Sidekiq Software Development Manual

Release 4.17.x

Epiq Solutions

May 09, 2023

Contents

1 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.

2 Document History

2.1 Document Revision History

Date	Revision	Description
04/28/2014	0.1	Initial version
05/01/2014	0.2	Updates after comments
05/27/2014	0.3	Updates for SDK v0.9
06/10/2014	0.4	Updates for SDK v0.10, system update 20140606
08/18/2014	0.5	Updates for SDK v0.14, system update 20140818
10/13/2014	0.6	Updates for SDK v1.1, system update 20141013
01/09/2015	0.7	Updates for SDK v1.6, system update 20150109
09/03/2015	1.0	Updates for SDK v2.0, system update 20150908
11/03/2015	1.1	Updates for SDK v2.3, system update 20151103
03/25/2016	1.2	Updates for Matchstiq S10 platform • Added Sections 10.5, 10.5.1, and 10.5.2 • Updates in Section 10.2 to indicate libc2.12 BUILD_CONFIG no longer required. • Updated for glib and deployment in Section 10.2.1
04/20/2016	1.2	Updates for SDK v3.0.0, system update v3.0.0 20160413
05/30/2016	1.3	Updates for SDK v3.0.3
06/20/2016	1.4	Updates for SDK v3.1.0
07/07/2016	1.5	Updates for SDK v3.2.0 • Migrated Appendix 6 to external document set
07/11/2016	1.5-develop	Adding information regarding RF IC control output in metadata
08/04/2016	1.5-develop	Updated channel bandwidth configuration
08/04/2016	1.5-develop	Add information regarding blocking receive feature
08/10/2016	1.6	Generic update for v3.3.0
09/27/2016	1.6-develop	RF port configuration update
09/29/2016	1.7	Miscellaneous updates for v3.4.0

Continued on next page

Table 2.1 – continued from previous page

Date	Revision	Description
01/05/2017	1.8	Updates for SDK v3.5.0 • Details on USB RX streaming (new transport) and FAQ • Details on logging function registration • Details on ref_clock test application and FAQ • Removed references to manual RFIC initialization as it now takes place during skiq_init() call. • Added to list of supported Linux kernels • Added reference to mxSidekiq, a MATLAB extension for controlling Sidekiq and transferring data with Sidekiq
05/08/2017	2.0	Updates for SDK v4.0.0 • Updates for libusb and deployment in Section 10.2.1 • skiq_init() and other general v4.0.0 updates • Updated section 9.10 for skiq_receive() and new skiq_rx_block_t type definition • Updated section 9.13 for skiq_transmit() and new skiq_tx_block_t type definition • Updated Tables 6 and 7 with new supported kernel versions which include 4.8 and 4.10 series kernels • Added accessing performance appendix
06/27/2017	2.1	Adding preliminary Sidekiq X2 details
09/22/2017	2.2	Updates for SDK v4.2.0 • Details on supported Sidekiq power consumption states in Appendix 9 • Updated Tables 6 and 7 with new supported kernel versions which include 4.11 and 4.12 series kernels • Updated Sidekiq X2 diagram
02/14/2018	2.3	Updates for SDK v4.4.0 • Added details on Sidekiq X2 Sample Rate / Bandwidth Settings • Updated Sidekiq mPCIe and m.2 Channel Bandwidth settings • Corrected Rx IQ Packet Structure in Figure 5 • Section 10.2.1: Added compiler flags for BUILD_CONFIGs: aarch64.gcc6.3 and arm_cortex-a9.gcc4.9.2_gnueabi • Added Section 9.10.3: Using receive calibration offsets • Updated supported kernel versions in Tables 7 and 8 • Added Section 10.6 Developing for the NVIDIA Jetson TX1/TX2 • Added Appendix 10 – Windows Sidekiq Development

Continued on next page

Table 2.1 – continued from previous page

Date	Revision	Description
06/13/2018	2.4-develop	Updates for SDK v4.6.0 • Fixed tables in Appendix 1 • Added more details on FPGA programming in Appendix 4 • Added Section 9.7: RF Port Configuration • Added Section 9.16.1 to describe new API functions for working with multiple handles when starting and stopping receive streaming • Added Section 9.19 on TX quadrature calibration details • Added Section 9.20 on Receive Stream Mode details • Added Section 10.7: Developing for the Sidekiq Z2 • Updated supported kernel versions in Tables 7 and 8. Added a lot more driver support for CentOS distributions. • Added introduction to Windows support in Appendix 10: Windows 10 is now supported. Sidekiq X2 is also supported. • Added details regarding new TX timestamps allowed late mode • Added appendix for detailed RF port configuration
06/15/2018	2.4	Finalize updates for SDK v4.6.0
09/21/2018	2.5-develop	Updates for SDK v4.7.0 • Added Balanced mode description to Section 9.20 • Added preliminary Sidekiq X4 details • Added details on 1PPS source selection • Added details on additional reference clock configuration • Updated X2 Architecture Diagram • Updated supported kernel versions in Tables 7 and 8 • Added section 9.10.4 on using I/Q phase and amplitude calibration
09/24/2018	2.5	Finalize updates for SDK v4.7.0
01/30/2019	2.6-develop	Updates for SDK v4.9.0 • Updated Sidekiq X4 Architecture Diagram • Added details on new Sidekiq X2 profiles • Added details on creating/loading runtime profile for Sidekiq X2 • Added RFIC control output details for Sidekiq X2 and Sidekiq X4 • Updated support kernel versions in Tables 7 and 8 - Added support for NVIDIA Jetson platforms (JetPack / L4T) • Updated Section 14.2 to include information about Annapolis Micro Systems' WILDSTAR FMC Carrier
02/05/2019	2.6	Finalize updates for SDK v4.9.0

Continued on next page

Table 2.1 – continued from previous page

Date	Revision	Description
07/25/2019	2.7	Updates for SDK v4.10.0 • Table 3: SDK Tarball Files, update to include tx_samples_from_FPGA_RAM • Add Section 9.8: include information for adjusting the ordering complex samples (I/Q) for both TX/RX • Section 9.12.1: Include text on special case for dual channel transmit and the TX FIFO size limit • Section 10.2.1: Add compiler flags for BUILD_CONFIG: arm_cortex-a9.gcc7.2.1_gnueabihf • Tables 7 and 8: update supported kernel versions • Section 11.1.3: Fixed typo in decimation factor example • Section 11.2.1.1: Add section to discuss Options for Sample Decimation (in X2 and X4) • Section 20.4: add subsections to discuss MinGW-64 versus Visual Studio usage • Table 14: RF Port Mapping, update Sidekiq X4 C1/D1 entries • Appendix 12 – FPGA user_app examples, added Section 22.1: Transmitting samples from FPGA memory • Appendix 13 – Release History, update release notes for v4.9.1 through v4.9.5 and v4.10.0 • Added Frequency Hopping section 9.5.1
10/15/2019	2.8	Updates for SDK v4.11.0 • Transition to using Sphinx for documentation, revision history references no longer apply • Add new product information for Sidekiq Stretch (M.2-2280)
02/03/2020	2.9	Updates for SDK v4.12.0 • Update supported kernel versions • Add subsection on “FPGA Configuration Flash Slots”
06/30/2020	2.10	Updates for SDK v4.13.0 • Update supported kernel versions • Add subsection on “Sidekiq X4 - Methods of LO frequency tuning” • Add note regarding Sidekiq X4 and TxA1/TxB1 transmit capability • Update section on “Dynamic Use of Sidekiq Cards” • Update code example for Frequency Hopping • Added RX Calibration to “Automatic Calibration” • Add FAQ regarding timestamp slips within products using the AD9361 RFIC

Continued on next page

Table 2.1 – continued from previous page

Date	Revision	Description
10/13/2020	2.11	Updates for SDK v4.14.0 • Update supported kernel versions • Add information on new exit handler in the "Exit" section of "Developing Apps" • Add "Hotplug" section to "Developing Apps" • Update "Clock and Time Management Resources" section of "Developing Apps" to expand on reference clock switching • Update information on sample rate/tuning function call order • Add Sidekiq X4 specific section "Pin Control enable of RFIC signal paths" • Update "Configuring Sample Rate / Channel Bandwidth" with more information on generating profiles for Sidekiq X4
2/3/2021	2.12	Updated for SDK v4.15.0 • Update supported kernel versions • Add "Advanced Topics" section to "Hosts & Platforms" • Update "Clock and Time Management Resources" section of "Developing Apps" to include information on the use of the GPSDO in Sidekiq Stretch (m.2-2280) • Add Matchstiq Z3u details • Add "Example X4 Use Cases" section to demonstrate typical Rx usage scenarios • Add "DKMS" section describing how DKMS is used with Sidekiq kernel modules • Update "Detailed RF Port Configuration" section • Update "Sidekiq X4 Built-in Profiles" section
5/28/2021	2.13	Updated for SDK v4.16.0 • Add "Using Libsidekiq Remotely" section describing the network transport (currently Z3u & Z2 only) • Add "Tx Timestamp Clock Source selection" section to "Developing Apps" • Update "Clock and Time Management Resources" section to include Matchstiq Z3u details • Update Sidekiq X4 diagram • Update "Number of Filter Taps" table for AD9631/4 parts in "Configuring Sample Rate / Channel Bandwidth" • Update code examples in "Developing Apps"
10/14/2021	2.14	Updated for SDK v4.17.0 • Add Sidekiq NV100 details • Correct filter line-up for last entry in the Decimation / Interpolation Factor table • Add "Analog Filtering" section describing how to override the analog filtering settings on AD936x-based products • Update supported kernel versions • Add `read_gpsdo.c` to the list of included example apps • Add paragraph in the "Clock and Time Management Resources" section describing how some Sidekiq products support changing the reference clock frequency at run-time with the libsidekiq API • Updated the network transport's unsupported API functions • Add SW4 and SW5 Errata

Continued on next page

Table 2.1 – continued from previous page

Date	Revision	Description
3/07/2022	2.15	Updated for SDK v4.17.2 • Update supported kernel versions • Add table listing NV100 supported Rx bandwidth percentages • Add note regarding NV100 sample rate dead-zones and a reference to the ADRV9001 User's Guide • Fix incorrect tx_hdl listed in the RF port mapping for NV100 • Update SW5 Errata to indicate resolution in FPGA v3.16.1 • Correct X4 Tx sample rates table, removing C1/D1 and add a note for configuring Rx handles at rates >= 200Msps

3 References

3.1 Links

[1] **Epiq Solutions Website**

https://www.epiqsolutions.com

[2] **Xilinx Website for the Spartan 6 FPGA**

http://www.xilinx.com/support/documentation/spartan-6.htm

[3] **GCC Website**

http://gcc.gnu.org

[4] **Cypress Website for FX2**

https://www.infineon.com/cms/en/product/universal-serial-bus-usb-power-delivery-controller/peripheral-controllers/ez-usb-fx2lp/

