Sidekiq X2/X4 PDK

FPGA Development Manual

Release 3.17.1

Jan 27, 2023

Table of Contents

List of Figures

List of Tables

Disclaimer

Epiq Solutions is disclosing this document ("Documentation") as a general guideline for development. Epiq Solutions expressly disclaims any liability arising out of your use of the Documentation. Epiq Solutions reserves the right, at its sole discretion, to change the Documentation without notice at any time. Epiq Solutions assumes no obligation to correct any errors contained in the Documentation, or to advise you of any corrections or updates. Epiq Solutions expressly disclaims any liability in connection with technical support or assistance that may be provided to you in connection with the Information.

THE DOCUMENTATION IS DISCLOSED TO YOU "AS IS" WITH NO WARRANTY OF ANY KIND. EPIQ SOLUTIONS MAKES NO OTHER WARRANTIES, WHETHER EXPRESSED, IMPLIED, OR STATUTORY, REGARDING THE DOCUMENTATION, INCLUDING ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT WILL EPIQ SOLUTIONS BE LIABLE FOR ANY CONSEQUENTIAL, INDIRECT, EXEMPLARY, SPECIAL, OR INCIDENTAL DAMAGES, INCLUDING ANY LOSS OF DATA OR LOST PROFITS, ARISING FROM YOUR USE OF THE DOCUMENTATION.

Revision History

Date	Version	Description
09/27/2017	3.7	Initial Release
02/28/2018	3.8.1	Update instructions for configuring FPGA and other minor updates
06/07/2018	3.9.0	Clarification on versioning
08/23/2018	3.10.0	Add support for external GPS module. Fix user_app timing constraints. Add support for Reference DAC SPI. Add iq multipliers. Move location of golden image. First X4 release.
10/11/2018	3.10.1	Document change, Section 8.3, change -c 1 to -c 0 Fix Tx Late timestamp getting stuck where the data_rd_counter reset is not synchronized to the Tx clock domain causing metastability and incorrect reset state for data_rd_counter once the reset pulse goes low.
01/05/2019	3.11.0	PCIe ip block now has its boundary at the core_wrapper. Add timing optimization looping to the build script. Add manual trigger. X2, added 245.76 Mhz ObsRx. X4, added dual two channel phase coherent and four channel independent Rx modes. Added single channel on dual JESD lane for ObsRx in anticipation of 491.52 MHz ObsRx.
02/22/2019	3.11.1	X4, update timing constraints to support 250 Mhz
06/25/2019	3.12.0	Add supoort for high sample rate Tx where dual JESD lanes are required for a single channel. Modified high speed ObsRx on X2 such that get two samples for clock. Add decimator on channels Rx1/Rx2. For X4, all Rx sample clocks are now driven from the same Rx clock source, all ObsRx sample clocks are now driven from the same ObsRx clock source, and all Tx sample clocks are now driven from the same Tx clock source. Fix PCIe Tx when packet size is larger than FIFO size. Add frequency hopping control logic. Upgrade to Vivado 2017.4 from 2016.4. Add iq swap mode. Added JESD Tx sample data memories to user_app.
10/04/2019	3.12.1	Minor changes to improve JESD synchronization. Doubled the PCIe Rx FIFO size on X4 ObsRx ports to support 491.52 MHz sample rates so that the Rx FIFO does not go full.
12/6/2019	3.13.0	Add support for User App GPIO control and user app RFIC SPI interface. Add user_pdk_config.v parameter file. X4 Only, Add support for transmitting via PCIe on Tx channels A1 and B1. Add WBSTAR 100 ms delay when reconfiuring the FPGA via ICAPE. X4 Only, Double the Rx FIFO size to help with timestamp gaps at higher sample rates.
6/05/2020	3.13.1	X4 Only, Updates for RevC board testability. X4 Only, Increase Rx and ObsRx FIFO depths. X2 Only, update Rx packet header to v1.
8/02/2020	3.14.0	Fix dual Tx underrun reporting bug.
01/14/2021	3.14.1	Upgrade to 8 lane PCIe for HTG-K800 and HTG-K810. Fix for starting/stopping streaming on a 1PPS edge.
04/06/2021	3.15.0	Fix Tx Timestamp dropped packets after late bug. Fix JESD Tx bug.

continues on next page

Table 1 – continued from previous page

04/12/2021	3.15.1	Add bus_synch_fifo to register interface of the decimators. Fix missing timestamp reset related to clock crossing synchronization on the register interface. Add register reset capability. Add BASELINE_VCS_STATUS register. Add frc_sel_for_tx to be used in Tx timestamp mode. X4 Only, Add FPGA TDD mode
01/27/2022	3.16.1	Fix Tx timestamp mode inter-packet bug. Fix Tx 65,532 block size bug. Remove extra flop delay of user_meta and sys_meta so that they line up with first sample in packet. Fix in PCIe Tx FIFO logic such that packet data is still correct even if user_app is not constantly requesting data and there is gaps between blocks.
10/26/2022	3.17.1	Increase rx_dma_meta_fifo sizes to increase throughput rate in Rx low latency mode. X4 xcku115 Only, Add 1 ms POR to the x4 xcku115 (htgk8100 and htgk810) PCIe module X4 Only, Update JESD Rx path for deterministic latency

1 Overview

1.1 About this Document

This document provides the necessary details for developing FPGA applications for use with the Sidekiq™ X2 or Sidekiq™ X4 multichannel RF transceiver card developed by Epiq Solutions [5]. It is provided with the purchase of a SidekiqX2 or a SidekiqX4 Platform Development Kit.

1.2 Legal Considerations

The SidekiqX2 and SidekiqX4 are distributed all over the world. Each country has its own laws governing the reception and/or transmission of radio frequencies. The user of the SidekiqX2 or SidekiqX4 and associated software is solely responsible for insuring that it is used in a manner consistent with the laws of the jurisdiction in which it is used. Many countries, including the United States, prohibit the reception and/or transmission of certain frequency bands, or receiving certain transmissions without proper authorization. Again, the user is solely responsible for the user's own actions.

1.3 Proper Care and Handling

The SidekiqX2 or SidekiqX4 unit is fully tested by Epiq Solutions before shipment, and is guaranteed functional at the time it is received by the customer, and **ONLY AT THAT TIME**. Improper use of the SidekiqX2 or SidekiqX4 unit can cause it to become non-functional. In particular, a list of actions that may cause damage to the hardware include the following:

- Handling the unit without proper static precautions (ESD protection) when the housing is removed or opened up
- Inserting or removing the unit from a host system when power is applied to the host system
- Connecting a transmitter to an Rx port without proper attenuation
- Executing custom software and/or an FPGA bitstream that was not developed according to guidelines

The above list is not comprehensive, and experience with the appropriate measures for handling electronic devices is required.

1.4 Introduction

The Sidekiq™ X2 or Sidekiq™ X4 Platform Development Kit (PDK) provides the ability for users to create their own custom applications. This is accomplished by customizing software and/or the RTL code that configures the host FPGA. This manual gives an overview of the FPGA reference design, with the intention of empowering the user to build upon the design to create custom applications.

Detailed information about the software environment, including how to create custom software applications, can be found in a separate document, the Sidekiq Software Development Manual [3], which can be downloaded from the Epiq Solutions support website [2]

In addition, the details of the hardware itself and system design of the unit is outside the scope of this document. For more details about the hardware, please download and review the Sidekiq X2 Hardware User's Manual [4] or the Sidekiq X4 Hardware User's Manual [6]. It is strongly recommended that the user read this document thoroughly before attempting to dive into FPGA development. It is also a requirement for the user to be experienced with Xilinx® FPGA devices, design flow and software tools, and in particular Vivado®.

