Sidekiq Z2 and Matchstiq Z3u FPGA Development Manual

Version 3.16.2
Updated 03/10/22

EPIQ SOLUTIONS

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.

property of their respective owners.

Revision History

Date	Revision	Description
06/06/2018	3.9.0	Initial release.
09/10/2018	3.10.0	FPGA builds are now built as part of the top level project Linux build process. Add basic Tx capability.
12/12/2018	3.10.0	Document updates only. Update to section 8.2.1, added instructions on how to obtain the actual schematic in Vivado for Figure 5, and added info about user_app reg x8708. Update build instructions.
04/25/2019	3.12.0	Add frequency hopping control logic. Upgrade to Vivado 2018.2. Add iq swap mode.
09/09/2019	3.12.0	Doc update only, update info about fpga programming.
01/14/2021	3.14.1	Fix FIFO full write bug which resulted in the Rx header and sample data to be scrambled in certain circumstances. Fix bug where Rx would enter packed mode when not requested which resulted in the timestamp being off by 338 counts and the data scrambled as if it were in packed mode. Fix for starting/stopping streaming on a 1PPS edge.
04/12/2021	3.15.1	Z3u: Add GPS_CONTROL_MASK read only register. Z3u: Add gps_pps mux control. All: Fix missing timestamp reset related to clock crossing synchronization on the register interface. All: Add register reset capability. All: Add BASELINE_VCS_STATUS register. All: Add frc_sel_for_tx to be used in Tx timestamp mode.
10/29/2021	3.15.1 Updated 10/29/21	Documentation update only relating to fpga programming and packed mode not being available on the Z platforms.
03/10/2022	3.16.2	Z3u: Add board_id to the FPGA_REG_VERSION register for Zu3 RevD.

Table of Contents

Table of Figures

Table of Tables

1 About this Document

This document provides the necessary details for developing FPGA applications on the Sidekiq™ Z2 SDR or the Matchstiq Z3u SDR developed by Epiq Solutions [1]. It is provided with the purchase of a Sidekiq Z2 Platform Development Kit or a Matchstiq Z3u Platform Development Kit.

2 Legal Considerations

Sidekiq or Matchstiq radio cards are distributed all over the world. Each country has its own laws governing the reception and/or transmission of radio frequencies. The user of Sidekiq or Matchstiq radio cards 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 in using Sidekiq or Matchstiq radio cards and other Epiq Solutions' products.

3 Proper Care and Handling

Each 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 Sidekiq or Matchstiq 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 Sidekiq or Matchstiq from a host system when power is applied to the host system
- Connecting a transmitter to the RX port without proper attenuation – see the Specifications section for details on maximum RF signal input levels
- 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.

4 Introduction

The Sidekiq and Matchstiq Platform Development Kit (PDK) provides the ability for users to create their own custom applications. This can be accomplished by customizing software or the RTL code that configures the 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.

This document describes the functionality of the Sidekiq and Matchstiq PDK reference design that was/is designed, developed, and supported by Epiq Solutions. The standard Sidekiq Z2 Evaluation Kit (EVK) bitstream was/is designed, developed, and supported by Analog Devices' Industrial I/O (IIO) and their open source ecosystem. Documentation and support for the IIO design is not provided by Epiq Solutions. The Sidekiq Z2 PDK reference design and the Matchstiq Z3u reference design does, however, utilize certain critical components from the Analog Devices' IIO design, and details for building both the EVK (Sidekiq Z2 only) and PDK components to achieve a final bitstream is described in Section 8.

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 [2], which can be downloaded from the Epiq Solutions support website (http://www.epiqsolutions.com/support [3]).

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 Z2 Hardware User's Manual [4] or the Matchstiq Z3U Hardware User's Manual [6]. It is strongly recommended that the user read these documents thoroughly before attempting to dive into FPGA development.

This manual is meant to concisely describe the FPGA reference design, but it is important for even an experienced developer to spend time evaluating the actual design (i.e. RTL source code), perhaps even while digesting the information presented here. 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.

5 References

[1] **Epiq Solutions Website**
https://epiqsolutions.com

[2] **SDK Documentation**
Sidekiq_Software_Development_Manual_for_x.xx.x.pdf
Available at: https://epiqsolutions.com/support

[3] **Epiq Solutions Support Website**
https://epiqsolutions.com/support

[4] **Sidekiq Z2 Hardware User's Manual**
Sidekiq_Z2_Hardware_Users_Manual_vx.x.pdf
Available at: https://epiqsolutions.com/support

[5] **Analog Devices, Inc. IIO and Support**
www.analog.com

[6] **Matchstiq Z3u Hardware User Manual**
Epiq-Solutions-Matchstiq-Z3u-Hardware-User-Manual_vx.x.pdf
Available at: https://epiqsolutions.com/support

[7] **Sidekiq Z2 Getting Started Guide**
Z2 Getting Started Guide.pdf
https://epiqsolutions.com/support

6 Terms and Definitions

Term	Definition
ADC	Analog to Digital (A/D) converter
DAC	Digital to Analog (D/A) converter
DSP	Digital Signal Processing
EVK	Evaluation Kit
FIFO	First In First Out
FPGA	Field Programmable Gate Array
FRC	Free-running counter
IIO	Industrial I/O (Analog Devices, Inc.)
I/Q	In-Phase / Quadrature Phase
IP	Intellectual Property
MHz	Megahertz
PC	Personal Computer
PDK	Platform Development Kit
RF	Radio Frequency
Rx	Receive
SDR	Software Defined Radio
Tx	Transmit

Table 1: Terms and Definitions

7 FPGA Reference Design

7.1 Overview

The Sidekiq and Matchstiq PDK provides a complete FPGA reference design that enables a user to quickly and efficiently create custom applications with a Xilinx Zynq-7000 xc7z010clg225-2 device for the Z2 or with a Xilinx ZynqUs+ xc7z010-clg225-1 for the Z3U. On each platofrm, the IIO data interface is used to move data between the Zynq processor and the FPGA.