[5] **Epiq Solutions Support Website**

https://support.epiqsolutions.com

[6] **Sidekiq Hardware User's Manual**

https://support.epiqsolutions.com/viewforum.php?f=119

[7] **Sidekiq GNU Radio Support**

https://github.com/epiqsolutions/gr-sidekiq

[8] **Sidekiq System Updates**

https://support.epiqsolutions.com/viewforum.php?f=125

[9] **AD9361 Reference Manual UG-570**

http://www.analog.com/media/en/technical-documentation/user-guides/AD9361_Reference_Manual_UG-570.pdf

[10] **AD9371 User Guide (UG-992)**

https://form.analog.com/Form_Pages/Catalina/CatalinaDesign.aspx?prodid=AD9371

(Registration Required)

[11] **ADRV9008-1/ADRV9008-2/ADRV9009 Hardware Reference Manual (UG-1295)**

https://form.analog.com/Form_Pages/Catalina/CatalinaDesign.aspx?prodid=ADRV9009

(Registration Required)

[12] **Xilinx Website for the Artix 7 FPGA**

https://www.xilinx.com/products/silicon-devices/fpga/artix-7.html

[13] **Xilinx Website for the Kintex Ultrascale FPGA**

https://www.xilinx.com/products/silicon-devices/fpga/kintex-ultrascale.html

[14] **Annapolis Micro System WILDSTAR WB3XZD Baseboard** (Discontinued)

https://www.annapmicro.com/products/wildstar-ultrakvp-zp-dram-3u-openvpx

[15] **ADRV9001 System Development User Guide (UG-1828)**

https://www.analog.com/media/en/technical-documentation/user-guides/adrv9001-system-development-user-guide-ug-1828.pdf

4 Overview

4.1 Introduction

This document provides the details required to enable a software developer to develop Linux software applications that will utilize the Sidekiq SDR. The following topics will be discussed:

- Overview of the software architecture of Sidekiq
- Acquiring the prerequisites for developing software applications, including the GCC compiler, the libsidekiq Linux userspace library, and a text editor for source code editing
- Building and downloading the suite of test applications that are shipped with each Sidekiq PDK
- Development of custom applications that use libsidekiq
- Reference documentation for libsidekiq

All documentation and support for Sidekiq is provided through Epiq Solutions' support website, which can be found at:

https://support.epiqsolutions.com

Please note that it is necessary to register prior to accessing the relevant information for your purchase.

4.1.1 Legal Considerations

The Sidekiq is distributed all over the world. Each country has its own laws governing reception and transmission of radio frequencies. The user of the Sidekiq 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 transmission and reception of certain frequency bands, or receiving certain transmissions without proper authorization. Again, the user is solely responsible for the user's own actions.

4.1.2 Proper Care and Handling

The Sidekiq 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 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
- Connecting a transmitter to the RX port without proper attenuation – A max input of -10 dBm is recommended
- 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.

4.1.3 Terms and Definitions

Term	Definition
1PPS	1 Pulse Per Second
A/D	Analog to Digital converter
API	Application Program Interface
CPU	Central Processing Unit
D/A (DAC)	Digital to Analog converter
dB	Decibel
DKMS	Dynamic Kernel Module Support
DMA	Direct Memory Access
DSP	Digital Signal Processor
ERA	Epiq RF Analyzer
FDD	Frequency Division Duplex
FIFO	First In, First Out
FPGA	Field Programmable Gate Array
GCC	GNU Compiler Collection
GHz	Gigahertz
GPS	Global Positioning System
GPSDO	GPS Disciplined Oscillator
IP	Internet Protocol
I/Q	In-Phase / Quadrature Phase
kHz	Kilohertz
MHz	Megahertz
Msps	Mega samples per second
PC	Personal Computer
PCIe	Peripheral Component Interconnect express
PDK	Platform Development Kit
PL	Programmable Logic
PLL	Phase Locked Loop
ppm	Parts Per Million
PS	Processing System
RAM	Random Access Memory
RF	Radio Frequency
RF IC	Radio Frequency Integrated Circuit
Rx	Receive
SCP	Secure CoPy
SDK	Software Development Kit
SDR	Software Defined Radio
SRFS	System RF Server
SSH	Secure SHell
TCVCXO	Temperature Compensated Voltage Controlled Crystal Oscillator
TDD	Time Division Duplex
Tx	Transmit
USB	Universal Serial Bus

4.1.4 Overview

The Sidekiq SDR is a miniature software defined radio platform in the form of a standards compliant miniPCIe, m.2 (both 3042 and 2280), or FMC card. These form factors provide the potential for a wide variety of host systems with various processors. For clarity, the remainder of this document covers building applications for the Sidekiq PDK

system except when otherwise specified.

Software Architecture

A Sidekiq system consists of a Sidekiq miniPCIe, m.2, or FMC card inserted into a Linux or Windows host platform supporting the Sidekiq form factor. Additionally, the Sidekiq Z2 consists of a Zynq 7010, which is a self contained embedded ARM processor running Linux and Xilinx FPGA in the mPCIe form factor. The Matchstiq Z3u is a fully housed software defined radio consisting of the Zynq Ultrascale+, running Linux and Xilinx FPGA, a wideband transceiver, and an integrated GPS. The Sidekiq miniPCIe or m.2 card combines a Xilinx FPGA and Cypress USB microcontroller to provide a complete SDR system accessible to the host platform through PCIe and/or USB. The Cypress USB microcontroller is the FX2 *[4]* (page 8). For the miniPCIe variant of Sidekiq, the FPGA is a Xilinx Spartan 6 LX45T FPGA *[2]* (page 8). For both of the m.2 variants of Sidekiq (traditional, Stretch, and NV100), the FPGA is a Xilinx Artix 7 XC7A50T FPGA *[12]* (page 8). For the Sidekiq FMC card, referred to as Sidekiq X2 or Sidekiq X4, a FPGA carrier card in the form of the Kintex Ultrascale (KU060 and KU115) *[13]* (page 8) is installed in a Thunderbolt 3 chassis for a PDK is provided. This chassis uses the Thunderbolt 3 interface of a laptop to provide access to the Sidekiq FMC card. As of Q1 2019, the Sidekiq X2 and Sidekiq X4 are also supported in the Annapolis Micro Systems' WILDSTAR baseboard (WB3XZD) *[14]* (page 8) for use in an OpenVPX system.

A Sidekiq signal processing application consists of a userspace software application running on the host platform, which links against a userspace library called libsidekiq (as shown in following architecture figures). The libsidekiq library provides an API for configuring various RF and baseband parameters, access to various peripherals (temperature sensor, accelerometer, etc), as well as an interface for transferring data between the CPU and the FPGA. This data is typically digitized baseband I/Q samples. Full details of the libsidekiq library can be found in *Sidekiq API* (page 68).

4.2 Sidekiq mPCIe Block Diagram

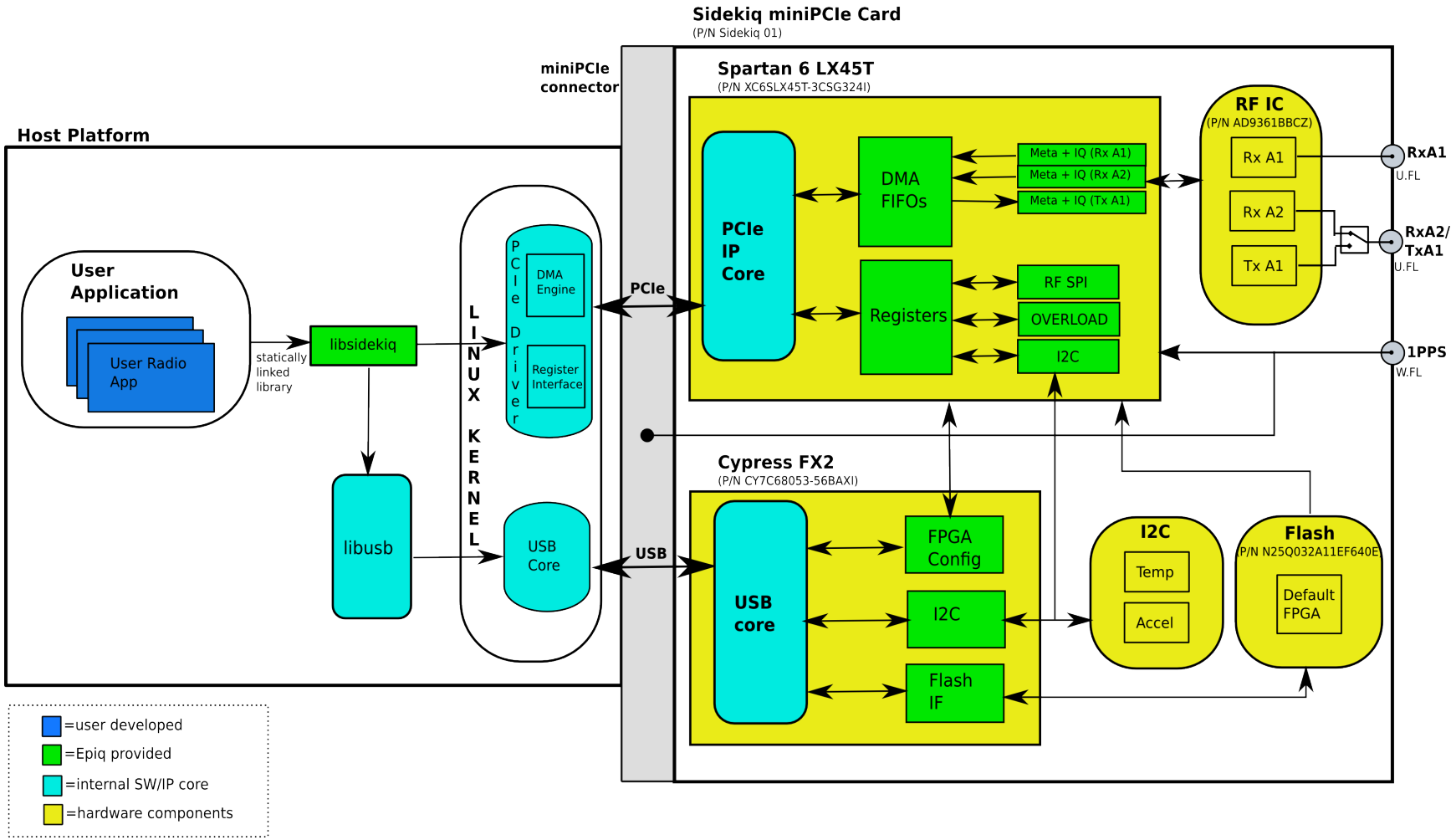

Fig. 4.1: Sidekiq mPCIe block diagram showing how libsidekiq + user applications fit in the system

