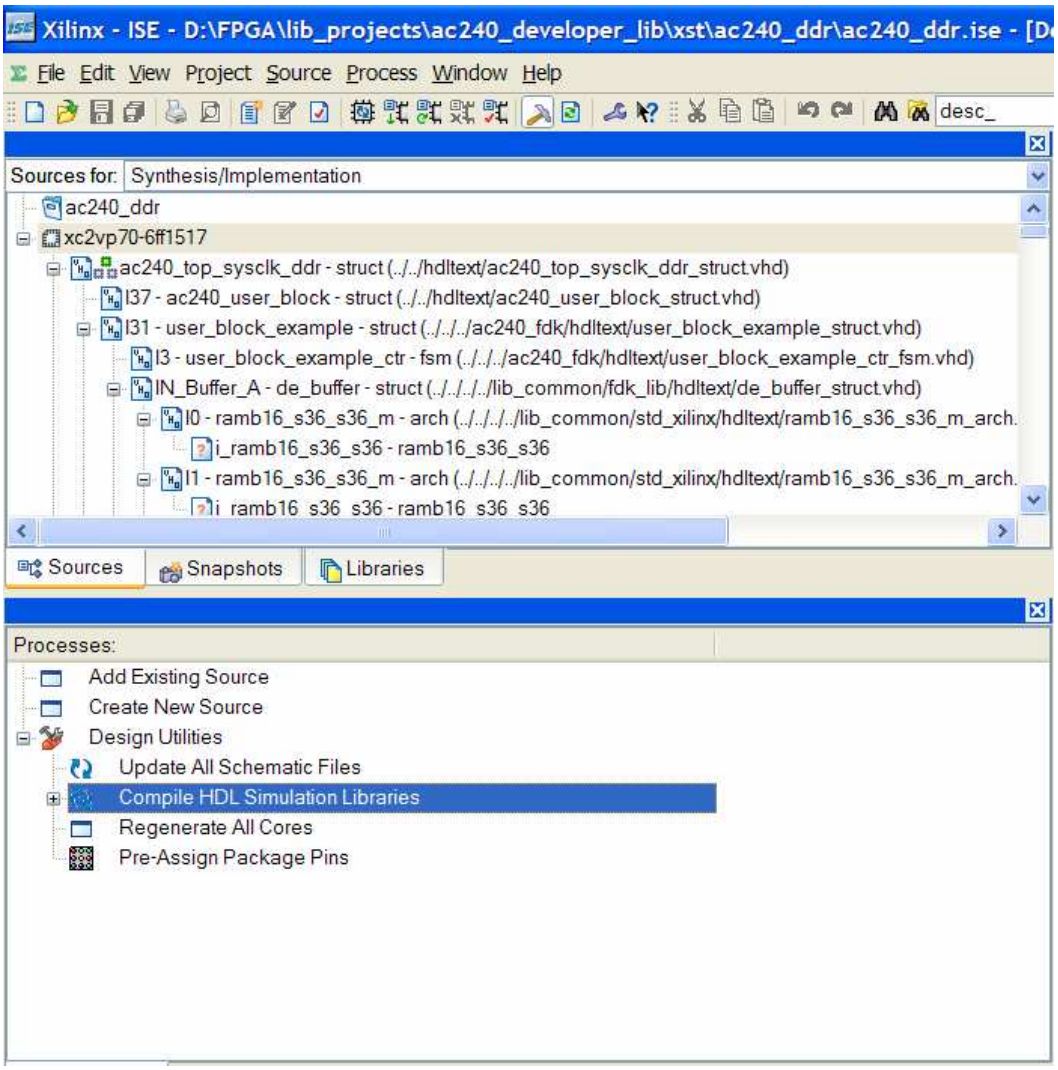
You shall install the Xilinx " ip update" on top of the standard Xilinx installation in accordance to the supported Xilinx version.