The unmodified reference design provides a full FPGA implementation to flow raw Rx I/Q samples from one ADC channel to the host Zynq processor and to transmit from the host Zynq processor to the RF chip via an FPGA channel of Tx data.

The PDK structure is created with the ease of the end-user in mind.

The Rx path transfers baseband I/Q samples which are received from an Analog Devices RFIC Transceiver. These samples pass through a DC offset correction block (which can be toggled on/off by software), then into a processing block which allows the user to process, timestamp, and transfer the samples via an IIO FIFO interface to the Zynq processor. In the reference design, the user application processing block can function as a simple pass-through which timestamps and drives the IIO FIFO with no changes to the samples. In this format, the 12-bit I and 12-bit Q components are sign-extended to 16 bits, for a 32 bit wide data bus. The “packed mode” (where 12 bit samples can be packed to fully utilize the 32 bit bus) available on some of the other platforms, is not currently available on the Sidekiq Z platforms.

On the Tx side, I/Q samples are transferred from the host system to the FPGA over IIO directly into the RFIC DAC interface. The user has the ability to process Tx if so desired. Similar to the receive side, transmit data can be pushed down in 16-bit sign extended mode or packed mode. Non-IQ data of an arbitrary format can also be sent down and processed in the FPGA for the user to process and transmit as desired.

In Figure 1, 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.

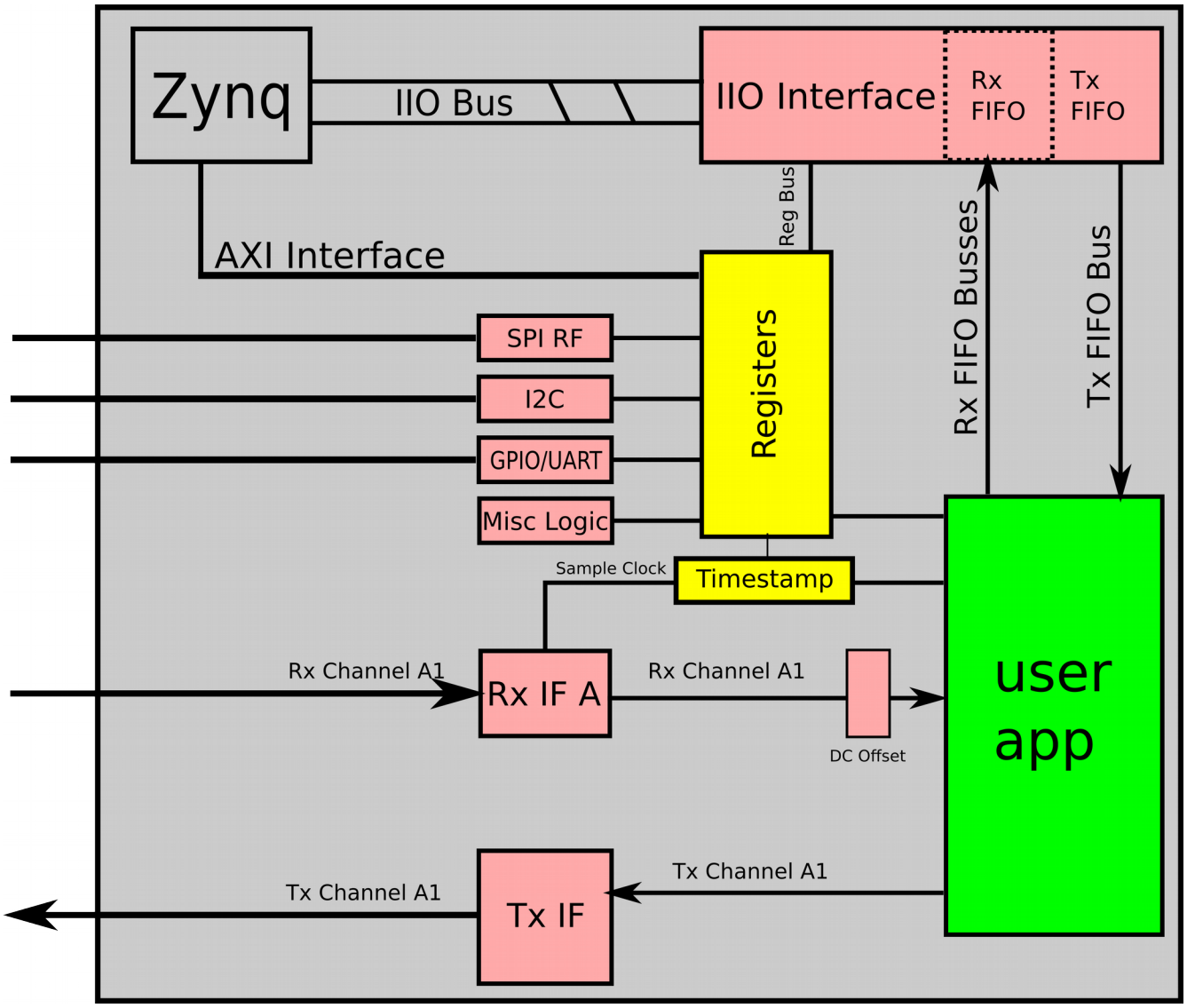

Figure 1: Sidekiq Z2 Simplified Block Diagram

7.2 Top Level

The Top Level block, sidekiq_z2_top (Sidekiq Z2 and Matchstiq Z3u share the some top level RTL file) is the top-level RTL and instantiates various blocks. 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.