4.3 Sidekiq m.2 Block Diagram

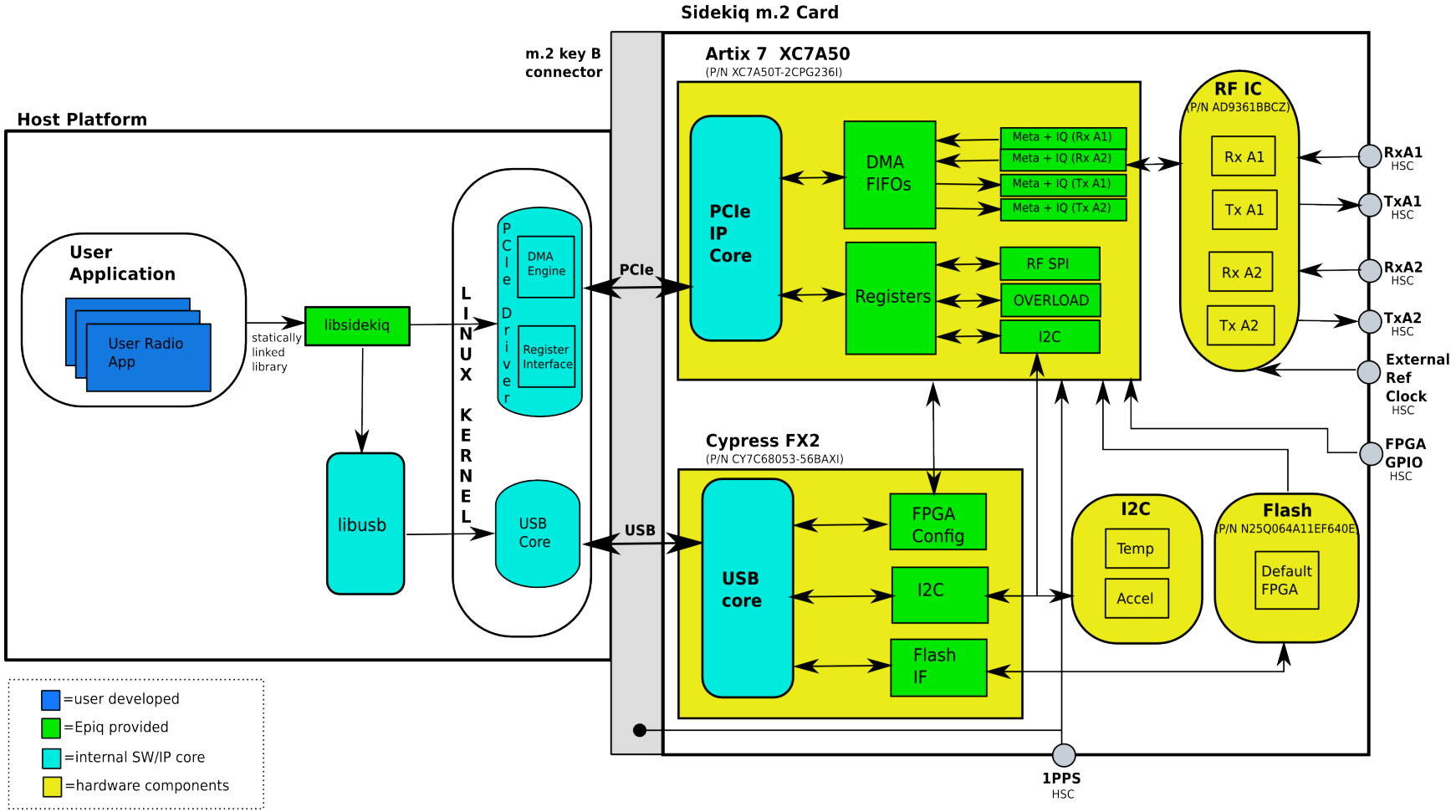

Fig. 4.2: Sidekiq m.2 block diagram showing how libsidekiq + user applications fit in the system

4.4 Sidekiq X2 Block Diagram

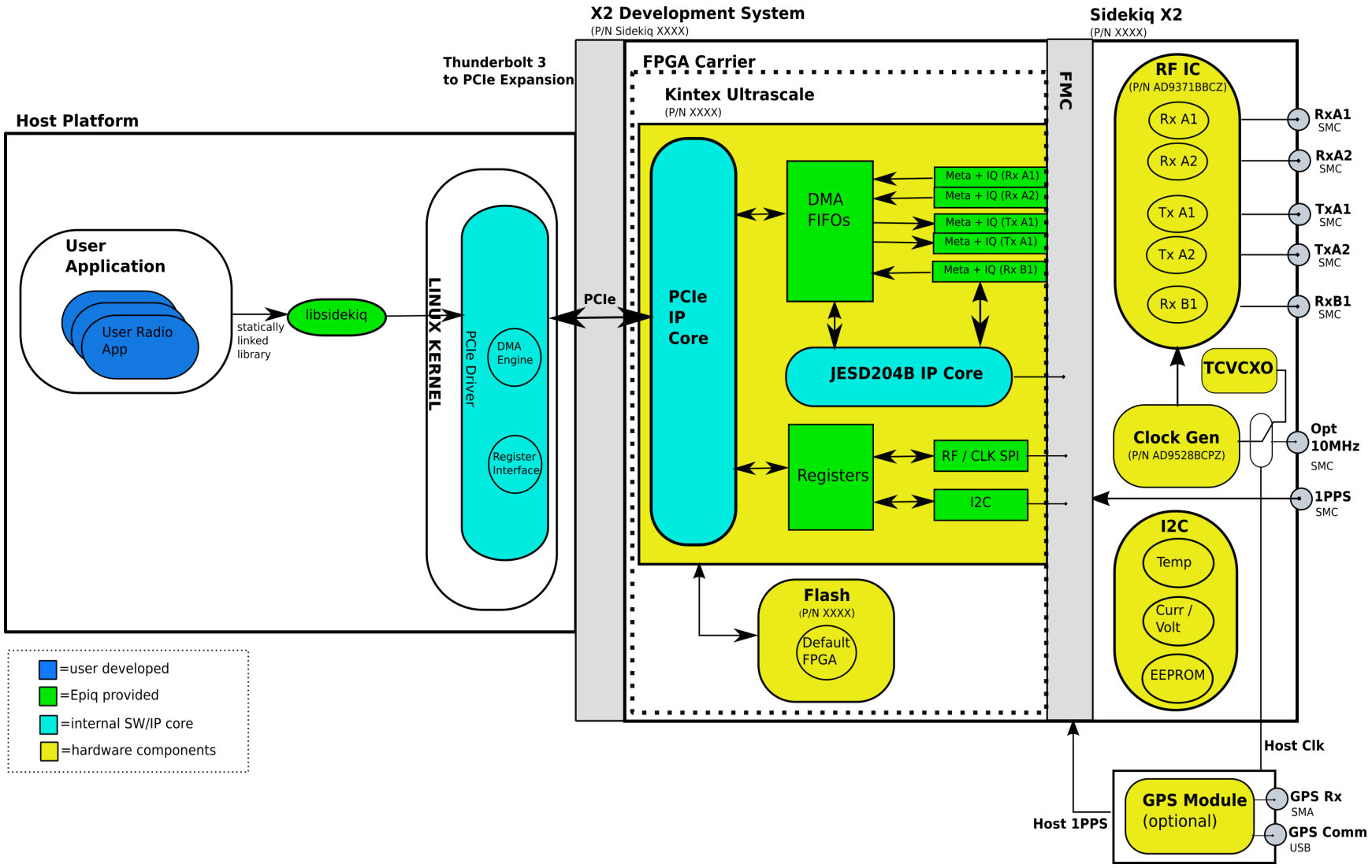

Fig. 4.3: Sidekiq X2 block diagram showing how libsidekiq + user applications fit in the system

4.5 Sidekiq Z2 Block Diagram

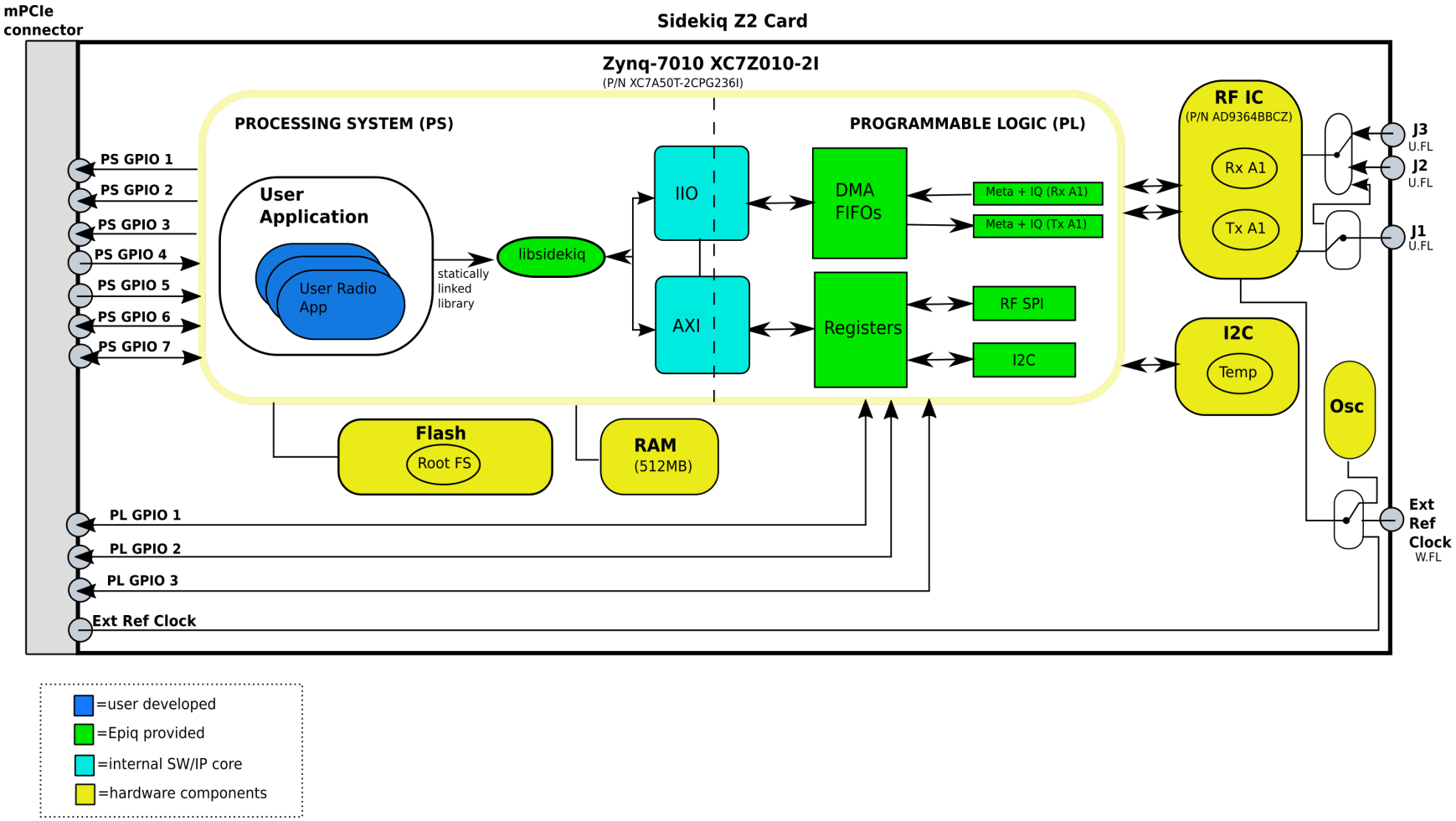

Fig. 4.4: Sidekiq Z2 block diagram showing how libsidekiq + user applications fit in the system