- download and install ise_81i_ip_update1.zip

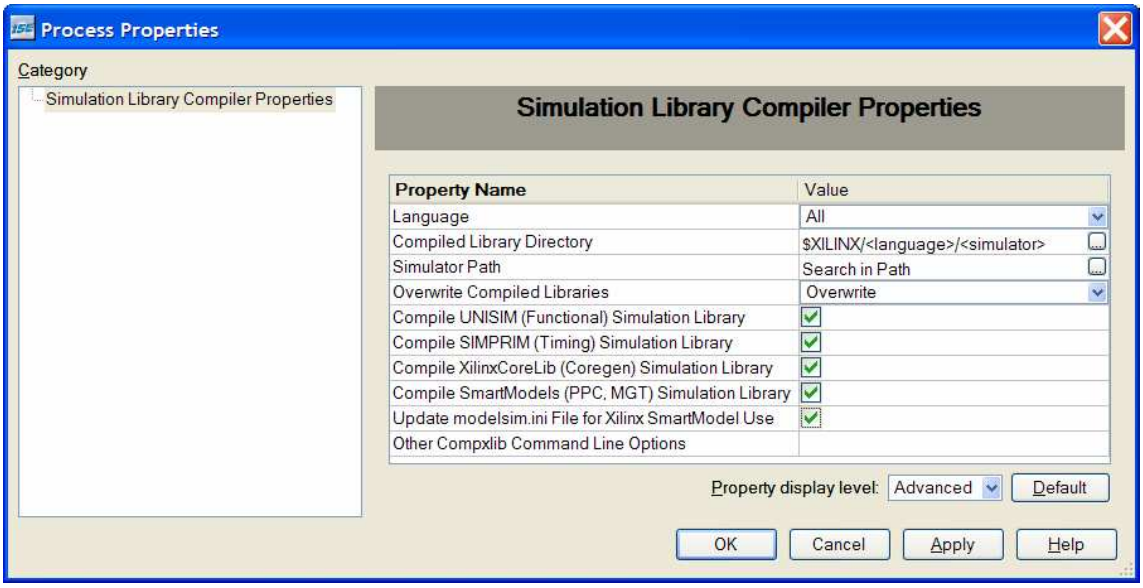
9.2.3 Compiling the Xilinx VHDL Libraries

Follow this procedure to compile the HDL Simulation Libraries with ISE 8.1.

- Open one of the delivered projects by double clicking on one ISE project file.
- Select the FPGA in the source window (here xc2vp70-6ff1517).
- Select the Compile HDL Simulation Libraries process in the Processes window.



- Set the properties for Compile HDL Simulation libraries as indicated below.



- Run the Compile HDL Simulation libraries process.

The file modelsim.ini shall be adapted to simulate the MGT (Multi Gigabit Transceiver) of the SC products. Note that the MGT simulation models are delivered as encrypted SWIFT models and the SWIFT interface must be enabled and correctly configured to simulate these models.

The above compilation changes the file “modelsim.ini” within the installation directory of ModelSim. If the used “modelsim.ini” file is a different one, as is the case in all modelsim directories of the FDK libraries, the following modifications shall be applied manually to these files:

Edit the modelsim.ini file related to the top level instance of your design (the one that is used to launch the simulation). It should be located at $AcqirisFDKRoot/ac240_developer_lib/modelsim if the design to simulate is developed from the ac240_developer_lib library as recommended.

Set the simulator resolution to 1ps

```
; Simulator resolution

; Set to fs, ps, ns, us, ms, or sec with optional prefix of 1,
10, or 100.

Resolution = ps
```

Set the libsm and libswift variables as following:

```
[lmc]

; The simulator's interface to Logic Modeling's SmartModel SWIFT
software

libsm = $MODEL_TECH/libsm.dll

libswift=$LMC_HOME/lib/pcnt.lib/libswift.dll

…

; The simulator's interface to Logic Modeling's hardware modeler
SFI software

libhm = $MODEL_TECH/libhm.sl
```

9.3 Library ac240_developer_lib

As the name suggests, this library, or a copy of it, should be the design library for new firmware(s). It contains the base designs, their test benches, and the user block core skeletons. The user block core skeletons are already instantiated in the base designs.