7.3 user_app

The user app and register interface are Verilog files in which the majority of signal processing is expected to be done. The user_app interface is designed to allow reuse through multiple Epiq SDR platforms. 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 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 is built in its own directory so multiple apps can be built, tested, and version controlled simultaneously and independently. See Section 8.1.3 for more information on building custom apps. The user_app structure is designed to allow for portable signal processing blocks between multiple Epiq SDR platforms, including all Sidekiq and Matchstiq variants. This allows end users to share user_app modules between platforms and allow upgrades to future platforms with minimal rework required.

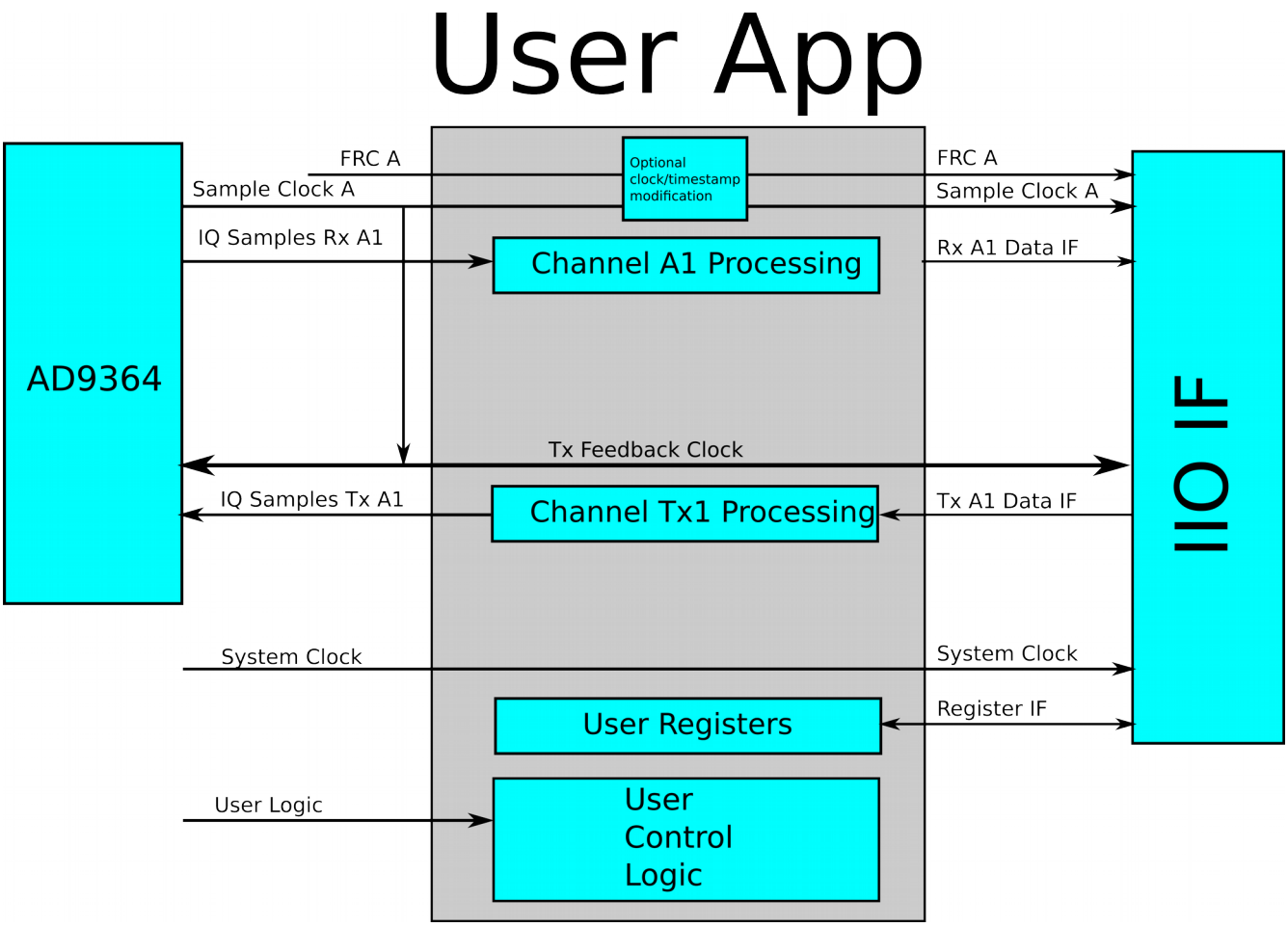

Figure 2: User App Block Diagram

7.3.1 user_app Signals

Several signals are provided to facilitate custom designs, some of which are not used in the stock user_app.

host_clock is the host system clock, which runs at 100 MHz.

clk_tx_fb is a feedback clock used to drive the transmit path. It runs at the same rate as the sample rate clock. It is discussed more in the user_app transmit section.

aux_clk is an always-on 100 MHz clock (assuming that the Zynq processor is up and running). It provides a clock source even if the RF chip is not running or no external reference is present. For Sidekiq Z2 and Matchstiq Z3u, this is the same clock as the host_clock.

ref_clk is an accurate 40MHz clock shared with the RF chip. If no external reference is present or if the RF chip is turned off, this clock will not be present.

external_enable can be used as an 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.

timestamp_rst used to reset timestamps and free-running counters. Timestamps can also be reset by software.

reg_rx_# is a per-channel control register that allows the user to monitor the state of the channel as controlled by software. Values of user interest are:

Bit # (indexed from 0)	Value : 0	Value : 1
1	IQ Data flowing	Test mode: counter data flowing
2	Rx Reset (No data should flow)	Rx Enabled
8	Tx continuous mode	Tx timestamp mode
9	Tx FIFOs enabled	Tx FIFOs reset
10	Passthrough mode	Packed Mode
11	DC Offset disabled	DC Offset enabled

Table 2: Rx Control Register

Bit 1 indicates if IQ data is flowing or if software has set test mode, which results in incrementing counter data being placed on the I and Q data bus.