4.6 Sidekiq X4 Block Diagram

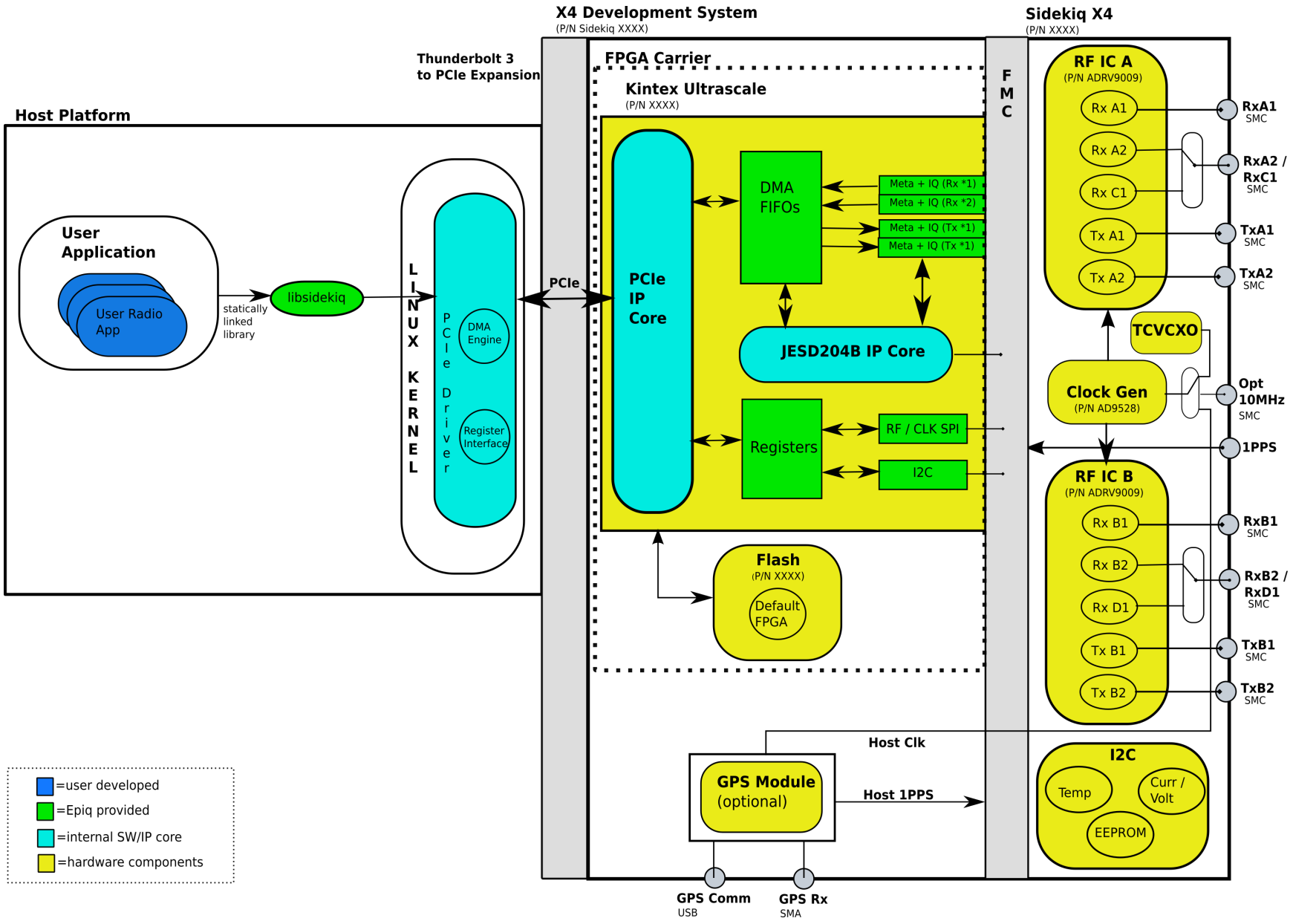

Fig. 4.5: Sidekiq X4 block diagram showing how libsidekiq + user applications fit in the system

4.7 Sidekiq Stretch (M.2-2280) Block Diagram

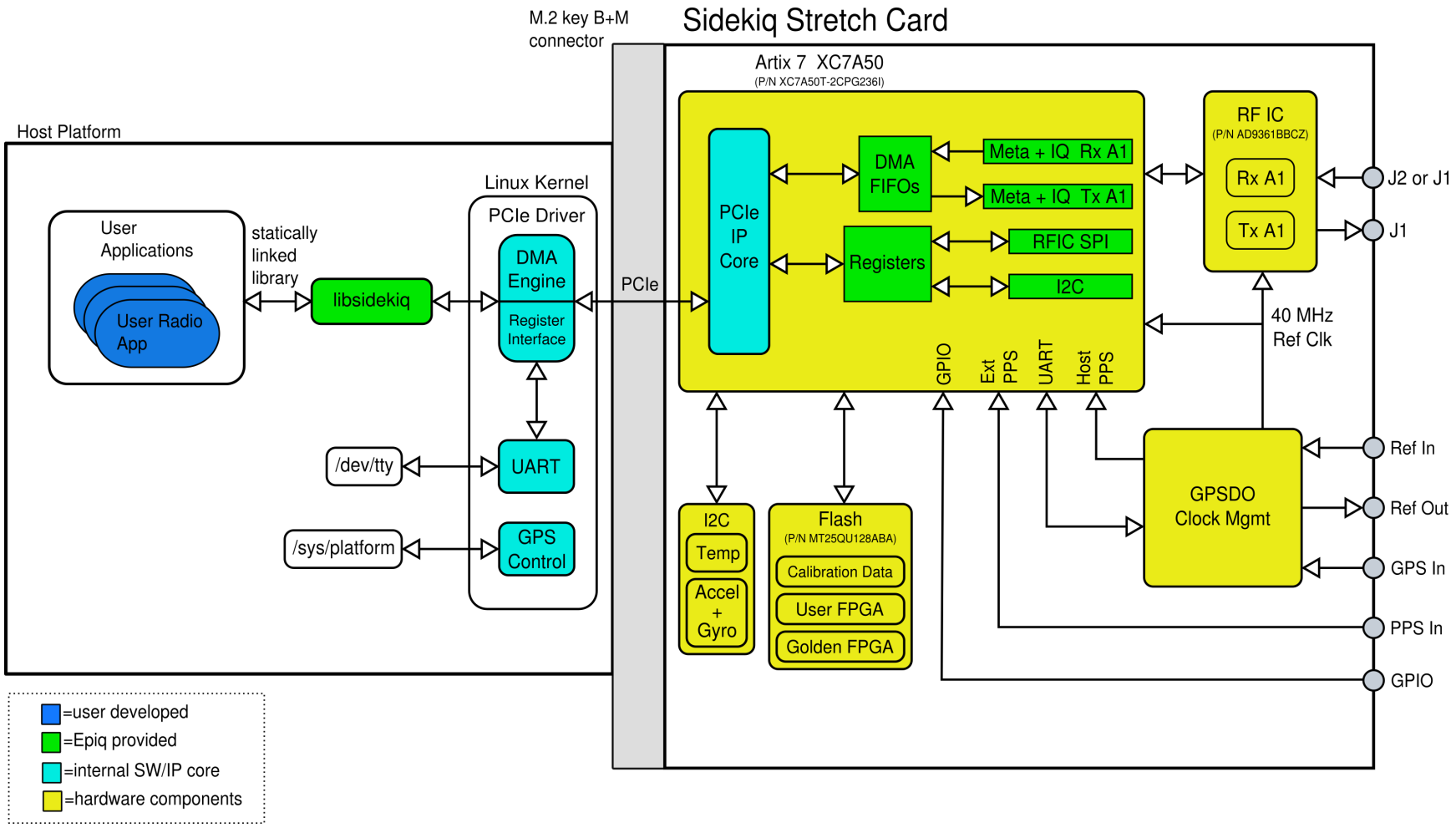

Fig. 4.6: Sidekiq Stretch block diagram showing how libsidekiq + user applications fit in the system

4.8 Matchstiq Z3u Block Diagram

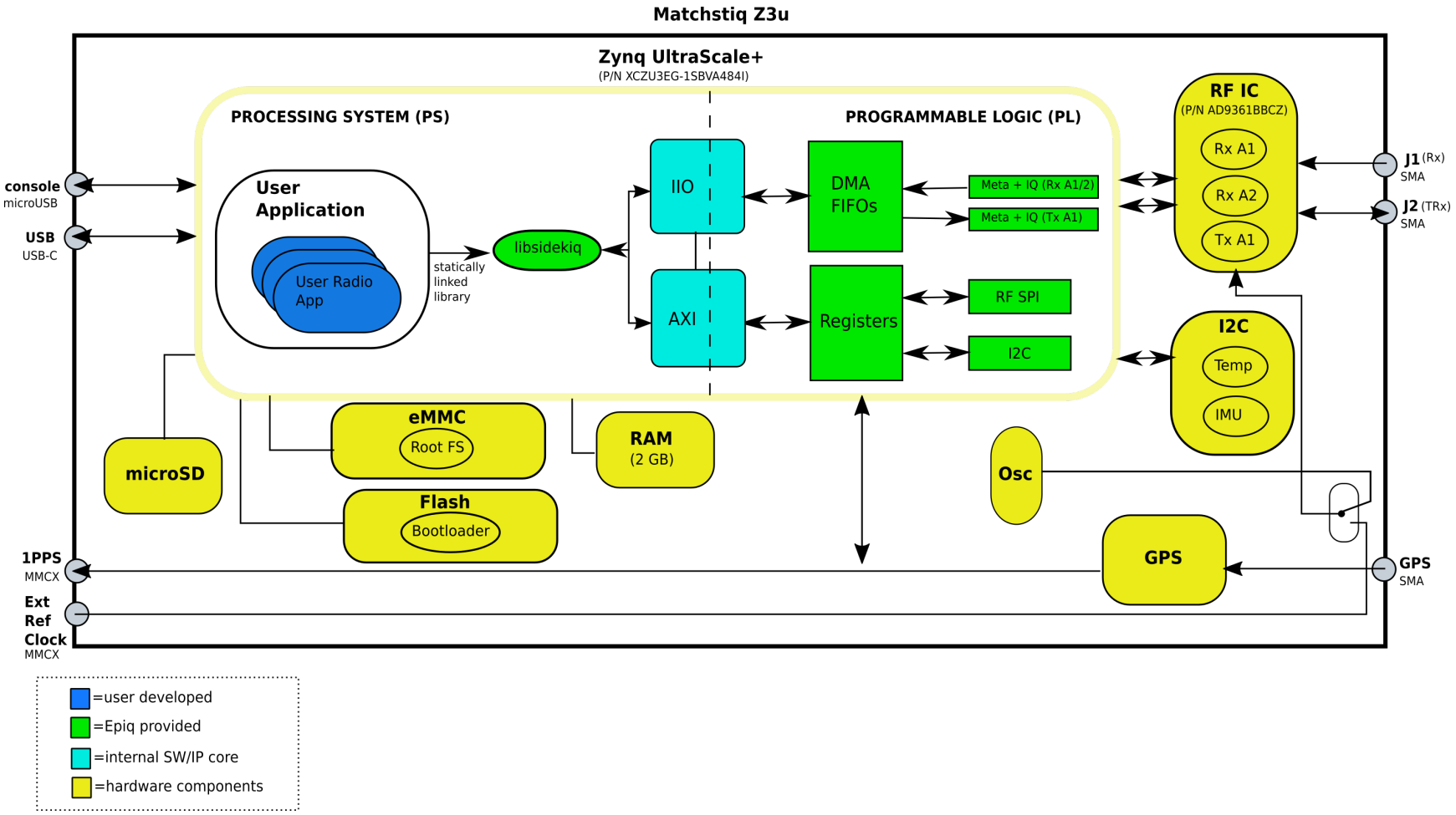

Fig. 4.7: Matchstiq Z3u block diagram showing how libsidekiq + user applications fit in the system