Component	Short Description
ac240_top_sysclk_ddr_tb	Base Design Test Bench for vhdl simulation
ac240_top_sysclk_tb	Base Design Test Bench for vhdl simulation
ac210_top_sysclk_tb	Base Design Test Bench for vhdl simulation
sc240_top_sysclk_str1_tb	Base Design Test Bench for vhdl simulation
ac240_top_sysclk_ddr	Base Design for AC240 firmware with memory cores
ac240_top_sysclk	Base Design for AC240 firmware (no memory cores)
ac210_top_sysclk	Base Design for AC210 firmware (no memory cores)
sc240_top_sysclk_str1	Base Design for SC240 firmware
ac240_user_block	user block skeleton for AC240 firmware
ac210_user_block	user block skeleton for AC210 firmware
user_block_example	Example of interfacing the input data stream. It implements the In-Buffer of the base designs.

9.3.1 Key Components and Files

Otherwise specified, the file paths are relative to the library directory of the library **ac240_developer_lib** which is $AcqirisFDKRoot/ac240_developer_lib

Comment	File(s)
Specific to HdlDesigner	
HdlDesigner Graphical Design Data	./hdlgraphic/component_name/*
HdlDesigner Library mapping for projects	../HdlDesigner/Mapping/ac240.hdp
HdlDesigner Library mapping for shared projects	../HdlDesigner/Mapping/ac240_shared.hdp
Base Design Test Benches	Top level design for “generation”, “compilation”, and simulation: ./hdlgraphic/sc240_top_sysclk_str1_tb ./hdlgraphic/ac240_top_sysclk_ddr_tb ./hdlgraphic/ac240_top_sysclk_tb ./hdlgraphic/ac210_top_sysclk_tb
Base Design	Top level design for “generation”, “compilation”, and synthesis: ./hdlgraphic/sc240_top_sysclk_str1 ./hdlgraphic/ac240_top_sysclk_ddr ./hdlgraphic/ac240_top_sysclk ./hdlgraphic/ac210_top_sysclk
Base Design Tester	$AcqirisFdkRoot/lib_projects/ac240_fdk/hdlgraphic/ac240_top_sysclk_tester $AcqirisFdkRoot/lib_projects/ac240_fdk/hdlgraphic/sc240_top_sysclk_str1_tester
Acqiris Tester	$AcqirisFdkRoot/lib_common/acq_lib/hdlgraphic/acqt_acqiris_tester_top
Key VHDL file	
Test Benches for Simulation	Top level design for simulation: ./hdlgraphic/sc240_top_sysclk_str1_tb_struct.vhd ./hdltext/ac240_top_sysclk_ddr_tb_struct.vhd ./hdltext/ac240_top_sysclk_tb_struct.vhd ./hdltext/ac210_top_sysclk_tb_struct.vhd
Base Designs for Synthesis	Top level design for synthesis: ./hdlgraphic/sc240_top_sysclk_str1_struct.vhd ./hdltext/ac240_top_sysclk_ddr_struct.vhd ./hdltext/ac240_top_sysclk_struct.vhd ./hdltext/ac210_top_sysclk_struct.vhd
Base Design Tester	$AcqirisFdkRoot/lib_projects/ac240_fdk/hdltext/ac240_top_sysclk_tester_struct.vhd $AcqirisFdkRoot/lib_projects/ac240_fdk/hdltext/sc240_top_sysclk_str1_tester_struct.vhd
Acqiris Tester	$AcqirisFdkRoot/lib_common/acq_lib/hdltext/acqt_acqiris_tester_top_struct.vhd
Functional Simulation	
Test Benches Main scripts	./hdlgraphic/sc240_top_sysclk_str1_tb/struct.bd.info/Sim/Control.txt ./hdlgraphic/ac240_top_sysclk_ddr_tb/struct.bd.info/Sim/Control.txt ./hdlgraphic/ac240_top_sysclk_tb/struct.bd.info/Sim/Control.txt ./hdlgraphic/ac210_top_sysclk_tb/struct.bd.info/Sim/Control.txt