This manual is meant to concisely describe the FPGA reference design, but it is important to spend time reviewing the actual design (i.e. RTL source code, build scripts, constraint files, etc.) to fully understand the design. The sections of the manual were intentionally created to align with the basic hierarchy of the design, and the source code itself is commented and will act as a supplement to the information provided here.

1.5 Terms and Definitions

Table 1.1: Terms and Definitions

Term	Definition
1PPS	1 Pulse Per Second
ADC	Analog to Digital (A/D) converter
DAC	Digital to Analog (D/A) converter
DSP	Digital Signal Processing
FIFO	First In First Out
FPGA	Field Programmable Gate Array
FRC	Free-running counter
I/Q	In-Phase / Quadrature Phase
<H>	Software channel handle
IP	Intellectual Property
JESD204B	A multigigabit serial data converter interface standard
MHz	Megahertz
PC	Personal Computer
PCIe	PCI Express
PDK	Platform Development Kit
RF	Radio Frequency
RFIC	RF Integrated Circuit (AD9371)
RTL	Register Transfer Level
Rx	Receive
SDK	Software Development Kit
SDR	Software Defined Radio
SPI	Serial Peripheral Interface
Tx	Transmit

2 FPGA Reference Design

2.1 Overview

The Sidekiq™ X2 or the Sidekiq™ X4 PDK provides a complete FPGA reference design that enables a user to quickly and efficiently create custom applications targeting SidekiqX2 or SidekiqX4 installed a compatible Thunderbolt™ 3 chassis hosting a Xilinx® Kintex® Ultrascale™ family FPGA. The supported carriers and devices are listed in Table 3.1 of Section 3.1.

The unmodified reference design provides a full FPGA implementation to flow raw Rx I/Q samples from up to three ADC channels for the X2 (Rx1, Rx2, and ObsRx), and up to six for the X4 (Rx1, Rx2, and ObsRx from each RFIC) to the host system processor and to transmit from the host up to two DAC channels (Tx1 and Tx2). The PCIe data transport mode is used on the SidekiqX2 and SidekiqX4 platforms.

The Rx path transfers baseband I/Q samples which are received from an Analog Devices AD9371 RFIC that is located on the SidekiqX2 FMC card or from two Analog Devices AD9379 RFICs that are located on the SidekiqX4 FMC card via a JESD204B receive interface. These samples pass into a processing block which allows the user to process, timestamp, and transfer the samples via a FIFO interface to a PCIe subsystem. In the reference design, the user application processing block functions as a simple pass-through which timestamps and drives the PCIe FIFO with no changes to the samples. The 16-bit I and 16-bit Q components are concatenated to form a 32-bit wide data bus.

On the Tx side, I/Q samples are first transferred from the host system to FPGA over PCIe, then they are transferred to the RFIC on the SidekiqX2 or SidekiqX4 FMC card via a JESD204B transmit interface. Similar to the receive side, transmit data consists of 16-bit I and 16-bit Q to form a 32-bit wide data bus. In Fig. 2.1, orange design blocks are ip modules and cannot be changed. Red blocks are available in RTL, but should not be modified. Yellow blocks should not require modification, but certain applications may necessitate changes. Green blocks are the intended targets for user modification, and most applications should only require changes to these green blocks. Please note that the SidekiqX4 (not shown in the diagram) is the same as SidekiqX2 except that it has three additional channels (Rx1, Rx2, and ObsRx) coming from the second RFIC chip and two additional JESD modules (JESD204 chC and chD) for interfacing these channels to the user_app.

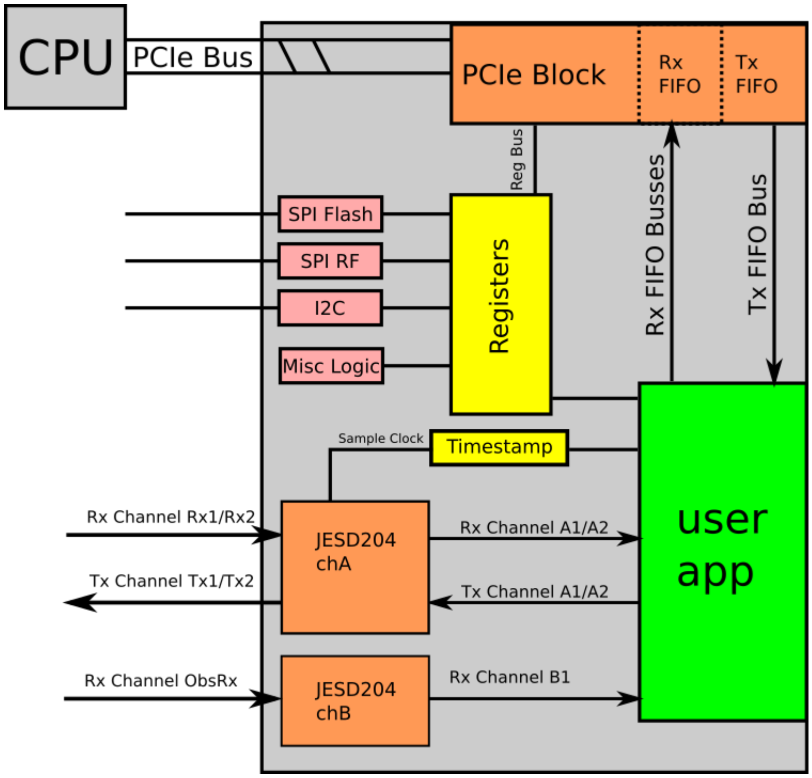

Fig. 2.1: Sidekiq X2/X4 Simplified Block Diagram

2.2 FPGA Top Level

The Top Level block, `sidekiq_x2_top` (`sidekiq_x4_top` for the X4) is a wrapper containing the top level RTL and all submodules. This section will serve to describe each block's functions and use. Sections that have more significant impact on a PDK user will be discussed in greater detail. Certain features of modules within the top level PDK can be configured by the user using the user_pdk_config.v file. The user_pdk_config.v gets included in the top level via a `` `include``. The file contains localparam statements that control how the PDK is built. PDK features that are not needed for a specific application can be turned off to reduce the logic utilization of the PDK. Other parameters control memory depths to balance the block RAM needs of the application.

2.2.1 PCIE Module

The `pcie_module` module implements the data transport interface to the host processor. To conserve resources within the FPGA, the user may configure the number of channels implemented via build time parameters. If the different receive and/or transmit channels are not required for the user's application, the user may update the parameters set in `user_pdk_include.v` to configure the module. The parameters are defined in Table 2.1.

Table 2.1: PCIE Module Configuration Parameters

Parameter	Range	Description
NUM_RX	0,1, or 2	Number of Rx DMAs to build into the design for RFIC A.
NUM_RX_B	0,1, or 2	Number of Rx DMAs to build into the design for RFIC B. (X4 Only)
NUM_OBSRX	0,1, or 2	Number of Rx Observation port Rx DMAs to build into the design.
TX_CAPABLE	0 or 1	Enables generation of the Tx transport logic. 0 = Disable Tx transport logic generation 1 = Enable Tx transport logic generation
NUM_TX	0,1, or 2	Number of Tx DMAs to build into the design. Requires TX_CAPABLE=1.
VAR_LATENCY_REG_RD	0 or 1	Variable Latency register Read mode. 0=Disabled (legacy); 1=Enabled

2.2.2 Timestamp Module

The `timestamp_block` module handles driving and resetting the free-running counters (FRC) which serve to timestamp samples. It also controls signaling to start and stop receipt/transmission of data based on an external PPS signal. There are three counters present: `frc_a`, `frc_b`, and `frc_sys` (five counters with X4 with additional `frc_c` and `frc_d`). `frc_a` is driven by `sample_clk_a`, `frc_b` is driven by `sample_clk_b`, `frc_c` is driven by `sample_clk_c`, `frc_d` is driven by `sample_clk_d`, and `frc_sys` is driven by the 100/122.88/153.6 MHz JESD reference clock. The reference clock frequency depends on the profile loaded by software. When not in reset, the counters will increment (and wrap) as long as there is a clock present. Time syncing multiple units is possible using the software API; see [3] for more information. `user_app` also has the ability to reset the timestamps by asserting `timestamp_rst`. The free-running counters will resume once `timestamp_rst` is cleared.