4.9 Sidekiq NV100 Block Diagram

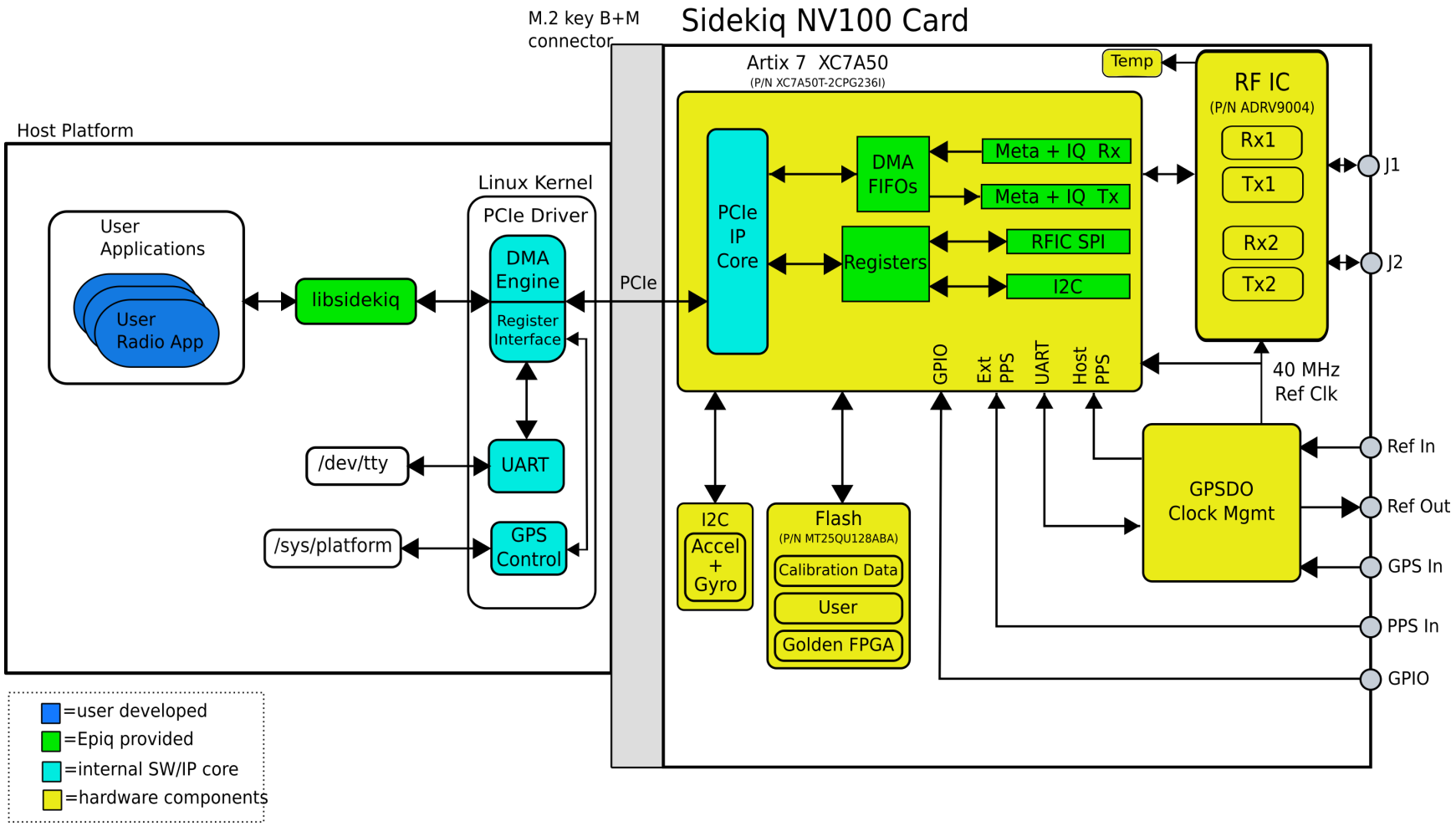

Fig. 4.8: Sidekiq NV100 block diagram showing how libsidekiq + user applications fit in the system

5 Developing with libsidekiq

5.1 Installation Procedure

The installation procedure installs both the SDK and the image directory to the install directory specified. The image directory contains: drivers, initialization scripts, and a link to prebuilt applications stored in the SDK. The SDK directory contents are depicted in the *Linux Sidekiq SDK Tarball Directories* (page 24) and *Linux Sidekiq SDK Tarball Files* (page 25) tables.

In addition to the image and SDK directories, updates are made to the Linux installation to automatically load necessary device driver for the Sidekiq card, support real-time priority threads by a Sidekiq user, and support the Sidekiq USB interface.

To initialize Sidekiq, a systemd-service or initscript (based upon the Linux distribution in use) is installed and enabled.

To support realtime priority threads by a Sidekiq user, a configuration file to allow realtime priorities is placed in `/etc/security/limits.d/99-sidekiq_limits.conf`.

To support a Sidekiq's USB interface, available on mPCIe and M.2-3042, a udev rules file is installed at `/etc/udev/rules.d/99-sidekiq.rules`.

As of libsidekiq v4.15.0, DKMS support is available for all Sidekiq kernel modules; see the *DKMS* (page 95) section for more information on what DKMS provides and how it may be used.

5.2 Software Development Flow

The software development flow for a Sidekiq system is similar to developing software applications for any Linux-based system. The general flow is described below.

Note: This development flow assumes that a user is developing software on the laptop delivered with the Sidekiq PDK

- Develop the source code for the custom software application on a PC such as the Sidekiq PDK laptop, using a text editor of choice (`emacs`, `vi`, `gedit`, etc.)
- Compile and link the source code that makes up the custom software application using the GCC toolchain. This includes linking the custom application against the appropriate libsidekiq userspace library that provides the API to access the RF transceiver hardware. For example, if a user is developing an application to execute on the Sidekiq PDK laptop, which has an Intel x86 CPU and a 64-bit Linux OS installed, then the `libsidekiq_x86_64.gcc.a` userspace library would be linked against. This process will output a binary executable file that will run on Linux on the Sidekiq PDK laptop. For details on compiling for alternative host platforms, refer to *Developing for Alternative Host Platforms* (page 80). For details on developing for Windows, refer to *Windows Sidekiq Development* (page 70).

- Execute the custom application on the Sidekiq system.

5.3 Tools/Libraries Needed for Linux Application Development

The following section provides an outline of all prerequisites needed to develop Linux-based software applications that execute on Sidekiq PDK laptop. For details on building on a Windows-based system, refer to *Windows Sidekiq Development* (page 70).

5.3.1 GCC Toolchain

The GNU Compiler Collection (GCC) toolchain *[3]* (page 8) provides a robust and widely used set of tools that can be used for building software applications for a variety of different target platforms. The GCC toolchain includes many different software components, including the appropriate C/C++ pre-processor, C/C++ compiler, linker, and other components needed as part of the building process.

The Sidekiq PDK laptop comes with the appropriate GCC toolchain pre-installed.

5.3.2 libsidekiq Userspace Library

The libsidekiq userspace library provides a high-level API for configuring the RF transceivers and transferring data between the CPU and the FPGA. This library is provided as a static library (e.g. `libsidekiq_x86_64.gcc.a`), such that a software application would link against this library as part of the software build process.

The current version of the libsidekiq userspace library can be downloaded at *[5]* (page 8) as part of the Sidekiq SDK. The complete documentation for the API provided by this library can be found as a separate document at *[5]* (page 8).

5.3.3 Source Code Editor

A standard text editor is used for editing the C/C++ source code that make up a Sidekiq software application. Most host PC Linux distributions ship with a variety of text editors already installed, including `emacs`, `vi`, `gedit`, and others; any text editor can be used for developing software applications for Sidekiq.

5.3.4 Re-Building the Sidekiq Test Applications

Each Sidekiq system ships with a suite of test applications installed in the `/home/sidekiq/sidekiq_image_current` directory (by default). These applications provide a means to demonstrate various features in the hardware, including receiving I/Q samples from the RF receivers, (re)programming the FPGA and reading sensors in the system such as the temperature sensor and accelerometer (if present).

A tarball containing the source code for the latest versions of these test applications, as well as the Makefile required to build them, can be found at Epiq Solutions' support website *[5]* (page 8). A Windows SDK for development under Windows is also available at Epiq Solutions' support website *[5]* (page 8). Assuming the GCC compiler has been downloaded and installed, the steps for re-building these applications for Linux are as follows:

1. Download the latest Sidekiq SDK tarball from *[5]* (page 8) to the Sidekiq PDK laptop. These tarballs have a version string in their filename. For example, version 4.17.x of the Sidekiq SDK tarball will be called:

`sidekiq_sdk_v4.17.x.tar.xz`

2. Untar the Sidekiq SDK tarball to a working directory on the Sidekiq PDK laptop. For example: if the development directory is located at `/home/sidekiq/`, the following command would be used to extract its contents:

```
$ tar xf /home/sidekiq/sidekiq_sdk_v4.17.x.tar.xz
```

3. Once the untar operation has been completed, the tarball's contents are available in the current working directory.
4. To re-build the test applications, change directories to the `test_apps` directory and run `make`.

```
$ cd /home/sidekiq/sidekiq_sdk_v4.17.x/test_apps
$ make BUILD_CONFIG=x86_64.gcc clean
$ make BUILD_CONFIG=x86_64.gcc
```

This will cause a complete re-build of all of the Sidekiq test applications, with the resultant binary executable files located in the `.../test_apps/bin/` directory. The applications built can be executed directly on the Sidekiq laptop just like any other Linux application. Most test applications can be executed without any command line arguments, and will report their expected usage. A user may use the *–help* command line argument to force the display of available arguments.