Bit 2 is especially important – when low, the user should not push data to the IIO interface and should reset any data counters, as the IIO bus is inactive.

Bit 5 in reg_rx_a1 indicates if the RFIC is in single or dual channel mode. Only single channel mode is supported for Sidekiq Z2 while dual channel mode is supported for Matchstiq Z3u.

Bit 8 indicates if the transmit path is sending data as soon as it arrives, or if it is waiting to transmit data at a specific timestamp driven by software.

Bit 9 acts similar to bit 2, but on the transmit path. When asserted, the transmit FIFOs in the IIO interface block are in reset.

Bit 10 indicates if the channel is operating in passthrough mode, which sign-extends the 12 bit I and Q data to 16 bits each. If high, packed mode is enabled, which keeps I and Q 12 bits, and packs in the next portion of I Q data in. This requires software decoding when received by the host but allows for a roughly 20% increase in throughput.

Bit 11 indicates if software has enabled or disabled the DC Offset block, which acts on the IQ data before entering the user_app.

7.3.2 Rx Path Inputs to user_app

I/Q samples, valid signals, and clocks are routed in to the user_app and can be modified within to perform user-desired DSP. The input ports related to I/Q samples used are:

[11:0] i_samples_in_#, q_samples_in_#

sample_clk_a

sample_valid_a_#

full_#

Sidekiq Z2 and Matchstiq Z3u supports one I/Q channel. While sample_valid is high, each rising edge of the sample clock delivers a new sample consisting of twelve bits each of I and Q data. Each channel must be enabled by software before the sample_valid signal will go high.

The full flag is from the IIO interface and is used to indicate that, while sample data flowing in may be valid, there is no longer room in the IIO buffer to contain them. This may be due to incorrect IIO signaling or may indicate the total data throughput on all active channels exceeds the IIO data transfer rate.

Two counters are passed in, which are passed through to the register interface for use by software.

[63:0] frc_a_in, frc_sys_in

frc_a begins running when the channel associated with it is enabled. They continually increment as long as an associated clock is running. They can also be reset using a software-programmable PPS based reset or by user logic. On a sample rate change, this clock will temporarily be lost.

frc_sys is a system counter that runs whenever the 40MHz reference clock is present. It is 64 bits wide, and is normally driven by the 40 MHz reference clock to ensure accurate timekeeping. It can be reset using a software-programmable PPS based reset, or by user logic. This counter is continuous.

The following diagram illustrates the relationship between *sample_clk*, *samples_valid*, IQ data, and the free-running counter. Note that samples are valid only when samples_valid is high, but the counter increments as long as there is a valid clock.

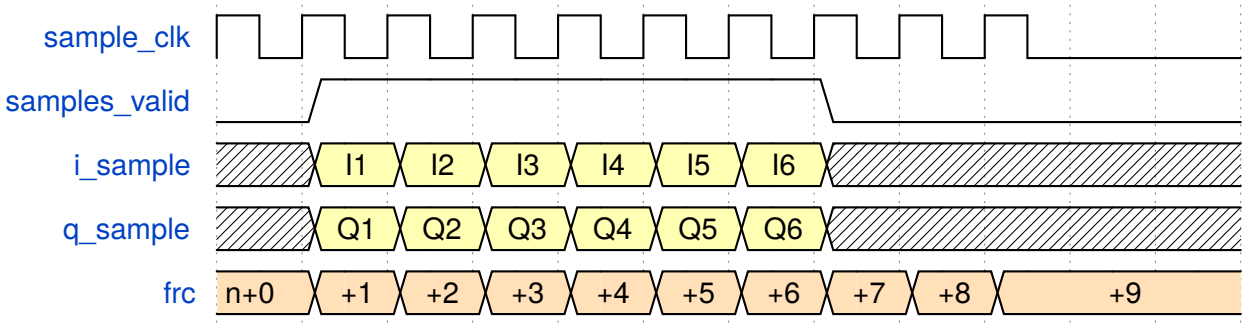

Figure 3: Sample Timing Diagram

7.3.3 Outputs from user_app

The user app drives data into a FIFO within the pcie_block NGC. Though the data can be thought of as a logical stream, each PCIe transfer consists of 1024 words of 32 bit length each. There are 1018 words of data and 6 words of metadata for 1024 total words. The first four words contain 64-bit timestamps based on *frc_a* and *frc_sys*. The fifth word contains channel and system control information. The sixth word is reserved for user definition.

Note that the PCIe bus works on 1018 data increments – until this size is reached in the FIFO, no data will be transmitted. If smaller data blocks are needed, the user must pad the remaining data words until the 1018 data length is reached.