2.3 The User Application

The user application (`user_app.v`) and register interface (`user_reg_if.v`) are the RTL source files in which the majority of signal processing and user customization is expected to be performed. `user_reg_if` is a submodule of `user_app`, which allows the user to maintain and customize their own register space. In most cases, only the Verilog files `user_app.v` and `user_reg_if.v` will need to be modified to create custom FPGA images with advanced signal processing capabilities. The `user_app` structure is designed to allow for portable signal processing blocks between multiple Epiq SDR platforms, including all Sidekiq variants. This allows end users to share user_app modules between platforms, including future platforms, with minimal rework required.

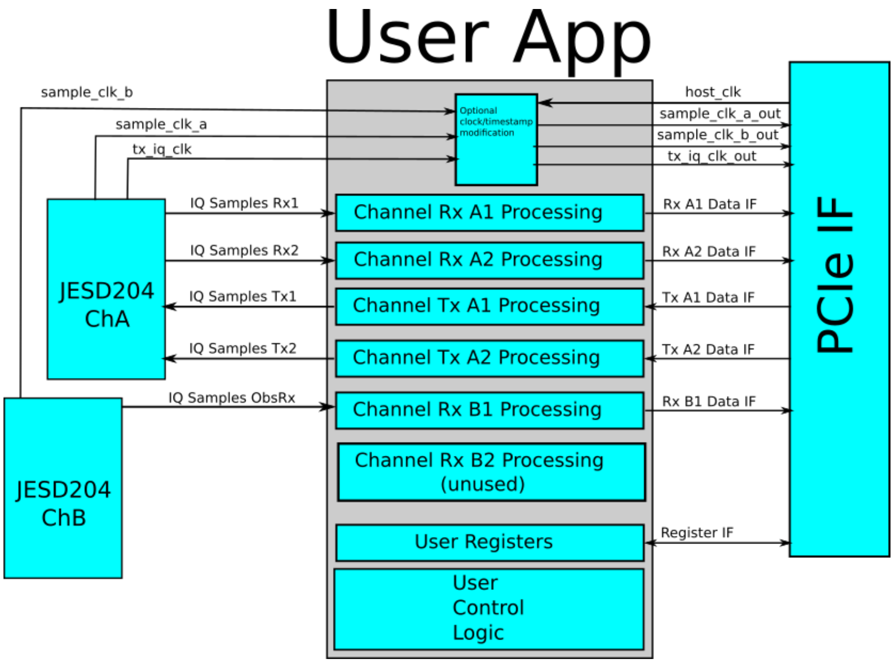

Fig. 2.2: Sidekiq X2/X4 User Application Block Diagram

2.3.1 User Application Interfaces

The following sections describe the various interfaces of the user_app module to the top level modules. All signal directions are referenced to the user app module.

This design has multiple logical data channels. The channels use the notation <H> to indicate the software handle associated with the channel. <H> can have values 'a1', 'a2', and 'b1' for Sidekiq X2, and 'a1', 'a2', 'b2', 'c1', or 'd1' for Sidekiq X4. These are logically mapped to the RFIC data ports differently depending on the Sidekiq platform and RFIC configuration. See Table 2.2 for the logical mapping of channels to signal names.

Table 2.2: RFIC Data Port Channel Mapping

Channel	Sidekiq X2 Connection	Sidekiq X4 Connection (ObsRx Disabled)	Sidekiq X4 Connection (ObsRx Enabled)
a1	RFIC Tx1/Rx1	RFIC_A Tx1/Rx1	RFIC_A Tx2/Rx2
a2	RFIC Tx2/Rx2	RFIC_A Tx2/Rx2	N/A
b1	RFIC ObsRx	RFIC_B Tx1/Rx1	RFIC_B Tx2/Rx2
b2	N/A	RFIC_B Tx2/Rx2	N/A
c1	N/A	N/A	RFIC_A ObsRx
d1	N/A	N/A	RFIC_B ObsRx

2.3.1.1 User App I/Q Data Sample Clocks

The table below, Table 2.3, shows how the various user_app RFIC sample clocks are driven in the top level RTL. Please note that both Sidekiq X2 and Sidekiq X4 have three separate sample clocks. This consists of an Rx sample clock, and ObsRx sample clock, and a Tx sample clock. Note that in the case of Sidekiq X4, the respective sample clocks are shared between RFIC A and RFIC B. As such, Sidekiq X4 still has three separate sample clocks just like Sidekiq X2, and as a result, the respective sample clock rates between RFIC A and B must be the same for Sidekiq X4.

Table 2.3: User App RFIC Sample Clock Mapping

User App Port	Sidekiq X2	Sidekiq X4
`sample_clk_a`	Rx1/Rx2 (jesd_rx_outclk_chA)	RFIC_A Rx1/Rx2 (jesd_rx_outclk_chA)
`sample_clk_b`	ObsRx (jesd_rx_outclk_chB)	RFIC_B Rx1/Rx2 (jesd_rx_outclk_chA)
`sample_clk_c`	N/A	RFIC_A ObsRx (jesd_rx_outclk_chB)
`sample_clk_d`	N/A	RFIC_B ObsRx (jesd_rx_outclk_chB)
`tx_iq_clk`	Tx1/Tx2 (jesd_tx_outclk_chA)	RFIC A Tx1/Tx2 (jesd_tx_outclk_chA) RFIC B Tx1/Tx2 (jesd_tx_outclk_chA)

2.3.1.2 System Clocks and Resets

Various clocks, resets, and enables are provided to the user application in order to support the system functions. The descriptions of these signals can be found in Table 2.4 Interface specific clocks and resets are defined in their respective interface section.

Table 2.4: Clocks and Resets Interface

Signal	Dir	Description
`rst`	in	Asynchronous reset, active high.
`host_clk`	in	Host System Clock. The frequency of this clock is 250 MHz when `VAR_LATENCY_REG_RD=0` and when `VAR_LATENCY_REG_RD=1`.
`sys_clk`	in	System Clock. Driven by the JESD reference clock. Runs at 122.88 MHz or 153.60 MHz depending on the RFIC profile loaded by the Sidekiq software. This clock runs the frc_sys timestamp counter.
`ddr_clk`	in	Reserved for future use.

2.3.1.3 System Control/Status Interface

The system control and status interface is comprised of signals relating to the overall system. The signals are described in Table 2.5. `reg_rx_<H>` is a 32-bit bus from the SW register interface. The bus contains individual status flags described in Table 2.6. Note that not all bits of `reg_rx_<H>` are defined.

Table 2.5: System Control/Status Interface

Signal	Dir	Description
timestamp_rst	out	Timestamp Reset. Used to clear the timestamps and free-running counters. Timestamps can also be reset by software.
reg_rx_<H>	in	A per-channel control register that allows the user to monitor the state of the channel as controlled by software.
frc_sys_in	in	Free running system counter. Synchronous to the 122.88/153.60 MHz JESD reference clock. It can be reset using a software-programmable PPS based reset, or by user logic using timestamp_rst.
frc_sys_in_pcie	in	A copy of frc_sys_in synchronized to host_clk. Note that the counter counts increments at the JESD reference clock rate.
reg_iq_swap	in	Register I/Q concatenation swap. This bit indicates the software requests the concatenation of I and Q samples into 32-bit buses should be reversed. I.E. typically I is loaded into the MSBs and Q loaded the LSBs of a concatenated data bus. If this signal is asserted the user application should pack Q into the MSBs and I into the LSBs of the concatenated data bus.
external_enable	in	External gating signal for transmit/receive. By default, it is wired at the top level to allow software to start/stop transmitting on a PPS edge. If this functionality is not needed but custom external gating is desired, this signal can be used.