Note: The prebuilt applications and device drivers for alternative host systems (non-x86_64) can be obtained from *[8]* (page 8).

Table 5.1: Linux Sidekiq SDK Tarball Directories

```
sidekiq_sdk_vX.Y.Z
|-- arg_parser
| |-- inc
| `-- lib
|-- custom_xport_bare
| |-- bin
| |-- src
| `-- test_apps
|     `-- src
|-- doc
|   |-- api
|   |   `-- html
|   |-- manual
|   `-- sdk_manual
|       `-- html
|-- lib
|-- prebuilt_apps
|-- sidekiq_core
| `-- inc
`-- test_apps
 |-- bin
 `-- src
```

Table 5.2: Linux Sidekiq SDK Tarball Files

```
sidekiq_sdk_vX.Y.Z/arg_parser/
|-- inc
|   `-- arg_parser.h
`-- lib
    `-- arg_parser__<BUILD_CONFIGs>.a
```

```
sidekiq_sdk_vX.Y.Z/custom_xport_bare/
|-- Makefile
|-- src
| `-- my_custom_xport.c
|-- test_apps
| `-- src
|     `-- version_test.c
`-- tools.mk
```

```
sidekiq_sdk_vX.Y.Z/sidekiq_core/
`-- inc
 |-- sidekiq_api.h
 |-- sidekiq_params.h
 |-- sidekiq_types.h
 |-- sidekiq_xport_api.h
 `-- sidekiq_xport_types.h
```

```
sidekiq_sdk_vX.Y.Z/doc/
|-- api
|-- manuals
`-- sdk_manual
```

```
sidekiq_sdk_vX.Y.Z/lib/
|-- libsidekiq__<BUILD_CONFIGs>.a
`-- support
    `-- <BUILD_CONFIGs>/*
```

Continued on next page

Table 5.2 – continued from previous page

```
sidekiq_sdk_vX.Y.Z/prebuilt_apps/
`-- x86_64.gcc
 |-- app_src_file1
 |-- check_fpga_config
 |-- fdd_rx_tx_samples
 |-- multicard_dynamic_enable
 |-- multicard_rx_samples
 |-- multicard_tx_samples
 |-- prog_fpga
 |-- read_accel
 |-- read_gpsdo
 |-- read_modules
 |-- read_temp
 |-- ref_clock
 |-- rx_benchmark
 |-- rx_samples
 |-- rx_samples_freq_hopping
 |-- rx_samples_minimal
 |-- rx_samples_on_trigger
 |-- set_rx_LO_freq
 |-- sidekiq_probe
 |-- store_user_fpga
 |-- sweep_receive
 |-- tdd_rx_tx_samples
 |-- test_golden_present
 |-- test_sample_rate
 |-- tx_benchmark
 |-- tx_configure
 |-- tx_samples
 |-- tx_samples_async
 |-- tx_samples_from_FPGA_RAM
 |-- tx_samples_on_1pps
 |-- user_reg_test
 |-- version_test
 `-- xcv_benchmark
```

```
sidekiq_sdk_vX.Y.Z/test_apps/
|-- Makefile
|-- src
|   |-- app_src_file1.c
|   |-- elapsed.h
|   |-- fdd_rx_tx_samples.c
|   |-- multicard_dynamic_enable.c
|   |-- multicard_rx_samples.c
|   |-- multicard_tx_samples.c
|   |-- prog_fpga.c
|   |-- read_accel.c
|   |-- read_gpsdo.c
|   |-- read_imu.c (applicable to Z2 rev C)
|   |-- read_temp.c
|   |-- rx_benchmark.c
|   |-- rx_samples.c
|   |-- rx_samples_freq_hopping.c
|   |-- rx_samples_minimal.c
|   |-- rx_samples_on_trigger.c
|   |-- set_rx_LO_freq.c
|   |-- sidekiq_probe.c
|   |-- store_user_fpga.c
|   |-- sweep_receive.c
|   |-- tdd_rx_tx_samples.c
|   |-- test_golden_present.c
|   |-- tx_benchmark.c
|   |-- tx_configure.c
|   |-- tx_samples_async.c
|   |-- tx_samples.c
|   |-- tx_samples_from_FPGA_RAM.c
|   |-- tx_samples_on_1pps.c
|   |-- user_reg_test.c
|   |-- version_test.c
|   `-- xcv_benchmark.c
`-- tools.mk
```

5.4 Developing Custom Applications with libsidekiq

The Sidekiq SDK enables a user to develop their own software applications that utilizes a Linux userspace library called libsidekiq. This library provides a high-level API for configuring the RF transceiver, as well as transferring digitized baseband I/Q samples between the CPU and the FPGA.

5.4.1 Structure of an Application using libsidekiq

The `.../test_apps/src/` directory found in the Sidekiq SDK tarball (currently `sidekiq_sdk_v4.17.x.tar.xz`) contains the source code for several example applications that utilize libsidekiq. Applications utilizing libsidekiq should generally follow some basic guidelines to ensure they operate properly. These guidelines are outlined below.

The general structure of a custom application should follow the template laid out in the `.../sidekiq_sdk_v4.17.x/test_apps/` directory. This provides an example of a Makefile, as well as a directory to hold source files, header files (`inc`), and output binary executable files (`bin`). It is certainly possible to use a different directory/file structure here,

but the included Makefile is set up to support this structure already. In addition, the `app_src_file1.c` (located in `.../test_apps/src/`) source file contains the majority of the typical structure found in a Sidekiq radio application.

5.4.2 Proper Header File Inclusion

The entire libsidekiq API is exposed through a single header file called `sidekiq_api.h`. This header file can be found at:

`.../sidekiq_sdk_v4.17.x/sidekiq_core/inc/sidekiq_api.h`

Any application that utilizes libsidekiq will need to include the `sidekiq_api.h` header file in order to access the provided services. This is the only header file that is required to be included.

There are four support header files, three of which are included indirectly by `sidekiq_api.h`. These are `sidekiq_types.h`, `sidekiq_params.h`, and `sidekiq_xport_types.h`. The fourth support header file is `sidekiq_xport_api.h` and is used by custom transport developers when implementing registration of a custom transport. This header file is found in the same location as `sidekiq_api.h`:

`.../sidekiq_sdk_v4.17.x/sidekiq_core/inc/sidekiq_xport_api.h`

5.4.3 Initializing libsidekiq

The libsidekiq library supports use of one or more Sidekiq cards in a single host system. Each Sidekiq card can utilize either the PCIe interface, the USB interface, or a custom defined interface for transport between the host platform and the Sidekiq card. Automatic detection of the interface is supported. The `skiq_init()` function is used to initialize the RFIC, initialize the libsidekiq library, reserve the resources required for the specified Sidekiq card(s), and initialize the communication transport. Inserting the kernel modules are required prior to accessing a Sidekiq card when using a PCIe transport. This is handled automatically on the PDK laptop and is performed by the initialization script described in *Installation Procedure* (page 22).

If it is desired to query how many Sidekiq cards are detected by the host platform and the mapping of Sidekiq card (based on serial number) to card number in the system prior to reserving the cards, the following sequence of libsidekiq calls may be made:

```
uint8_t number_of_cards = 0;
uint8_t card_list[SKIQ_MAX_NUM_CARDS];
uint8_t card_num = 0;
uint8_t i = 0;
char *serial_str;

/* query the list of all Sidekiq cards on the PCIe interface */
skiq_get_cards( skiq_xport_type_pcie, &number_of_cards, card_list );

for (i = 0; i < number_of_cards; i++)
{
    /* determine the serial number based on the card number */
    skiq_read_serial_string( card_list[i], &serial_str );
    printf("Sidekiq card number %u has serial number %s\n", card_list[i], serial_str);

    /* determine the card number based on the serial number */
    skiq_get_card_from_serial_string( serial_str, &card_num );
    printf("Sidekiq serial number %s is located at card number %u\n",
           serial_str, card_num);
}
```

A card can only be used by a single application at any point in time. To determine a card's availability, the `skiq_is_card_avail()` API should be used. If the card is already in use, the process ID of the application using

the card is provided. When `skiq_init()` is called for a specific card, the card is reserved for use by that application and other applications are denied access to the locked card.

To perform the initialization of libsidekiq, the `skiq_init()` function should be called specifying the cards to use and the initialization level.

Note: As of libsidekiq v4.8.0, specification of the transport interface is deprecated and that `skiq_xport_type_auto` is always used regardless of what is specified. The example below illustrates how to initialize two cards for full RF functionality.

```
uint8_t num_cards = 2;
uint8_t cards[num_cards] = {0, 1};
skiq_xport_type_t type = skiq_xport_type_auto;
skiq_xport_init_level_t level = skiq_xport_init_level_full;

/* initialize libsidekiq for card numbers 0 and 1 */
skiq_init( type, level, cards, num_cards );
```

The initialization level specifies how much initialization should be performed for each of the interfaces. Any application interested in streaming sample data or configuring the RF radio properties should perform a full initialization by specifying the level as `skiq_xport_init_level_full`.

The `skiq_init()` function also initializes three mutexes per Sidekiq card provided internally by libsidekiq: one mutex is used to control access to FPGA registers, another mutex is used to protect access to the FPGA $\rightarrow$ CPU interface used for receiving I/Q samples, and a third mutex is used to protect access to the CPU $\rightarrow$ FPGA interface used for transmitting I/Q samples. This allows a multi-threaded host application to access the various libsidekiq services, while ensuring that calls to libsidekiq are thread-safe.

Note: Running multiple applications that utilize libsidekiq is supported as long as each instance of libsidekiq is using a different Sidekiq card. Applications using the same Sidekiq card cannot be executed at the same time.

Lastly, as part of the initialization process, a signal handler should be installed for capturing `SIGINT` signals that may occur. Graceful shutdown should be performed, including stopping any active streaming and calling `skiq_exit()`. Note that the stop streaming and exit calls should not be made within the context of the signal handler as those calls attempt to lock a mutex which may already be locked if a receive or transmit call is in progress prior to the handling of the signal. A recommended approach to handle the signal is to set a flag which the main application checks to see if it should continue to run. Upon the clearing of the "run flag", the application should cleanup and call `skiq_exit()`. See the `tx_samples.c` test application for an example of this.

Dynamic Use of Sidekiq Cards

libsidekiq v4.9.0 adds the ability for an application to dynamically enable or disable use of Sidekiq cards independent of the library initialization performed via the `skiq_init()` API function. The `skiq_init_without_cards()` API function may be used to initialize the library without reserving any card for immediate use; this is the equivalent of calling `skiq_init()` with an empty card list.