The following signals are used to drive the PCIe FIFO:

[31:0] fifo_din_#

fifo_wren_#

sample_clk_a_out,

[63:0] frc_a_out, frc_sys_out

[31:0] user_metadata_#

fifo_din_# is a 32 bit wide data bus containing the data to be transferred to software. In the stock app, this contains unprocessed IQ data. Custom apps may process and place data in whatever format desired on this bus. The *fifo_wren_#* signal must be driven high to push *fifo_din_#* data into the IIO interface buffers. The write enable does not need to be continuous – it can be asserted and deasserted as necessary to push the proper processed data through. If tied high, data will continuously be streamed into the IIO interface

The channel A1 clock is driven by *sample_clk_a1_out*. This is normally tied to *sample_clk_a1*, but user applications may have other requirements.

If user blocks introduce a processing delay, the free-running counters can be modified to account for the delay. Alternatively, *frc_x_out* can be tied directly to *frc_x_in* and software can account for delays.

The *metadata_#* registers are optional. They contain up to 32 bits of user-supplied data which is embedded within each data block transferred over IIO. This allows users to transfer extra data with each set of 1018 samples. The metadata is latched in on each the first of the 1018 data words.

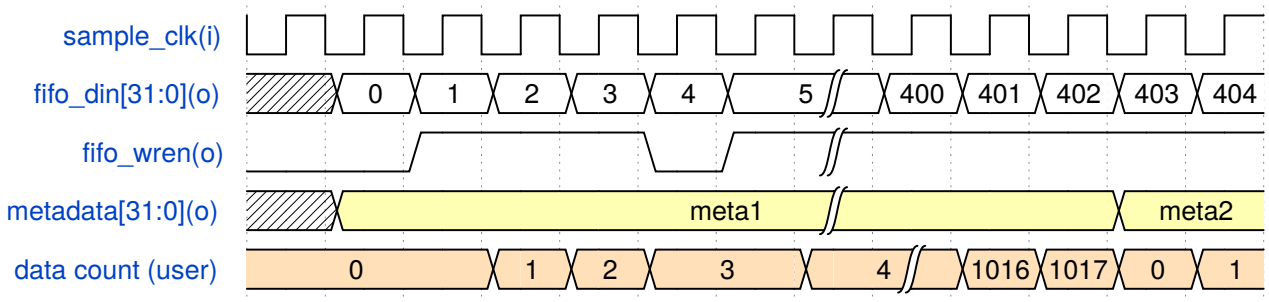

Figure 4: Sample User App to IIO Diagram

7.3.4 user_app Tx Interface

The Tx user interface allows software to pass down either IQ or other encoded data for processing to the FPGA to then be transmitted via the RF DAC interface. The DAC interface will be described first.

The interface to the RF chip is a 12 bit wide CMOS interface DDR bus, which means the clock driving it will run at the sample clock rate. On the receive side, the data_clk is also running at the sample clock rate. This is all transparent to user_app as it receives a sample_clock so each rising edge brings in a new IQ pair. On the transmit side, the transmit clock is the same rate as the receive sample clock rate, so each rising transmit clock will send a new IQ pair.

In passthrough mode, where each 32 bit data word *tx_din_#* coming from IIO contains a single I and Q pair and no additional data to be transmitted, *tx_dac_en_in/out* and *tx_rd_en_in/out* can be wired together to pass data straight from the IIO FIFO to the dac_if block.

The IIO interface can be modified if a user desires additional processing on transmit data. This interface pulls data from the IIO bus. To signal data is available, *tx_dac_en_in* goes high. On each *tx_rd_en_out* strobe, data is pulled from the IIO FIFO.

Note that if timestamp mode is used, the IIO interface will not drive *tx_dac_en_in* 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.

Several other signals are available to assist in debug.

tx_ts_# indicates the timestamp (on the frc_# domain) that the transmit path is waiting on.

tx_err_# indicates that a timestamp error has occurred, which is typically when data is not be able to be pushed down over the PCIe bus fast enough to transmit a packet at the correct time.

tx_empty_# indicates the PCIe FIFO is empty.

7.3.5 user_reg_if

Within the user_app, user_reg_if provides an address space to drive or read status of user logic. As the functionality of user_reg_if is nearly identical to reg_if, please see Section 7.4 for information. The code comments within user_reg_if serve to provide a template for adding user registers.

7.4 reg_if / user_reg_if

reg_if and user_reg_if are Verilog file that provides a register map for all FPGA functions, starting at address 0x8000 as referenced by software. Addresses below 0x8000 will be visible on the bus, but should be ignored, as they deal with AXI transactions beyond the scope of this document.

The R/W data bus is 32 bits wide, and 14 address lines are provided. The FPGA considers each 32 register a single address, while software addresses registers byte-wise. Logically, this results in the software address's two least significant bits being truncated off. The first software address, b1000 0000 0000 0000 (x8000), is seen as b10 0000 0000 0000 by the 14 address lines on the FPGA. The second address, b1000 0000 0000 0100 (x8004) is seen as b10 0000 0000 0001 by the FPGA, and so on.

Within the module, bits [9:6] on the address bus are used to denote the logical bank of the register, and bits [5:0] denote the address. This mapping results in the third nibble of the address from software being the bank(i.e. x8000 is bank zero, x8100 is bank two, x8300 is bank three, and so on).

Banks 0-6 are used by the system, and it is recommended they not be modified, though PDK customers do have the ability to do so. These banks are located in reg_if.v, which is instantiated in the top level.

Bank 0 is a general purpose bank, and does not contain any clock domain crossing logic.

Banks 1-4 provide system control over each Rx/Tx channel, and are synced to the sample clock on the channel domain.