Table 2.6: reg_rx<H> Field Description

Bit	Description
1	Rx DMA Data Source: 0 - I/Q Data 1 - Counter Data
2	Rx DMA Control: 0 - Rx DMA Disabled 1 - Rx DMA Enabled
5	Tx Data Interface Mode: 0 - Single Channel Mode 1 - Dual Channel Mode
8	Tx DMA Mode: 0 - Continuous Transmit Mode 1 - Tx on Timestamp Mode
9	Tx DMA Control: 0 - Tx DMA Disabled 1 - Tx DMA Enabled

2.3.1.4 Software Register Interface

The software register interface is a set of control signals used to access a user defined software accessible memory space. This memory space is used to provide control and status of the user application to the host processor. Register read and write transactions consist of single 32-bit (double word) accesses to the registers. The read interface can operate in one of two modes set by the VAR_LATENCY_REG_RD parameter in user_pdk_config.v. When VAR_LATENCY_REG_RD=0 the register read data must be present and valid on the reg_rd_data bus within 1 host_clk clock cycle of a valid address driven to the reg_rd_addr bus. VAR_LATENCY_REG_RD=1 configures the register interface for variable read latency mode. This mode should be used when the register read data can not be driven to the bus within 1 clock cycle. When configured in this mode reg_rd_data should be held low while reg_rd_rdy is not asserted. After a read request is issued, indicated by a pulse on reg_rd_en, the register interface shall place the read data on the reg_rd_data bus and assert the reg_rd_rd for 1 host_clk clock cycle to indicate the read data is valid.

Table 2.7: Software Register Interface

Signal	Dir	Description
reg_wr_addr	in	Register Write Address. The address write during a register write operation.
reg_wr_en	in	Register Write Enable. Pulse used to issue a register write operation. When this signal is asserted reg_wr_data should get written to the address indicated by reg_wr_addr
reg_rd_en	in	Register Read Enable. Pulse used to indicate a register read operation. reg_rd_addr is valid and indicates address of the requested data.
reg_wr_data	in	Register Write Data. The data to store into reg_wr_addr during a register write operation.
reg_rd_addr	in	Register Read Address. The address to access during a register read operation
reg_rd_data	out	Register Read Data. The data stored at reg_rd_addr
reg_rd_rdy	out	Register Read Ready. VAR_LATENCY_REG_RD=1 mode only. Handshaking pulse indicating the register read data is available on the bus.

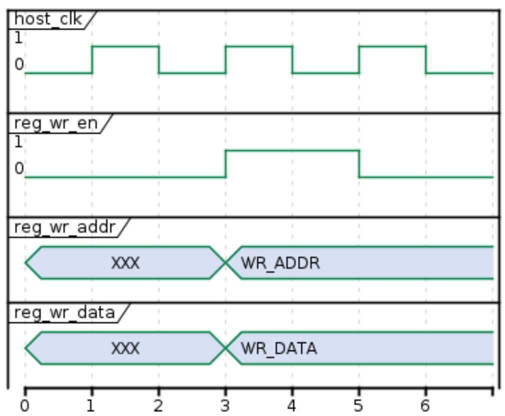

Fig. 2.3: Register Write Interface Timing Diagram

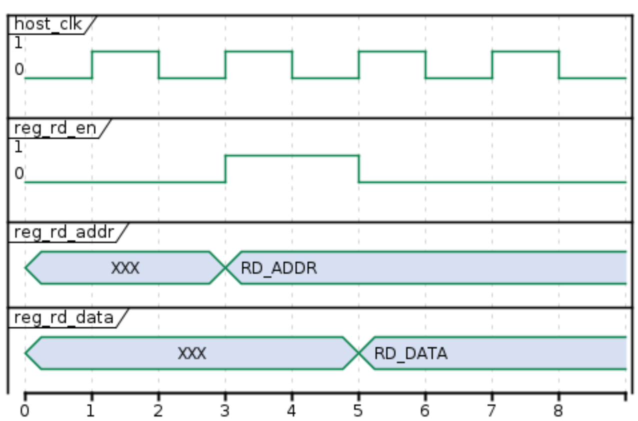

Fig. 2.4: Register Read Interface Timing Diagram (VAR_LATENCY_REG_RD=0)

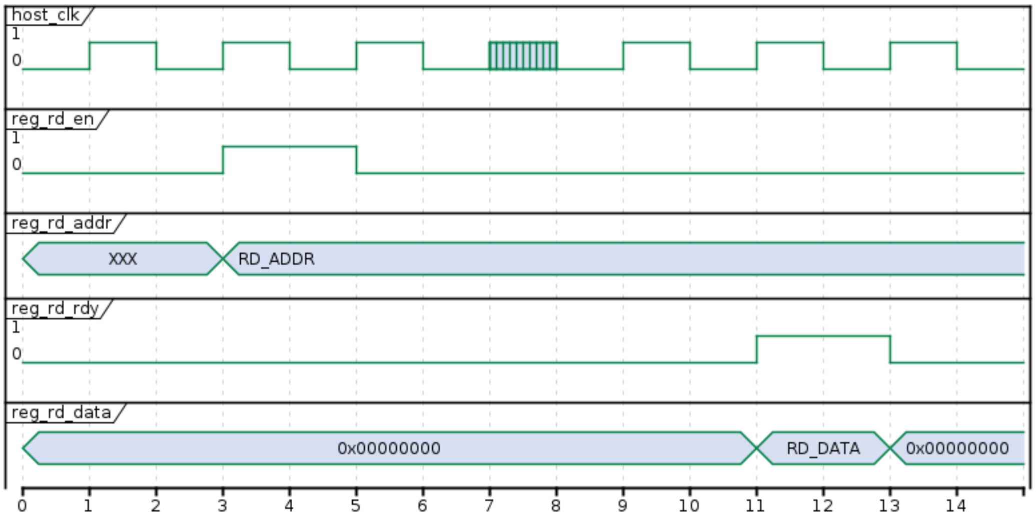

Fig. 2.5: Register Read Interface Timing Diagram (VAR_LATENCY_REG_RD=1)

2.3.1.5 Test Memory Interface

The JESD Tx interface can be driven from FPGA BRAM instantiated in the user application reference design. An example of how to use samples stored in FPGA BRAM to drive the Tx JESD Interface has been added to the user_app. There is also an example software test application that demonstrates how to set up and exercise this feature using the API. The various tx_test_mem modules are instantiated at the bottom of `user_app.v`. The various top level user_app ports associated with this functionality are described in Table 2.8.