At a later point during the application's execution, a card can be initialized and enabled for use through either the `skiq_enable_cards()` or the `skiq_enable_cards_by_serial_str()` API function. This will perform a full initialization of the card and reserve the card's use to the requesting application. At a later point in time, if the card is no longer needed by the application, the card can be released using the `skiq_disable_card()` API. This shuts down the card, and this card is the considered available for use by either the same or another application. Calling `skiq_exit()` performs a full shutdown of the library as well as any card currently reserved by the application. There is no need to perform an additional call to `skiq_disable_card()` prior to calling `skiq_exit()`.

A simple example (without error checking) of accessing cards dynamically could be:

```
uint8_t num_cards = 0;
uint8_t cardList[SKIQ_MAX_NUM_CARDS] = { 0 };

/* Initialize libsidekiq with no cards - none are needed right now */
skiq_init_without_cards();

/* Do some tasks */
...

/* A Sidekiq card is now needed; dynamically enable card 0 */
cardList[0] = 0;
num_cards = 1;
skiq_enable_cards(cardList, num_cards, skiq_xport_init_level_full);

/* Perform some radio tasks */
...

/* Done with the card; dynamically disable card 0 */
cardList[0] = 0;
num_cards = 1;
skiq_disable_cards(cardList, num_cards);

/* Perform some cleanup tasks */
...

/* Shutdown libsidekiq */
skiq_exit();
```

The test application `multicard_dynamic_enable.c` provides a basic example of dynamically enabling or disabling a card for use by the application.

Logging

By default, libsidekiq executes any logging of the library with syslog. The default configuration is to display any logging message to both the console as well as within `/var/log/syslog`. A user can utilize their own logging function by registering their own logging function pointer with the `skiq_register_logging()` API function. Refer to the `app_src_file1.c` test application for an example of this. If the user is interested in completely disabling any logging or prints within the Sidekiq library, `NULL` can be provided to `skiq_register_logging()`.

Transport Layer

Within libsidekiq, there is the concept of a "transport" (or xport) layer. This layer facilitates I/O between a Sidekiq card and host, such as configuring radio parameters via register transactions and streaming I/Q samples. Sidekiq's hardware by default is configured to implement PCIe and requires only that software be initialized in the manner as described above. The Sidekiq mPCIe and m.2 radios additionally have an available USB transport; making use of this transport is only recommended for use in systems where PCIe is not feasible, as the USB transport is significantly less performant when streaming samples and transacting configuration registers. Both Sidekiq Z2 and Matchstiq Z3u utilize a custom transport and the transport-specific FPGA bitstream is automatically loaded upon boot for these radios. For details on configuring the FPGA, refer to *FPGA Programming* (page 102).

Note: Sidekiq X2, X4, Stretch, and NV100 do not support the USB transport layer.

Parameters

Depending on the Sidekiq product and configuration, availability of certain hardware peripherals may be limited. Additionally, configuration of various RF parameters such as frequency, sample rate, and gain, vary. To determine the parameters and ranges of a specific Sidekiq card, the `skiq_read_parameters()` API should be used. The parameters available to query are defined in `skiq_param_t`.

5.4.4 Configuring an Interface using a Handle

Each of the underlying RF interfaces presents itself as a handle that can be used to reference the desired RF interface for purposes of configuration and data transfer. The libsidekiq API provides read/write access functions for a variety of radio configuration parameters. The full listing of these access functions can be found in the Sidekiq API documentation (`sidekiq_sdk_current/doc/api/Sidekiq_API_4.17.0.pdf`) as well as in `sidekiq_api.h`.

Handles sharing the same letter (A1/A2) or (B1/B2) indicates a phase-coherent relationship and shared resources, mainly a shared local oscillator (LO) and clocking. Consequently, adjusting the sample rate or center frequency on handle A1 will also impact the sample rate on A2, and vice versa. Despite this dependence, each Rx interface has its own independently configurable RF lineup including gain, overload detection, etc.

Please see the table below containing Sidekiq models, supported handles, and usage restrictions.

Sidekiq Variant	Supported Rx Handles	Notes
mPCIe	A1	
mPCIe (-001 variant)	A1, A2	
m.2	A1, A2	
X2	A1, A2, B1	
X4	A1, A2, B1, B2, C1, D1	Neither C1/A2 nor D1/A2 support simultaneous operation. 4 channel phase coherent Rx (A1,A2,B1,B2) is supported.
Sidekiq Stretch	A1	
NV100	A1, A2, B1	A2 and B1 cannot be used simultaneously.
Matchstiq Z3u	A1, A2	

Note: If streaming is attempted for conflicting streams, the start streaming call will fail with `-EBUSY`. To determine conflicting handles, the `skiq_read_rx_stream_handle_conflict()` API function can be used.

An example sequence for configuring a sub-set of the available parameters for multiple Rx interface handles is shown below.

```
result = skiq_write_rx_sample_rate_and_bandwidth(card, hdl_a1, sample_rate, bandwidth);
if ( result != 0 )
{
  printf("Error: failed to set sample rate to %u Hz on RxA1\n", sample_rate);
  return(-1);
}

result = skiq_write_rx_sample_rate_and_bandwidth(card, hdl_a2, sample_rate, bandwidth);
```

(continues on next page)

(continued from previous page)

```
if ( result != 0 )
{
  printf("Error: failed to set sample rate to %u Hz on RxA2\n", sample_rate);
  return(-1);
}

result = skiq_write_rx_LO_freq(card, hdl_a1, lo_freq);
if ( result != 0 )
{
  printf("Error: failed to set Rx LO freq to %" PRIu64 " Hz on RxA1\n", lo_freq);
  return(-1);
}

result = skiq_write_rx_gain_mode(card, hdl_a1, skiq_rx_gain_manual);
if ( result != 0 )
{
  printf("Error: failed to set Rx gain mode to manual on RxA1\n");
  return(-1);
}

result = skiq_write_rx_gain_mode(card, hdl_a2, skiq_rx_gain_manual);
if ( result != 0 )
{
  printf("Error: failed to set Rx gain mode to manual on RxA2\n");
  return(-1);
}

result = skiq_read_rx_gain_index_range(card, hdl_a1, &gain_index_min, &gain_index_max);
if ( result != 0 )
{
  printf("Error: failed to read Rx gain index on RxA1\n");
  return(-1);
}
rx_a1_gain = gain_index_min;

result = skiq_read_rx_gain_index_range(card, hdl_a2, &gain_index_min, &gain_index_max);
if ( result != 0 )
{
  printf("Error: failed to read Rx gain index on RxA2\n");
  return(-1);
}
rx_a2_gain = gain_index_max;

result = skiq_write_rx_gain(card, hdl_a1, rx_a1_gain);
if ( result != 0 )
{
  printf("Error: failed to set Rx gain to %u on RxA1\n", rx_a1_gain);
  return(-1);
}

result = skiq_write_rx_gain(card, hdl_a2, rx_a2_gain);
if ( result != 0 )
{
  printf("Error: failed to set Rx gain to %u on RxA2\n", rx_a2_gain);
  return(-1);
}
```

All access functions return an `int32_t` with a status code, where 0 = success and any other value indicates an error that occurred.

5.4.5 Frequency Hopping

As of libsidekiq v4.10, the ability to rapidly hop amongst a user pre-defined frequency list is supported for most products. Examples of how to use the frequency hopping APIs are provided in the `tx_configure` and `rx_samples_freq_hopping` test applications.

The RFICs used on Sidekiq X4 and libsidekiq impose a restriction on how to perform frequency hopping. A hop index can only be written to a "mailbox" slot on the RFIC. A hop operation executes a retune to the frequency in the "next" slot, then delivers the index from the "mailbox" to the "next" slot. It acts much like a 2 element deep FIFO that must have 1 or 2 indices enqueued and cannot go empty. Even though this approach is only required for Sidekiq X4, libsidekiq presents a consistent interface across all radio products.

The general API flow (without error checking) to configure and utilize frequency hopping are shown in the figure below. Details of the API functions related to frequency hopping are outlined in the sections that follow.

```
skiq_rx_hdl_t hdl = skiq_rx_hdl_A1;
uint8_t card = 0;
uint8_t num_hop_freqs = 5;
uint8_t initial_hop_idx = 0;
uint64_t freq_list[num_hop_freqs] = { 100000000, 200000000, 300000000, 400000000, 500000000 };

/* an RF timestamp of 0 indicates an "immediate" hop since 0 is always in the past */
uint64_t rf_timestamp = 0;

/* configure frequency tune mode for "immediate" */
skiq_write_rx_freq_tune_mode( card, hdl, skiq_freq_tune_mode_hop_immediate );

/* configure the hopping frequency list and the initial hop index */
skiq_write_rx_freq_hop_list( card, hdl, num_hop_freqs, freq_list, initial_hop_idx );

/* prepare the next hop to follow the initial hop index to be at freq_list[3] */
skiq_write_next_rx_freq_hop( card, hdl, 3 );

/* immediately execute the hop to the initial frequency (i.e. freq_list[initial_hop_idx]) */
skiq_perform_rx_freq_hop( card, hdl, rf_timestamp );

/* prepare the next hop to follow the freq_list[3] to be at freq_list[1] */
skiq_write_next_rx_freq_hop( card, hdl, 1 );

/* immediately execute the hop to the second frequency (i.e. freq_list[3]) */
skiq_perform_rx_freq_hop( card, hdl, rf_timestamp );

/* at this point, there is one hop left in the RFIC's hop FIFO: freq_list[1].  In order to hop to
   that frequency, however, a frequency index *must* first be written to the "mailbox" slot using
   skiq_write_next_rx_freq_hop() or the FIFO will go empty (which is not allowed) */
```

Configuring Tune Mode

The tune mode can be configured to hop on timestamp (`skiq_tune_mode_hop_on_timestamp`), hop immediately (`skiq_tune_mode_hop_immediate`), or to use the default of the "standard" tuning (`skiq_tune_mode_standard`).

The mode can be configured with `skiq_write_rx_freq_tune_mode()` / `skiq_write_tx_freq_tune_mode()`. Additionally, the current mode can be queried with `skiq_read_rx_freq_tune_mode()` / `skiq_read_tx_freq_tune_mode()`.

Note that while Sidekiq X2's RFIC does not support fast frequency hopping, the API to perform the LO tuning is supported when using `skiq_tune_mode_hop_immediate` (as of libsidekiq v4.12.0).

When retuning with the frequency hopping API in either the `skiq_tune_mode_hop_on_timestamp` or `skiq_tune_mode_hop_immediate` tuning mode, the RFIC's calibration algorithms are not performed to support faster tuning. If rapid tuning is desired at the cost of reduced RF performance, the frequency hopping tune mode is recommended. However, if maximum reduction of DC offset or image is desired, then the standard tuning mode is recommended.

Frequency List Definition

When using frequency hopping, a list of frequencies must be specified prior to executing a hop to a specific frequency. The maximum number of frequencies that can be specified is limited to `SKIQ_MAX_NUM_FREQ_HOPS`.

The frequency list can be defined with `skiq_write_rx_freq_hop_list()` / `skiq_write_tx_freq_hop_list()`. A previously specified frequency list can be queried with `skiq_read_rx_freq_hop_list()` / `skiq_read_tx_freq_hop_list()`.