preference for modelsim	./modelsim/modelsim.ini

Precision Synthesis

Precision Synthesis script and setup	These scripts are generated by HdlDesigner and contain several absolute pathnames. Non HdlDesigner users must adapt these pathnames. ./precision/sc240_top_sysclk_str1_struct/hds/precision.tcl ./precision/ac240_top_sysclk_ddr_struct/hds/precision.tcl ./precision/ac240_top_sysclk_struct/hds/precision.tcl ./precision/ac210_top_sysclk_struct/hds/precision.tcl
Add file script	These scripts are generated by HdlDesigner and contain several absolute pathnames. Non HdlDesigner users must adapt these pathnames. ./precision/sc240_top_sysclk_str1_struct/hds/add_files.tcl ./precision/ac240_top_sysclk_ddr_struct/hds/add_files.tcl ./precision/ac240_top_sysclk_struct/hds/add_files.tcl ./precision/ac210_top_sysclk_struct/hds/add_files.tcl
Precision Synthesis config.	./hdlgraphic/sc240_top_sysclk_str1/struct.bd.info/Synthesis/Constraints/precision.tcl ./hdlgraphic/ac240_top_sysclk_ddr/struct.bd.info/Synthesis/Constraints/precision.tcl ./hdlgraphic/ac240_top_sysclk/struct.bd.info/Synthesis/Constraints/precision.tcl ./hdlgraphic/ac210_top_sysclk/struct.bd.info/Synthesis/Constraints/precision.tcl
Constraint file	Pad location constraints and timing constraints. Precision Synthesis will translate these constraints to ucf format and generate the output ucf file for ISE. ./hdlgraphic/sc240_top_sysclk_str1/struct.bd.info/Synthesis/Constraints/sc240_top_str1_struct.sdc ./hdlgraphic/ac240_top_sysclk_ddr/struct.bd.info/Synthesis/Constraints/ac240_top_struct.sdc ./hdlgraphic/ac240_top_sysclk/struct.bd.info/Synthesis/Constraints/ac240_top_struct.sdc ./hdlgraphic/ac210_top_sysclk/struct.bd.info/Synthesis/Constraints/ac240_top_struct.sdc
Edif output	The netlist file generated by Precision Synthesis in EDIF format for ISE: ./precision/sc240_top_sysclk_str1_struct/sc240_top_sysclk_str1_struct/sc240_top_sysclk_str1.edf ./precision/ac240_top_sysclk_ddr_struct/ac240_top_sysclk_ddr_struct/ac240_top_sysclk_ddr.edf ./precision/ac240_top_sysclk_struct/ac240_top_sysclk_struct/ac240_top_sysclk.edf ./precision/ac210_top_sysclk_struct/ac210_top_sysclk_struct/ac210_top_sysclk.edf
Ucf output	The constraint file generated by Precision Synthesis in Xilinx UCF format for ISE: ./precision/sc240_top_sysclk_str1_struct/sc240_top_sysclk_str1_struct/sc240_top_sysclk_str1.ucf ./precision/ac240_top_sysclk_ddr_struct/ac240_top_sysclk_ddr_struct/ac240_top_sysclk_ddr.ucf ./precision/ac240_top_sysclk_struct/ac240_top_sysclk_struct/ac240_top_sysclk.ucf ./precision/ac210_top_sysclk_struct/ac210_top_sysclk_struct/ac210_top_sysclk.ucf
Project file	./precision/sc240_top_sysclk_str1_struct/sc240_top_sysclk_str1_struct.psp ./precision/ac240_top_sysclk_ddr_struct/ac240_top_sysclk_ddr_struct.psp ./precision/ac240_top_sysclk_struct/ac240_top_sysclk_struct.psp ./precision/ac210_top_sysclk_struct/ac210_top_sysclk_struct.psp