Table 2.8: JESD Transmit Test Memory Interface

Signal	Dir	Description
`reg_tx_mem_internal_select`	out	Register bit to enable JESD Tx samples to be sourced from internal FPGA BRAM (instead of PCIe).
`jesd_tx_clk_b1_b2`	in	Tx clock, connected to tx_iq_clk in the sidekiq_x2/x4_top.v outside the user_app module. Not used on X2.
`jesd_tx_clk_a1_a2_dual_lane`	in	Tx clock, connected to tx_iq_clk in the sidekiq_x2/x4_top.v outside the user_app module. Only needed when JESD lane is used in single channel on dual lane mode.
`jesd_tx_clk_b1_b2_dual_lane`	in	Tx clock, connected to tx_iq_clk in the sidekiq_x2/x4_top.v outside the user_app module. Only needed when JESD lane is used in single channel on dual lane mode. Not used on X2.
`reg_soft_reset_tx`	in	Soft reset bit from the top level used to align the various memory read counters after JESD synchronization before coherent transmit starts.
`reg_dma_ctrl_rx1_tx_9`	in	Bit 9 of the reg_dma_ctrl_rx1 register synchronized into the Tx clock domain. Used to start Tx phase coherent streaming across the Tx memories.
`tx_rd_en_in_a1_dual_lane`	in	Similar functionality to tx_rd_en_in_a1. This read enable is only used in single channel on dual JESD lane mode for this channel.
`tx_rd_en_in_a2_dual_lane`	in	Similar functionality to tx_rd_en_in_a2. This read enable is only used in single channel on dual JESD lane mode for this channel.
`tx_rd_en_in_b1`	in	Tx read enable for RFIC B. Similar function to rf_rd_en_in_a1 for RFIC A. Not used for X2.
`tx_rd_en_in_b2`	in	Tx read enable for RFIC B. Similar function to rf_rd_en_in_a2 for RFIC A. Not used for X2.
`tx_rd_en_in_b1_dual_lane`	in	Tx read enable for RFIC B. Similar function to rf_rd_en_in_a1_dual_lane for RFIC A. Not used for X2.
`tx_rd_en_in_b2_dual_lane`	in	Tx read enable for RFIC B. Similar function to rf_rd_en_in_a2_dual_lane for RFIC A. Not used for X2.
`jesd_tx_mem_tdata_a1`	out	32 bit JESD Sample Tx data.
`jesd_tx_mem_tdata_a2`	out	32 bit JESD Sample Tx data.
`jesd_tx_mem_tdata_b1`	out	32 bit JESD Sample Tx data. Not used on X2.
`jesd_tx_mem_tdata_b2`	out	32 bit JESD Sample Tx data. Not used on X2.
`jesd_tx_mem_tdata_a1_dual_lane`	out	32 bit JESD Sample Tx data. Only used in single channel on dual JESD lane mode for this channel.
`jesd_tx_mem_tdata_a2_dual_lane`	out	32 bit JESD Sample Tx data. Only used in single channel on dual JESD lane mode for this channel.
`jesd_tx_mem_tdata_b1_dual_lane`	out	32 bit JESD Sample Tx data. Only used in single channel on dual JESD lane mode for this channel. Not used on X2.
`jesd_tx_mem_tdata_b2_dual_lane`	out	32 bit JESD Sample Tx data. Only used in single channel on dual JESD lane mode for this channel. Not used on X2.

2.3.1.6 RFIC Control Interface

The RFIC Control Interface provides access to the RFICs digital interface in two ways. The first is the RFICs GPIO pins. The signals of GPIO interface are described in Table 2.9. The second, is the RFIC's SPI Bus interface. The signals of SPI Bus interface are described in Table 2.10. Both the GPIO and SPI interfaces are synchronous to the `host_clk`.

Table 2.9: RFIC Control GPIO Interface

Signal	Dir	Description
adc_dig_bus_ctrl	out	RFIC Digital Bus Control Override. A set bit causes the user app to override the top level control of the corresponding bit of the RFIC's digital bus. (X2 only)
adc_dig_bus_tristate	out	RFIC Digital Bus Tristate Enable. A set bit tristates the corresponding bit of the RFIC's digital bus. The corresponding bit of adc_dig_bus_ctrl must also be set. (X2 only)
adc_dig_bus_in	in	RFIC Digital Bus Input. When adc_dig_bus_ctrl is 0 or adc_dig_bus_tristate is 1, this bus is the value of the RFIC's Digital GPIO bus. (X2 only)
adc_dig_bus_out	out	The value to drive onto the RFIC's Digital GPIO bus when the corresponding bit of adc_dig_bus_ctrl is set and adc_dig_bus_tristate is cleared. (X2 only)
adc_dig_bus_b	in	RFIC Digital Bus Input. This bus is the value driven by the RFIC onto its Digital GPIO pins. (X4 only)

Caution: Overriding the RFIC Digital control bus may break features provided by the Sidekiq SDK. Only override the bits of the bus when the default behavior is not required for the application.

Table 2.10: RFIC Control SPI Interface

Signal	Dir	Description
spi_wr_en	out	SPI Write Enable Flag. 0 - Perform SPI Read Operation. 1 - Perform SPI Write Operation.
spi_addr	out	SPI Address. Address of the SPI register to Read/Write
spi_wr	out	SPI Write Data. Data to write to the SPI register.
spi_select	out	SPI Interface Select. Indicates which chip select to enable during the SPI transaction. 0 - AD9371 RFIC 1 - AD9528 JESD204 IC 2 - N/A 3 - 10 MHz REF_DAC
spi_start	out	SPI Command Start. Single pulse that initiates the SPI transaction
spi_done	in	SPI Done. An signal indicating the state of the SPI transaction. This signal goes low after a SPI command has been issued and returns high when the command completes.
spi_rd	in	SPI Read Data. This bus contains the read data after a SPI read transaction has completed.

2.3.1.7 RFIC Receive Data Interface

The RFIC Rx data interface is responsible for moving received I/Q data from the RFIC into the user application. While samples_valid is high, each rising edge of the sample clock delivers a new sample consisting of sixteen bits each of I and Q data. Each channel must be enabled by software before the samples_valid signal will go high. The exception to this is when ObsRx is used on the Sidekiq X4. To meet the higher sample rates needed, the JESD is configured for a single channel on dual JESD lane mode. In this scenario, the channel is clocked at one half of the sample rate and two samples (two sixteen bit samples for both I and Q) are provided each clock edge giving the desired sampling rate. The even samples are placed on the `i_sample_<H>` and the `q_sample<H>` ports while the odd samples are placed on the `i_sample<H>_dual_lane` and `q_sample<H>_dual_lane` ports. `frc_<H>` is a free running counter that starts when the channel associated with it is enabled and increments at the corresponding `sample_clk_<H>` rate. Each counter continually increments as long as its associated clock is running. The counters can be reset using a software-programmable PPS based reset or by user logic. Fig. 2.6 shows the relationship between the sample clock, data buses, and free running counter.

Note: Samples are valid only when samples_valid_<H> is high, but frc_<H>_in increments as long as there is a valid clock.

Table 2.11: RFIC Receive Data Interface

Signal	Dir	Description
sample_clk_<H>	in	Clock derived from he JESD Rx interface. It runs at the Rx sample rate.
i_sample_<H>	in	In-phase sample data
q_sample_<H>	in	Quadrature-phase sample data
samples_valid_<H>	in	Sample Valid Flag. High indicates sample data on the bus is valid
frc_<H>_in	in	Sample counter synchronized to sample_clk_<H>.
frc_<H>_in_pcie	in	Sample counter synchronized to host_clk.
i_sample_<H>_dual_lane	in	In-phase dual lane data path. When sample rates exceed 290 MHz this data path deliverers the odd numbered samples. Only valid for <H> = c1 and d1. (X4 only)
q_sample_<H>_dual_lane	in	Quadrature-phase dual lane data path. When sample rates exceed 290 MHz this data path delivers the odd numbered samples. Only valid for <H> = c1 and d1. (X4 only)