Banks 5-6 are currently unused, but reserved.

Banks 7-9 are user register space, and are located with user_reg_if.v, which instantiated within user_app.

Eight read and four write registers are provided in the example design in the 8700 register space, and are not synced to any specific clock domain. They are driven by the PCIe clock. 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 (FPGA driven) registers are:

SW Address (hex)	FPGA Name	R / R/W
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

Table 3: User Registers

Please note that bit zero of x8708 is currently 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_app. The following line of RTL is used for this in user_app.v:

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

If you do not want this functionality and would like to be able to write to 0x8708 without it affecting the timestamps, replace the line of code in user_app.v with the following:

```
assign timestamp_rst = 1'b0;
```

Banks 8 is synced to sample_clk_a via a FIFO. The write side of the FIFO is driven by the AXI interface, and the read out from the FIFO is driven by the respective sample_clk. This gives a block of registers which can be used in sync-sensitive designs. It is recommended to only add registers and not modify the FIFO sync portion of the code. A template for adding registers is provided within the comments of user_reg_if.v.

If custom clock syncs are desired, it is suggested to use the 8700 register space, and sync the desired registers individually.

7.5 iio_data_if

Sidekiq Z2 and Matchstiq Z3u only supports one channel of Rx and one channel of Tx. Being able to adjust the size of the Tx FIFO is not required like it is on older platforms that use the PCIe interface. However, the RTL can be customized to remove the one Tx channel to free up some BRAM resources if needed.

The instantiation of the iio_data_if occurs at the top level in sidekiq_z2_top.v. To remove the Tx channel, simply edit this file to set the TX_CAPABLE parameter to 1'b0 (default is 1'b1) for this instantiation and then build the project.

7.6 timestamp_block

The timestamp_block.v is a Verilog file which handles driving and resetting the free-running counters (FRC) which serve to time stamp the samples. It also controls signaling to start and stop receipt/transmission of data based on a PPS signal.

Time syncing multiple units is handled primarily by software; see the Sidekiq Software Development Manual [2] for more information.

As it relates to the FPGA, the user_app can drive a reset, which will clear all free-running counters for as long as it is held high.

There are two counters present frc_a and frc_sys. frc_a is driven by sample_clk_a, and frc_sys is driven by the 40MHz reference clock. As long as reset is not held, these counters will increment as long as there is a clock present.

7.7 gpio/uart

Several pins are used as GPIO or UART signals. These are controlled by the Zynq processor. Some of these are MIO pins connecting directly to the PS, while others are EMIO that route through the PL to the PS. Signals associated with the latter case can be found as top level pins of sidekiq_z2_top.v that route to the Zynq system wrapper module.

Please see the Sidekiq Z2 Hardware User's Manual [4] and Matchstiq Z3u Hardware User's Manual [6] for more detailed information regarding these gpio and uart pins and functionality.

7.8 system_wrapper

7.8.1 Sidekiq Z2 system_wrapper

The system wrapper contains the Zynq processing system. This module is based on the open source Analog Devices Pluto Zynq processing system. It is created with the Vivado block designer. Changes from Pluto were made to support Sidekiq Z2. These changes include removing unnecessary modules, adding an axi_to_reg_if module to support Sidekiq Z2 registers, enabling UART0 functionality through the PL, and modifications to the adc_dma module to support packet streaming with timestamp capability. A block diagram of the Zynq processing system used by Sidekiq Z2 is shown below. This image is just to give a feel for the architecture of the system block design. For the actual schematic, open your vivado project in gui mode, and double click on the system.bd file in your design.

7.8.2 Matchstiq Z3u system_wrapper

The system_wrapper for Matchstiq Z3u is very similar to that of Z2. The main difference is that the linkage to pluto is not needed for Matchstiq Z3u. However. Two ADI IIO modules are instantiated inside the system wrapper. One for the ADC interface and one for the DAC interface.