ISE cores for all flows

NGC and EDIF file for subparts	NGC and EDIF files for pre-synthesized parts like the coregen component: ./cores/*

ISE EDIF flow for synthesis with Precision Synthesis

ISE Project file	./ise/sc240_str1/sc240_str1.ise ./ise/ac240_ddr/ac240_ddr.ise ./ise/ac240/ac240.ise ./ise/ac210/ac210.ise
Constraint File	Otherwise notified here, the constraint file to use is the UCF constraint file generated by Precision Synthesis (see above). For the ac240 base design with the memory option, the following file shall be used: ./ise/ac240_ddr/ac240_ddr.ucf
Netlist File	The EDIF netlist generated by Precision Synthesis shall be used (see above)

ISE VHDL flow for synthesis with XST

ISE Project file	./xst/ac240_ddr/ac240_ddr.ise ./xst/ac240/ac240.ise ./xst/ac210/ac210.ise
Constraint File	For the base designs without the memory option, the constraint file to use is the UCF constraint file generated by Precision Synthesis (see above) For the base design with the memory option the following file shall be used: ./xst/ac240_ddr/ac240_ddr.ucf

9.4 Library ac240_fdk

This is the main library specific for the ac240 and ac210. In order to simplify upgrades to newer fdk versions, **a developer must never modify this library**.

New firmware shall be developed in the developer's library where the Acqiris Test Bench, the top level design, and the user core skeletons have already been copied.

Main component:

Component	Short Description
ac240_top_sysclk_ddr_tb	Base Design Test Bench for vhdl simulation
ac240_top_sysclk_tb	Base Design Test Bench for vhdl simulation
ac210_top_sysclk_tb	Base Design Test Bench for vhdl simulation
sc240_top_sysclk_str1_tb	Base Design Test Bench for vhdl simulation
ac240_top_sysclk_ddr	Base Design for AC240 firmware with memory cores
ac240_top_sysclk	Base Design for AC240 firmware (no memory cores)
ac210_top_sysclk	Base Design for AC210 firmware (no memory cores)
sc240_top_sysclk_str1	Base Design for SC240 firmware
ac240_top_sysclk_tester	Tester for all AC2x0 base designs
sc240_top_sysclk_str1_tester	Tester for the SC240 base design
str1_example	Example of generating frame and streaming data to the ODL of a SC240.

9.4.1 Key Components and Files

The structure and files are identical to those of the library ac240_developer_lib. Only the design files are included; there are no downstream files, either for synthesis or for simulation.

9.5 Library fdk_lib

This is the library for common component shared between multiple FDKs. The table below is an abstract of the available components or cores. The components listed as a "Core" in the table below are described in the Chapter FDK Core Library. Only components listed in the table to be part of "This FDK" should be used.

Component	This FDK	Core	Short Description
DE_BUFFER	☑		4 Ram blocks to implement the DE-Buffer. 128+16 bit wide Input port 128+16 bit wide output port
de_chip_io	☑		Xilinx IO primitives for the core de_interface_*, single channel
de_controller	☑		State machine for de_interface startup
de_data_mix_8	☑		Convert to signed an 8-bit vector
de_dataformat	☑		Format the data in the stream, with pipeline. Signed / Unsigned. Convert gray to binary.
de_dff_r_8	☑		Constrained 8-bit dff with reset
DE_IBTARGET_IND_4_8X32	☑		State machine controlling IB-BUS reading or writing from/to the DE-Buffer
de_interface_1ch	☑	☑	1 channel Interface for data input from the ADC de-multiplexer.
de_interface_2ch	☑	☑	2 channel Interface for data input from the ADC de-multiplexer
de_interface_2ch_rg	☑	☑	2 channel Interface for data input from the ADC de-multiplexer for the SC240 with the high resolution trigger core
de_rd_mux	☑		Mux for DE-Buffer to IB-BUS readout
de_stream_start	☑		Glue logic for de_interface control