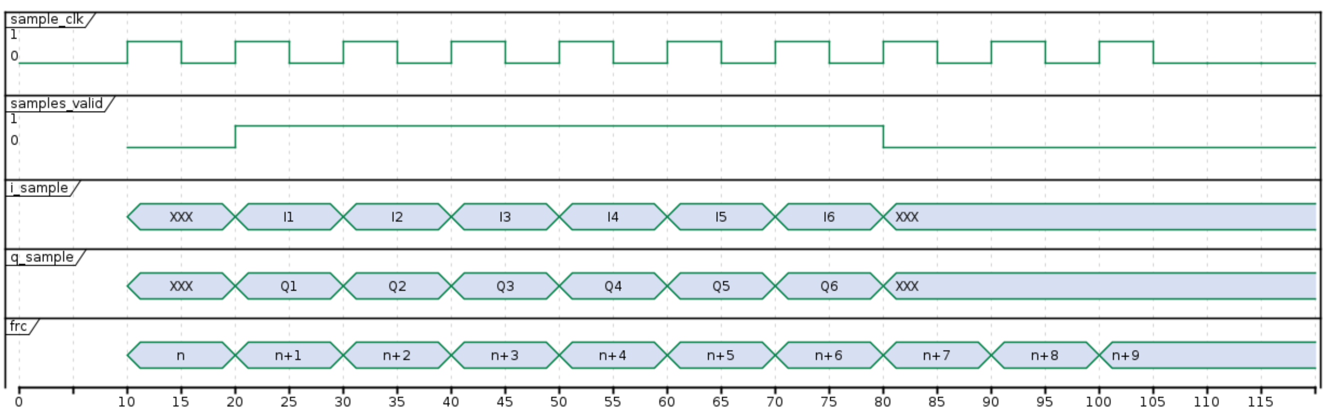

Fig. 2.6: Rx Data Interface Timing Diagram

2.3.1.8 RFIC Transmit Data Interface

The RFIC Tx interface moves I/Q data from the user application to be transmitted by the RFIC.

Table 2.12: RFIC Transmit Data Interface

Signal	Dir	Description
tx_i_<H>	out	In-phase sample data to be transferred to the DAC
tx_q_<H>	out	Quadrature-phase sample data to be transferred to the DAC
tx_dl_i_<H>	out	In-phase dual lane data path.
tx_dl_i_<H>	out	Quadrature-phase dual lane data path.
tx_rd_en_in_<H>	in	Flag indicating tx data interface is ready for data. This flag must be high prior to asserting tx_rd_en_out<H>.
tx_dac_en_out_<H>	out	Flag indicating tx_i_<H> and tx_q_<H> are valid data.

2.3.1.9 Rx Transport DMA Interface

The Rx transport DMA interface is responsible for moving data from the user app to the firmware running on the host processor. DMA frames are transferred to the host processor in 1018 32-bit data word frames. 6 header words are appended to the data frame for a total of 1024 words sent to the host processor. The `full_<H>` flag is used to throttle the data into the buffer. When it is asserted the user should halt data transfer to the Rx transport DMA interface. The user can write an additional 512 words into the buffer after `full_<H>` is asserted without overflowing the FIFO. To handle bursty data the user can add additional buffering within the user application, or drop data as necessary. The interface timing fort the Rx transport DMA interface is shown in Fig. 2.7.

Table 2.13: Receive Transport DMA Interface

Signal	Dir	Description
`sample_clk_<H>_out`	out	Clock synchronous to `fifo_din_<H>` used to drive the Rx transport DMA interface.
`fifo_din_<H>`	out	FIFO Data In. 32-bit data word to be transferred to the host processor.
`fifo_din_<H>_dual_lane`	out	FIFO Dual Lane Data In. Additional 32-bit data word to be transferred to the host for higher throughput transfers.
`fifo_wren_<H>`	out	FIFO write enable. Indicates the data on the `fifo_din` data buses are valid.
`full_<H>`	in	FIFO Full Flag. Indicates the transport DMA FIFO can not fit a full DMA frame of data.
`user_meta_<H>`	out	32-bit word that is appended as the 6th word of the transport header. The metadata is sampled on the first word of each 1018 word data set.

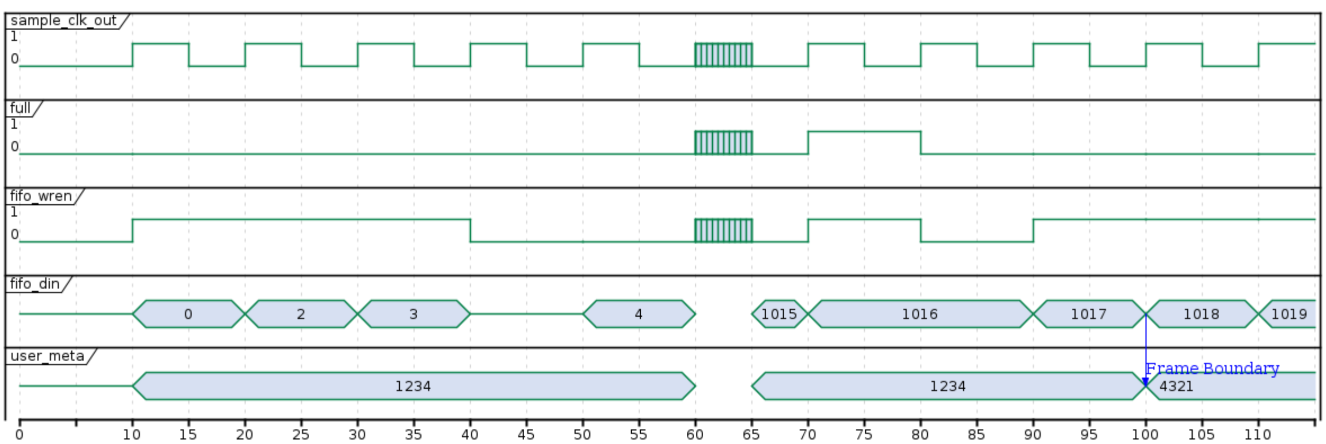

Fig. 2.7: Rx FIFO Interface Timing Diagram

2.3.1.10 Tx Transport DMA Interface

The Tx transport DMA interface is responsible for moving data from firmware running on the host processor to the user application. During times when there is not any transmit data available to be sent, the JESD Tx lanes will send all zeros to the RFIC.

Table 2.14: Transmit Transport DMA Interface

Signal	Dir	Description
tx_din_<H>	in	Data from the host processor
tx_din_dl_<H>		Dual lane data from the host processor
tx_rd_en_out_<H>	out	Flag indicating the user application is ready to process data from the host
tx_dac_en_in_<H>	in	Flag indicating the data on tx_din_<H> (and tx_din_dl_<H>) is valid.
tx_ts_<H>	in	Transmit timestamp. Indicates the timestamp (on the frc_<H> domain) when the DMA will assert tx_dac_en<H> when in transmit on timestamp mode.
tx_err_<H>	in	Transmit Error. Indicates that a timestamp error has occurred, which typically when data is not able to be pushed down over the PCIe bus fast enough to transmit a packet at the correct time.
tx_empty_<H>	in	Transmit FIFO empty. Flag indicating an underflow condition occurred.

In the reference design, each 32-bit data word tx_din_<H> from the Tx transport DMA interface contains a single I and Q pair. Due to the buffering by the FIFOs in the Tx transport DMA, it be directly connected to the Tx data interface to stream samples from the host processor. The following Verilog code provides an example on how make the direct connection:

```
// pop the next word data from the FIFO
assign tx_rd_en_out_<H>  = tx_dac_en_in_<H> & tx_rd_en_in_<H> & !tx_empty_<H>;

// assert the tx data interface data valid flag
assign tx_dac_en_out_<H> = tx_dac_en_in_<H> & tx_rd_en_in_<H> & !tx_empty_<H>;

// Connect the data buses.
assign tx_i_<H> = tx_din<H>[31:16];
assign tx_q_<H> = tx_din<H>[15:0];
```

Note: If timestamp mode is used, the PCIe interface will not drive tx_dac_en_in_<H> high until the proper timestamp is reached. In continuous mode, data is provided as soon as it is available in the FIFO. Continuous/timestamp mode is set by software.