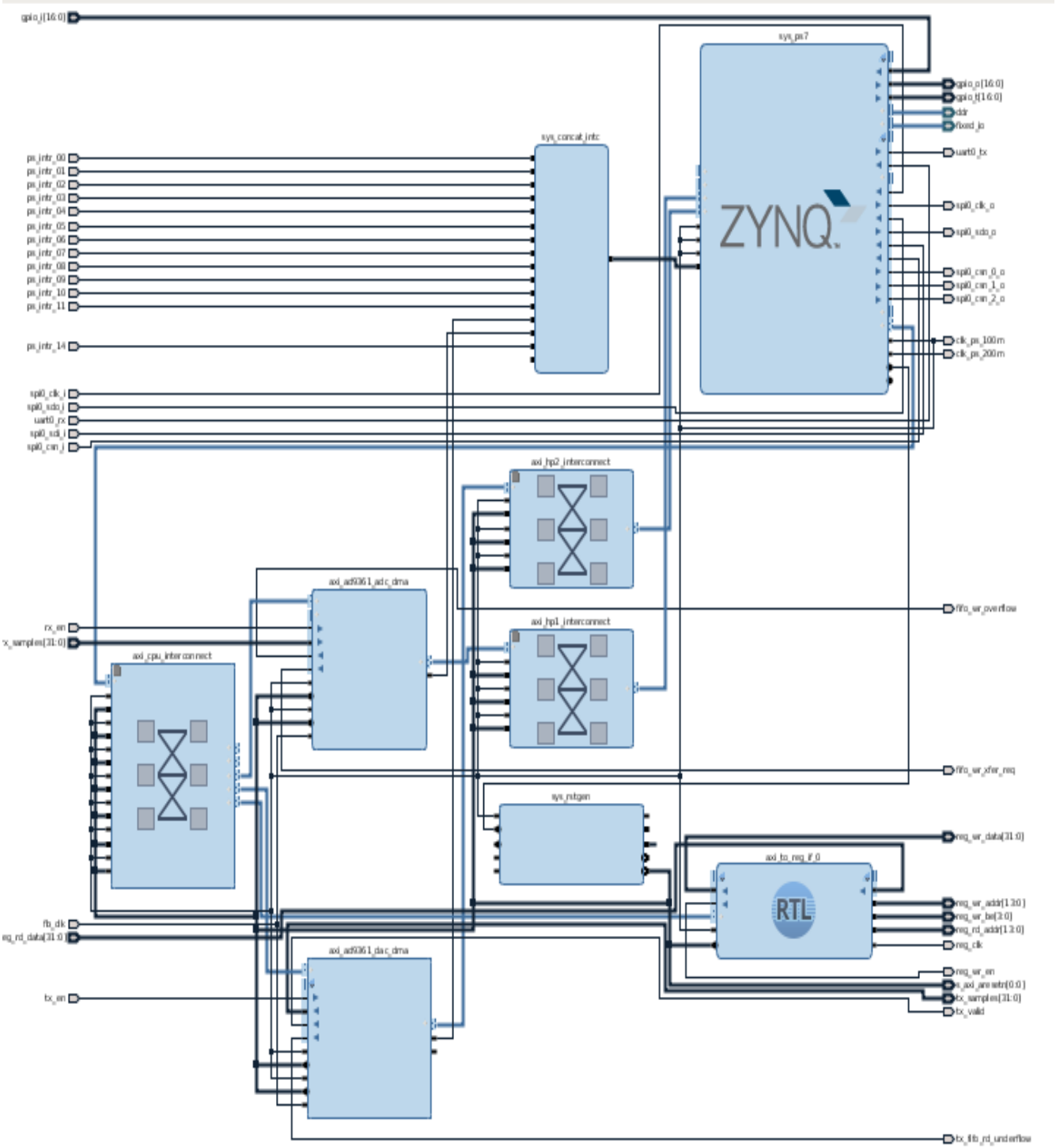
ZYNQ
RTL

Figure 5: Sidekiq Z2 Zynq Processing System

8 Building and Debugging

There are two steps to follow when developing custom designs for the Sidekiq/Matchstiq platform.

1. Create the user_app
2. Build the project and generate the bitstream

Each of these will be addressed in this section.

8.1 Building a user_app

The reference design can be built out of the box with a standard user_app named user_app because raw I/Q samples pass through unmodified before being sent over the IIO interface. Copy the reference design user_app as a model for building custom user apps, and rename the directory to an appropriate name. This section provides insight into building the included design, as well as how to generate a custom design to work with the included build scripts. The PDK user may customize the build flow to better match their existing processes if desired.

8.1.1 Sidekiq Z2 Reference Design

The user_app folder can be found in the plutosdr-fw/hdl/projects/sidekiqz2/epiq_hdl/user_app directory. This contains several files.

user_app.v contains the top level file. In this design, I/Q samples, clocks, and timestamp counters are passed straight through to the IIO FIFOs.

user_reg_if.v contains read and write user registers, as discussed in Table 3.

user_app_rx.v and user_app_tx.v contain the receive and transmit processing blocks. These can be instantiated per-channel, if different apps are desired on different channels.

user_reg_bank_# contains a generated bank of registers, which can be modifed to interact and control/read status of a custom user_app.

user_app_sidekiq.xdc is a constraint file in which custom constraints can be added. On Sidekiq Z2, the period constraint for sample_clk_a is defined by the data_clk constraint found in the user_app_sidekiq.xdc file within the user_app directory.

8.1.2 Matchstiq Z3u Reference Design

All of the RTL files can be found in the hdl directory.

user_app.v contains the top level file. In this design, I/Q samples, clocks, and timestamp counters are passed straight through to the IIO FIFOs.

hdl/auto_gen/user_reg_if.v contains read and write user registers, as discussed in Table 3.

user_app_rx.v and user_app_tx.v contain the receive and transmit processing blocks. These can be instantiated per-channel, if different apps are desired on different channels.

hdl/auto_gen/user_reg_bank_# contains a generated bank of registers, which can be modified to

interact and control/read status of a custom user_app.

constraints/user_app_sidekiq.xdc is a constraint file in which custom constraints can be added. On Matchstiq Z3u, the period constraints for the sample clocks are derived from the data_clk constraint found in user_app_sidekiq.xdc file.

8.1.3 Custom user_apps

To create a custom app, add your custom logic in user_app.v. The user has full use of Xilinx components, such as BRAM, DSP blocks, and PLLs. Most designs will not need to modify the user_app top level ports. If the top level ports are modified, plutosdr-fw/hdl/projects/sidekiqz2/system_top.v will need to be modified for Sidekiq Z2, or sidekiq_z2_top.v will need to be modified for Matchstiq Z3u. This is so that the instantiation of user_app.v will match the user's design.

If you have added any new Xilinx IP (*.xci files) to your design, then you will need to modify the build script plutosdr-fw/hdl/projects/sidekiqz2/system_project.tcl for Sidekiq Z2 or vivado_build.tcl for Matchstiq Z3u. You will need to add a command line to this file to include your new Xilinx IP to the project. See the add_files commands for the existing *.xci in the build script that applies to your platform (Sidekiq Z2 or Matchstiq Z3u). Use this as a template for the command line that you will need to add. If you do not include all necessary modules, the build will fail with an error indicating what needs to be added to the project.