lb_chip_io	☑		Xilinx IO primitives for the core lb_interface
lb_cpld	☑		-- actually unused
lb_ibtarget_bram_4_8x32	☑		State machine controlling IB-BUS reading or writing from/to the IN-Buffer
lb_ibtarget_io_base	☑		Example implementing a single 32-bit register
lb_ibtarget_io_base2x	☑		Example implementing a dual 32-bit register
lb_ibtarget_io_base3x	☑		Example implementing a triple 32-bit register
lb_ibtarget_io_base4x	☑		Example implementing a quad 32-bit register
lb_interface_m	☑	☑	A wrapper to the core lb_interface of the library fdk_lib_h. A synthesized edif version is delivered for "Place and Route"
slc_controller	☑	☑	A wrapper to the core slc_controller of the library fdk_lib_h. A synthesized edif version is delivered for "Place and Route"

9.6 Library fdk_lib_h

Not described in details. This library contains the design files for the Local Bus interface. Versions and changes will not be documented unless a major change seriously affects customer designs.

9.7 library std_lib

Not described in details. Basic logic functions. The name of a component informs the developers on its function. Any developer could use it. See the file(s) for details. Versions and changes will not be documented unless a major change seriously affects customer designs.

9.8 Library acq_lib

Not described in details, only a short description is given here:

Component	Short Description
ACQT_ACQIRIS_TESTBENCH	Test bench for acqt_acqiris_tester
acqt_acqiris_testbench_lb	Local Bus Target for test
acqt_acqiris_tester_top	The main tester component. Script File parsing, signals generation for emulation of Local Bus, MAC100 DE port, Clock generation, and signal Probing. It performs automatic test and reports the test results to the modelsim transcript window.
acqt_clock_gen_interface	Clock generation.
acqt_execom	Get the command from the parser, dispatch the command to the dedicated controller.
acqt_interleaver_mac	Interleaver to emulate two interleaved MAC100.
acqt_ibus_interface	To emulate the internal bus.
acqt_lbus_interface	Local Bus control and automatic test for readings.
acqt_mac	MAC100 DE port emulation
ACQT_MAC2	Additional MAC100 DE port control for interleaved acquisition.
acqt_menu	Display the simulation menu in the modelsim transcript window, control the parsing mode of actq_parser.
acqt_parser	Parse the script file, send the command to acqt_execom.
acqt_probe_interface	Probe control
acqt_run	Run control
acqt_sram_zbt_pipelined	Zbt sram model for test
FUNCTION_LIB	Package for specific acqt functions
type_def	Package for definitions of acqt specific types

9.9 Library std_xilinx

Xilinx primitives: The complex function are mapped to use the Xilinx model from the library unisim. Synthesis tools will recognize these components as black box.

9.10 Library cypress

Simulation model for the SRAM memory.

Component	Short Description
cy_dual_port_0852	Cypress model for the dual port SDR SRAM memory chip

9.11 Library samsung_ddr

Simulation model for the DRAM memory.

Component	Short Description
k4h511638b_b3	AC2x0 DDR memory Bank. It is a combination of four DDR memories
k4h511638b_b3x4	Samsung model for the DDR DRAM memory chip

9.12 Library ddr_ctrl_virtex2

Micron simulation models for the DDR memory.

Component	Short Description
DDR_CTRL	Top level of the DDR controller

9.13 Version History

Date	FDK Version	Comments
September 05	Beta 4	Added Internal Bus Port to acq_lib.acqt_acqiris_tester_top
February 06	Beta 6	Added library samsumg_ddr, cypress, ddr_ctrl_virtex2, std_xilinx Removed the library std_virtex2
May06	Beta7	Reviewed the entire chapter, add information for the XST flow.
January 07	1.0	New SC240 Base Design