2.3.2 User Register Interface Module

The reg_if.v and user_reg_if.v modules implement the software interface used to provide control and status of the user application. The memory space for the user application starts at byte address offset 0x8000. Addresses below 0x8000 will be visible on the bus, but should be ignored, as they deal with PCIe transactions beyond the scope of this document. The user application accesses the 32-bit registers at word boundaries. I.E. a byte address of 0x8000 is represented on the address bus as 0x2000.

Within the module, the memory space is split into banks containing 64 32-bit registers. This mapping results in the third nibble of the byte offset address identifying the bank(i.e. 0x8000 is bank zero, 0x8100 is bank one, 0x8200 is bank two, and so on). Banks 0x7-0xF are reserved for the user application space. Banks 0-6 are used by the system, and should not be used by the user application. Banks 0-6 are located in reg_if_auto_gen_bank_0, which is instantiated in the top level.

Four writable and six read-only registers are provided in the example design in the 0x8700 register space, and are not synced to any specific clock domain. They are driven by host_clk. The user may rename, add, or delete as desired to interface with user logic. If further register customization is desired, follow the template found in the source code. The provided read/write (software driven) registers and read-only registers are enumerated in Table 2.15.

Table 2.15: User Registers

Byte Address (hex)	Signal Name	Access
0x8700	reg_7_0	R/W
0x8704	reg_7_1	R/W
0x8708	reg_7_2	R/W
0x870C	reg_7_3	R/W
0x8710	reg_7_4	R
0x8714	reg_7_5	R
0x8718	reg_7_6	R
0x871C	reg_7_7	R
0x8720	reg_7_8	R
0x8724	reg_7_9	R

Bit zero of `0x8708` is used to illustrate how a timestamp reset can be driven up to the top level via custom logic with a user_app register bit in the user application. The following line of RTL is used for this in user_app.v:

```
assign timestamp_rst = user_reg_2_w[0];
```

This assignment can be safely removed if this feature is not desired.

Banks 8 and 9 are synced to sample_clk_a/b respectively via a FIFO. The write side of the FIFO is synchronous to `host_clk`, while the read side of the FIFO is synchronous to the respective sample_clk. A template for adding registers is provided within the comments of `user_reg_if.v`. The SidekiqX4 reference design does not have any register banks synchronous to `sample_clk_c` and `sample_clk_d`. To implement this, the user can modify the `user_reg_if` module clock port connections and to use the desired clock instead of `` sample_clk_a` and or ``sample_clk_b ``. If more than two domains are required, the user can add synchronization logic for additional clock domains using the bank 8 and 9 synchronization logic as a template.

Alternatively, the user may synchronize individual signals or buses to a specific clock domain using save clock domain crossing techniques.

3 Building and Debugging

The following sections describe how to build an FPGA targeting the SidekiqX2 and SidekiqX4 platforms and how to program and store them onto the respective platform hardware. The extracted project files have a hierarchy as shown in Fig. 3.1.

```
<platform>_<version>  Top level project folder (PROJ_ROOT)
├ constraints          Contains the constraint *.xdc files used to define pins, clock periods, and timing exceptions
├ docs                 Contains the project design documentation
├ hdl                  Contains the project RTL source
├ ip                   Contains any 3rd party IP configuration files (e.x. Xilinx *.xci)
├ include              Contains any Verilog include files
├ scripts              Contains helper build scripts sourced by the top level build script
├ ams_library          Top level directory containing the WILDSTAR libraries (AMS_LIB_ROOT). Not included. Only required for AMS builds. See the AMS build instructions.
│ └ vhdl-ws-ultra      directory extracted from the AMS WILDSTAR tarball
└ vivado_build.tcl     Top level build script
```

Fig. 3.1: Project Folder Hierarchy

3.1 Reference Design

The reference design can be built out of the box using the reference user application. For custom user applications, it is recommended that `user_app.v` is copied in its entirety for use as a model and starting point.

This section provides the steps to build the included design, as well as how to generate a custom design to work with the included build scripts. The bitstreams listed in Table 3.1 are the pre-built reference design images for the various platforms. A version identifier `<V>` is appended to the image's filename in the form `<git_hash>_<build_date>_<version>`. The bitstreams are located in the `sidekiq_image_current` directory after installing the SDK (see [3]).

Table 3.1: Pre-build Reference Design Images

Filename	Device	Description
sidekiq_x2_pdk_<V>.bin	XCKU060-2FFVA1517E	SidekiqX2 I/Q streaming reference design targeting the HiTech Global HTG-K800 carrier. (see [1])
sidekiq_x2_xcku115_pdk_<V>.bin	XCKU115-2FLVA1517E	SidekiqX2 I/Q streaming reference design targeting the HiTech Global HTG-K800 carrier. (see [1])
sidekiq_x2_ams_pdk_<V>.bin	XCKU060-2FFVA1517E	SidekiqX2 I/Q streaming reference design targeting the Annapolis Micro Systems WB3XZD carrier. (see [8])
sidekiq_x4_pdk_<V>.bin	XCKU060-2FFVA1517E	SidekiqX4 I/Q streaming reference design targeting the HiTech Global HTG-K800. (see [1])
sidekiq_x4_xcku115_pdk_<V>.bin	XCKU115-2FLVA1517E	SidekiqX4 I/Q streaming reference design targeting the HiTech Global HTG-K800. (see [1])
sidekiq_x4_ams_pdk_<V>.bin	XCKU060-2FFVA1517E	SidekiqX4 I/Q streaming reference design targeting the Annapolis Micro Systems. WB3XZD carrier. (see [8])

3.2 Custom User Applications

The SidekiqX2 and SidekiqX4 PDK reference designs should be used as the starting point for building custom FPGA bitstreams targeting the SidekiqX2 or SidekiqX4 respectively. It is strongly recommended to first build the design without modification to ensure that the build environment is suitable for generating valid bitstreams.

The majority of custom user applications should be developed by editing `user_app.v` and the modules within as described in Section 2.3. Most designs will not need to modify the `user_app.v` ports. If the top level ports are modified, `sidekiq_x2_top.v` (or `sidekiq_x4_top.v` for X4 designs) will need to be modified so the instantiation of the `user_app` module reflects the updates to the port map in user_app.v

Once the modifications to `user_app.v` have been made, the TCL build script `vivado_build.tcl` will need to be updated to add any new RTL source file, Xilinx IP, and constraint files.

3.3 Building the Project

Note: To build the SidekiqX2 and SidekiqX4 PDKs the full version (not webpack) of Vivado 2017.4 is required.

The project is built using the `vivado_build.tcl` script. `vivado_build.tcl` creates a Vivado in memory project, adds all of the project source and launches the various build steps (synthesis, route, place, ...). The build script will automatically set the build date parameter of the top level based on the system time of the build machine. If this project is built within a Git repository the build script will also retrieve the current Git hash and pass that in to the design via a top level parameter as well. The Git hash is stored with the format `0xCHHHHHHH` where `C` is a flag representing if the status of the repository, 0 = clean, 1 = dirty (modified files exist), and `HHHHHHH` is the 7 digit representation of the commit SHA. If Git is not used, the git hash embedded in the bitstream will be all zeros. The build date and Git hash of the FPGA image can be read from the software interface once the image is programmed onto the host FPGA.

FPGA builds targeting the Anapolois Micro Systems (AMS) carrier require installation of the AMS WILDSTAR libraries prior to running the build steps below. See Section 3.3.1 or Section 3.3.2.