Note that any new custom RTL files that you have will automatically get added to the project with the “add_files your_rtl_filename” command in one of the build scripts mentioned above. If you have placed your new RTL files in a directory other than one of the predefined PDK directories, then you will need to adjust your path in the instantiation command in the build script accordingly.

If your design includes custom clocking that creates a faster clock to process samples, it is likely you will need to add constraints to the user_app_sidekiq.xdc file to reflect your changes. The default sample clock constraint (as discussed previously in Section 8.1.1) is 62.5 MHz which is the max AD9364 rate. If the user design does not require these faster sample rates, it is recommended to reduce this constraint if you are having trouble with timing closure.

8.2 Building the project and bitstream

The Sidekiq and Matchstiq PDK reference designs should be used as the starting point for building custom FPGA bitstreams. It is strongly recommended to first build the design without modification to ensure that the build environment is suitable for generating valid bitstreams. Furthermore, a suitable version control system should be used to facilitate development. A git flow is already built into the included build script; if git is not used, the git hash embedded in the bitstream will be all zeros.

The Xilinx tools must be installed and properly configured in order generate a bitstream.

There are a couple of parameters that are passed into the top level RTL. The first is the the git hash; if git version control is used, the top 7 digits of the current git hash, along with a clean/dirty flag in the most significant nibble, will be placed into register address 0x8000 of the register map. The second is the build date. The current date will be placed into a register at address 0x8004.

The user can modify the build script to pass in additional parameters to configure their own designs if necessary.

8.3 Build with Linux

8.3.1 Building Sidekiq Z2

Note that the only supported Vivado version for Sidekiq Z2 is 2018.2.

The Sidekiq Z2 FPGA is now built as part of the top level project wide linux build process. Once you have the plutosdr-fw directory sitting in your current working directory, you can build the entire project with a top level make command. Note for PDK users that there is an additional step before making to integrate the Epiq RTL into the Pluto environment. This is currently done with a patch file. Please see the SidekiqZ2/System Releases and Updates Forum on Epiq Solutions Support. In this forum, you will find the latest build instructions in the Firmware Release thread. This includes how to download pluto, install the Epiq patch file, and set all necessary environment variables. Once all these directions have been followed, you can build the entire project with the following as per the Firmware Release thread:

```
$ make TARGET=sidekiqz2
```

Note that Xilinx Vivado 2018.2 is currently required to build the entire Sidekiq Z2 project.

For users who would like to use Vivado in GUI mode after the initial build has occurred, sidekiqz2.xpr will be generated as part of the build process. This will allow the project to be opened in the Vivado GUI mode by doing the following:

```
$ cd plutosdr-fw/hdl/projects/sidekiqz2
$ source /opt/Xilinx/Vivado/2018.2/settings64.sh

$ vivado -mode gui sidekiqz2.xpr
```

To implement any changes that you might have made to the RTL after the initial build, once the gui window has been invoked, simply hit the “Generate Bitstream” button on the left.

8.3.2 Building Matchstiq Z3u

Note that the only supported Vivado version for Matchstiq Z3u is 2018.3.

The Matchstiq Z3u Vivado build requires a custom board support file. Before building the FPGA for Matchstiq Z3u, the following commands need to be run one time or after installing a new Xilinx/Vivado install. Note that you may need to use sudo with these commands on your system. Also note that your Vivado install path might be different from what is shown below. Also note, that depending on your situation, you might need to use “sudo” with the below commands, particularly the cp and chmod commands.

tar -xvzf board_files.tar.gz

cd board_files

cp -rf sidekiqz3u /opt/Xilinx/Vivado/2018.3/data/boards/board_files/.

chmod -R 775 /opt/Xilinx/Vivado/2018.3/data/boards/board_files/sidekiqz3u

Now, to build the bitsream, simply run the following commands:

tar -xvzf sidekiq_z3u_pdk_vx_xx_x.tar.gz

source /opt/Xilinx/Vivado/2018.3/settings64.sh (note that your install path may be different)

vivado -mode batch -soruce vivado_build.tcl

8.4 Build with Windows

Windows is not supported as a development environment.

8.5 Programming

8.5.1 Programming the Sidekiq Z2 FPGA

The Zynq Programmable Logic (PL) can be programmed either using First Stage Boot-loader (FSBL), U-Boot or through Linux. To see Xilinx's recommended flow for programming from Linux, see the following link:
https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841645/Solution+Zynq+PL+Programming+With+FPGA+Manager

To see the various methods available for programming your custom bitstream, please also see the Sidekiq Software Development Manual [2].

8.5.2 Programming the Matchstiq Z3u FPGA

To see the various methods available for programming your custom bitstream, please see the Sidekiq Software Development Manual [2].

8.6 Testing the Bitstream

Test apps are provided with the SDK (Software Development Kit) bundle. See the Sidekiq Software Development Manual [2] for descriptions of the provided test apps. Each app can be run without any parameters to view proper usage and view the command line parameters that are available for each app.

8.7 Using JTAG for Debug

A JTAG port is provided to facilitate debug for PDK customers via the I/O Access/JTAG breakout Card. Xilinx's Chipscope application can be used to view internal FPGA signaling. Full Chipscope use is beyond the scope of this document. Both Xilinx and Digilent type programmers can be connected to the JTAG port on the I/O Access card. Please see the Sidekiq Z2 Hardware User's Manual [4] or the Matchstiq Z3u Hardware User's Manual [6] for more information on the I/O Access/JTAG breakout card and it's associated JTAG port.