The PDKs can be built using Vivado's batch or GUI modes using the following commands.

For GUI mode:

```
$ vivado -mode gui -source vivado_build.tcl
```

Caution: The build data and Git hash are determined when vivado_build.tcl is sourced. If Vivado GUI mode is used and the in-memory project is converted to a standard Vivado project, subsequent builds using the Vivado GUI will not update the build data and Git hash.

Caution: The build scripts contain a feature for a Vivado incremental compile flow. This is currently for Epiq use only, and is turned off by default. If you would like more information on how the incremental compile flow works, please consult the appropriate Xilinx documentation for an explanation of how to properly use this feature.

For batch (non-GUI) mode:

```
$ vivado -mode batch -source vivado_build.tcl
```

3.3.1 Annapolis Micro Systems Build Steps for the wszkvup_vpx3u_iope_dram platform

The SidekiqX2 and SidekiqX4 builds targeting the Annapolis Micro Systems (AMS) carrier require the AMS WILDSTAR libraries. The libraries should be extracted and built. Follow the steps below to configure and build the WILDSTAR library:

```
$ mkdir ams_library
$ cd ams_library
$ cp -rf /path_to_AMS_ISO/ISO.3.68.0 .
$ tar -xzvf ISO.3.68.0/WILDSTAR\ US\ VHDL/vhdl-wszkvup_vpx3u_iope_dram-1.50.0.tgz
$ cd vhdl-ws-ultra/wszkvup_vpx3u_iope_dram
$ export WS_ULTRA_BOARD_HOME=`pwd`
$ cd examples/iope/led_interface_example/synth
$ ./build.sh
    PART_TYPE?             1) xcku115-flvb2104-2-e
    WFMCP_SLOT0_INTERFACE? 1) NONE
    WFMCP_SLOT1_INTERFACE? 1) NONE
$ cd ../../../../../../../
```

Note: It is possible for the user to use a different directory name than `ams_library` and/or store it in a different location. If this is desirable, the user must set the `AMS_LIB_ROOT` environment variable using:

```
$ export AMS_LIB_ROOT=`pwd`
```

If `AMS_LIB_ROOT` is not set, the build script assumes the library is located in the projects root folder as shown in Fig. 3.1.

3.3.2 Annapolis Micro Systems Build Steps for the WB3XBM platform

The SidekiqX2 and SidekiqX4 builds targeting the Annapolis Micro Systems (AMS) carrier require the AMS WILDSTAR libraries. The libraries should be extracted and built. Follow the steps below to configure and build the WILDSTAR library:

```
$ mkdir ams_library
$ cd ams_library
$ cp -rf /path_to_AMS_ISO/ISO.3.110.0 .
$ tar -xzvf ISO.3.110.0/VHDL/vhdl-WB3XBM-1.38.0.tgz
$ tar -xzvf ISO.3.110.0/VHDL/vhdl-shared-1.38.0.tgz
$ cd vhdl-ws-ultrap/examples/WB3XBM/iope/led_interface_example/synth
$ export WS_ULTRA_BOARD_HOME=`pwd`
$ ./build.sh
    PART_TYPE?             1) xcku115-flvb2104-2-e
    WFMCP_SLOT0_INTERFACE? 1) NONE
    WFMCP_SLOT1_INTERFACE? 1) NONE
$ cd ../../../../../../../
```

Note: It is possible for the user to use a different directory name than `ams_library` and/or store it in a different location. If this is desirable, the user must set the `AMS_LIB_ROOT` environment variable using:

```
$ export AMS_LIB_ROOT=`pwd`
```

If `AMS_LIB_ROOT` is not set, the build script assumes the library is located in the projects root folder as shown in Fig. 3.1.

3.4 Programming the Flash

The generated binary files can be loaded onto the hardware using software utilities run from the host system. To reprogram the FPGA on X2 or X4, you must first store the FPGA to the flash and then initiate a reload from flash. This can be done with the store_user_fpga test application as follows with the store_user_fpga command (note that if you have another sidekiq card installed in your system, you might need to use -c 1 on the command line below if the X2 or X4 is considered the second card in your system). The following is an example for X2:

```
$ ./store_user_fpga -c 0 -s x2_top.bin --reload
Info: configuring for PCIE
Info: skiq_save_fpga_config_to_flash returned 0
Info: reloading FPGA from flash
```

Alternatively, the Sidekiq API skiq_save_fpga_config_to_flash() can be used to store the bitstream to flash. After that, a reload of the FPGA from the bitstream in flash can be initiated via the skiq_prog_from_from_flash() API.

3.5 Testing the Bitstream

The /test_apps/ directory contains several apps that can be scripted to run a regression test. As each user app may perform differently, the user may need to modify the source of each app to properly test their own custom bitstreams. See the SDK documentation [3] for descriptions of the provided test apps. Each app can be run with no parameters to view proper usage.

3.5.1 Using JTAG for Debug

A JTAG port is provided to facilitate debug for PDK customers. Xilinx's Chipscope application can be used to view internal FPGA signaling. Full Chipscope capabilities and use are beyond the scope of this document. For more information, see the Sidekiq X2 Hardware User's Manual [4] or the Sidekiq X4 Hardware User's Manual [6].

Caution: Xilinx programmers can be connected to the JTAG port, however the user should not use JTAG to program a bitstream. This will disrupt operation on the PCIe bus. Use the provided `prog_fpga` application to program new bitstreams as described in Section 3.4 to avoid misconfiguration of the PCIe interface.

4 Appendix: Top Level RTL Design Information

The information presented in this appendix is for users who need a more in depth understanding of the top level RTL. For example, someone who might be porting the design to a new FPGA platform or developing a custom carrier card. If you are solely using the FPGA PDK reference design without modification and making RTL changes within the user_app, most likely, you will not need to know the details presented in this appendix.

4.1 JESD RTL and Mapping

4.1.1 JESD RTL Hierarchy

The top level RTL contains a jesd_mapping module, as well as a jesd204_chA_support and jesd204_chB_support for Sidekiq X2 and a jesd204_chA_support, jesd204_chB_support, jesd204_chC_support, and jesd204_chD_support for Sidekiq X4. Each jesd204_chX_support module contains two JESD PHYs that interface to a single Tx/Rx JESD serial interface, as well as the JESD core logic needed to support those two PHY interfaces. The jesd_signal_mapping module converts the jesd data into i/q samples.

4.1.2 JESD I/Q Data Sample Clocks

For information about how the various clock trees are created for the i/q data samplec clocks, please see the section Section 2.3.1.1.

References

[1] HiTech Global. Hitech global's product page for the htg-k800 fpga pcie carrier card. URL: http://www.hitechglobal.com/Boards/Kintex-UltraScale.htm.

[2] Epiq Solutions Inc. Epiq solutions support website. URL: http://www.epiqsolutions.com/support.

[3] Epiq Solutions Inc. Sidekiq software development manual. URL: http://www.epiqsolutions.com/support.

[4] Epiq Solutions Inc. Sidekiq x2 hardware user's manual. URL: http://www.epiqsolutions.com/support.

[5] Epiq Solutions Inc. Sidekiq x2 product page. URL: http://www.epiqsolutions.com.

[6] Epiq Solutions Inc. Sidekiq x4 hardware user's manual. URL: http://www.epiqsolutions.com/support.

[7] Epiq Solutions Inc. Sidekiq x4 product page. URL: http://www.epiqsolutions.com.

[8] Annapolis Micro Systems. Annapolis micro systems' product page for the wb3xzd fpga pcie carrier card. URL: https://www.annapmicro.com/products/wildstar-ultrakvp-zp-dram-3u-openvpx/.