Prepare Next Frequency Hop

The next frequency to hop to **must** be configured prior to executing the frequency hop using either the `skiq_write_next_rx_freq_hop()` / `skiq_write_next_tx_freq_hop()` API calls. To determine the details of the previously configured "next hop", the `skiq_read_next_rx_freq_hop()` / `skiq_read_next_tx_freq_hop()` API calls can be used. When a hop is executed with `skiq_perform_rx_freq_hop()` / `skiq_perform_tx_freq_hop()`, this will be the frequency that is configured.

Execute Frequency Hop

As long as there was a previously configured "next hop", the "perform hop" APIs can be used to execute the frequency hop. In the case of `skiq_tune_mode_hop_immediate`, the frequency hop is executed immediately. In the case of `skiq_tune_mode_hop_on_timestamp`, the frequency hop is not initiated until the specified timestamp has been reached; if a timestamp in the past is specified, then the frequency hop occurs immediately. The API to complete the frequency hop is `skiq_perform_rx_freq_hop()` / `skiq_perform_tx_freq_hop()`. The `skiq_read_curr_rx_freq_hop()` / `skiq_read_curr_tx_freq_hop()` API functions can be used to query the details of the currently configured frequency.

Note: Hopping on timestamp uses GPIO pins between the FPGA and RFIC to execute the frequency hop. If these pins are already in use with a custom FPGA design, unexpected behavior may occur.

5.4.6 Operation Modes

Sidekiq can support 3 different modes of operation: single channel receiver (Rx A/B/C/D 1 only), dual channel receiver (both Rx A/B1 and Rx A/B2), or single channel transceiver (Rx A1 / Tx A1). With the Sidekiq-002, Sidekiq Z2 products, and Sidekiq m.2 2280 only a single channel transceiver is available. For Matchstiq Z3u, either a single channel transceiver is supported or a dual channel receiver mode can be used. The operation mode is specified by configuring the channel mode:

```
/* configure mode for both Rx A/B1 and Rx A/B2 */
skiq_write_chan_mode(card, skiq_chan_mode_dual);
```

If only Rx A1/B1/C1/D1 is needed for receiving or if transmitting, the mode should be set to `skiq_chan_mode_single`. This mode should be configured prior to starting an interface or undefined behavior may occur.

The Sidekiq m.2, Sidekiq X2, Sidekiq X4, and Sidekiq NV100 products additionally support a dual channel transceiver (Rx A/B 1/2 and Tx A/B 1/2). The `skiq_write_chan_mode()` API call may be used as well to configure the channel mode.

Finally, both Sidekiq X4 and Sidekiq NV100 are each based on different RFICs found on the other Sidekiq products. These RFICs are designed primarily for TDD usage. This means that Rx and Tx cannot be streaming at the same time for the A* or B* handles. Note that streaming TX for an A* handle can happen simultaneously while receiving on a B* handle. Additionally, the LO frequency is shared across Rx and Tx frequencies. In the case of Sidekiq X4, `skiq_rx_hdl_A1 / skiq_rx_hdl_A2` share the LO frequency with `skiq_tx_hdl_A1 / skiq_tx_hdl_A2` and `skiq_rx_hdl_B1 / skiq_rx_hdl_B2` share the LO frequency with `skiq_tx_hdl_B1 / skiq_tx_hdl_B2` for the Sidekiq X4 product. In the case of Sidekiq NV100, `skiq_rx_hdl_A1 / skiq_rx_hdl_A2` share an LO, and `skiq_tx_hdl_A1/skiq_tx_hdl_B1` share an LO with `skiq_rx_hdl_B1`. The ability to adjust the Tx LO selection will be supported in a future libsidekiq version.

5.4.7 RF Port Configuration

Certain versions of the Sidekiq product support configuration of the RF port mode to perform either transmission or reception of sample data (referred to as TRx operation). Alternatively, the Sidekiq card operates in fixed mode, where the RF ports cannot be switched between receive and transmit. A detailed table outlining options and configurations per Sidekiq is captured in *Detailed RF Port Configuration* (page 100).

Certain versions of the Sidekiq card also support configuration of the RF port to RF handle mapping. Note that only specific RF ports are available in the fixed or TRx mode. To determine RF ports available for a specific handle, the following API can be used:

```
/* read the RX ports available for Rx A1 */
skiq_read_rx_rf_ports_avail_for_hdl(card,
                                    skiq_rx_hdl_A1,
                                    &num_fixed_rf_ports,
                                    fixed_rf_ports,
                                    &num_trx_rf_ports,
                                    trx_rf_ports );

/* read the TX ports available for Tx A1 */
skiq_read_tx_rf_ports_avail_for_hdl(card,
                                    skiq_tx_hdl_A1,
                                    &num_fixed_rf_ports,
                                    fixed_rf_ports,
                                    &num_trx_rf_ports,
                                    trx_rf_ports );
```

The currently configured RF port can be queried and optionally configured, as shown below.

```
/* read the configured RF port for Rx A1 */
skiq_read_rx_rf_port_for_hdl(card,
                             skiq_rx_hdl_A1,
                             &rf_port );

/* read the configured RF port for Tx A1 */
skiq_read_tx_rf_port_for_hdl(card,
                             skiq_tx_hdl_A1,
                             &rf_port );
```

(continues on next page)

(continued from previous page)

```
/* write the configured RF port for Rx A1 */
skiq_write_rx_rf_port_for_hdl(card,
                              skiq_rx_hdl_A1,
                              skiq_rf_port_J2 );

/* write the configured RF port for Tx A1 */
skiq_write_tx_rf_port_for_hdl(card,
                             skiq_tx_hdl_A1,
                             skiq_rf_port_J1 );
```

The RF port modes available for a specific Sidekiq can be queried directly as shown below, or accessed via the `skiq_params_t`.

```
/* determine the RF port configuration available */
skiq_read_rf_port_config_avail( card, &fixed_mode, &trx_mode );
```

Additionally, some variants of Sidekiq support updating the RF port configuration dynamically. The current RF port configuration can be read by:

```
/* determine the current RF port configuration */
skiq_read_rf_port_config( card, &rf_port_config );
```

Finally, when operating with the RF configuration of `skiq_rf_port_config_trx`, the mode of RF port can switch between receive and transmit with the following API.

```
/* switch the operation of the RF port to transmit */
skiq_write_rf_port_operation( card, true );

/* switch the operation of the RF port to receive */
skiq_write_rf_port_operation( card, false );
```

For more details on the modes available for a specific hardware variant of Sidekiq, please contact Epiq Solutions support *[5]* (page 8). For an example of switching between receive and transmit modes, refer to the `tdd_rx_tx_samples.c` test application.

5.4.8 I/Q Ordering Mode

The order in which complex samples are received or transmitted by Sidekiq is a configurable option. Ordering can be adjusted by the user at run-time *before* streaming is started by means of the `skiq_write_iq_order_mode()` function and the `skiq_iq_order_t` type. If not specified, Sidekiq will operate with `skiq_iq_order_qi`, as depicted in the *Rx I/Q Packet Structure* (page 38) and *Tx Packet Structure* (page 45) figures. In this mode, Q0 is data[0], I0 is data[1], Q1 is data[2], I1 is data[3], and so on.

Alternatively in the case of `skiq_iq_order_iq`, the order will be swapped: I0 is data[0], Q0 is data[1], I1 is data[2], Q1 is data[3], and so on. For an example of adjusting the ordering mode, please refer to the test application `rx_samples.c` or `tx_samples.c`.

5.4.9 Packed Mode (Sidekiq mPCIe, m.2, and Stretch / m.2-2280 only)

By default, Sidekiq radios operate in 'unpacked' mode where each I or Q sample is stored in its own 16-bit value. However, not all radios use all 16-bits for sample data - see below table for more information. To query the supported I/Q sample size (resolution) using the libsidekiq API, use `skiq_read_rx_iq_resolution()` and `skiq_read_tx_iq_resolution()`.

Table 5.3: Unpacked sample sizes per radio

Radio	RX sample size (in bits)	TX sample size (in bits)
Sidekiq mPCIe	12 *	12 *
Sidekiq m.2	12 *	12 *
Sidekiq Stretch	12 *	12 *
Sidekiq Z2	12 *	12 *
Matchstiq Z3u	12 *	12 *
Sidekiq X2	16	14 (upper)
Sidekiq X4	16	14 (upper)
Sidekiq NV100	16	16

* – sample is sign extended to 16 bits

Additionally, Sidekiq mPCIe, Sidekiq m.2, and Sidekiq Stretch also support a mode of operation that compacts the I/Q samples to use 12-bits instead of 16-bits per I and Q sample. This mode is referred to as packed mode. Packed mode is useful when it is desired to operate at a sample rate higher than the rate at which the transport interface can reliably transfer data. This allows for more samples to be transferred within a given time period at the cost of requiring unpacking of the samples prior to consumption when receiving or packing of the samples while transmitting. Packed mode can be enabled or disabled by calling the `skiq_write_iq_pack_mode()` function. The mode applies to both transmitting and receiving samples. When packed mode is enabled, the sample data should be formatted as shown in *Sidekiq Packed Sample Structure* (page 36). The format of the metadata remains the same regardless of whether packed mode is enabled or not.

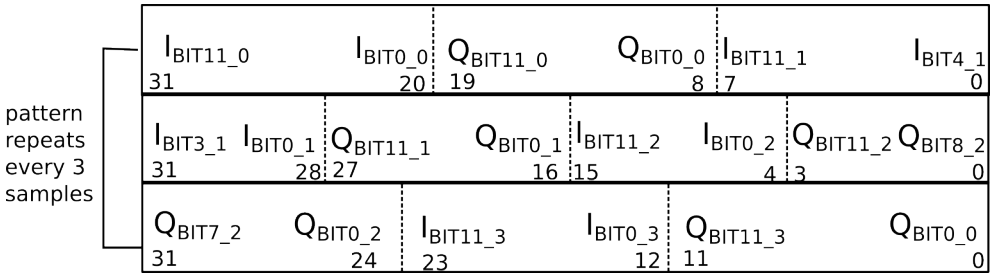

Fig. 5.1: Sidekiq Packed Sample Structure

5.4.10 Starting an Rx Interface

Once an Rx interface has been properly configured for operation, the interface can then be started to begin data flowing between the CPU and FPGA. The process of starting an Rx interface allows a user to specify the Rx interface handle to start. An example of starting an Rx interface is shown below.

```
/* begin streaming on the Rx interface */
result = skiq_start_rx_streaming(card, hdl);
```

Reading I/Q Samples from an Rx Interface

Once an Rx interface has started streaming, a block of contiguous I/Q samples can be read from the Rx interface using the `skiq_receive()` function. This function accepts a pointer to a `skiq_rx_block_t` pointer that will then be populated with the address of where the metadata and I/Q data block is stored in the DMA engine provided by libsidekiq. No memory copy operation is performed here when using the PCIe transport, just an update of the passed in pointer, to maximize efficiency of accessing the data. Note that this memory is controlled by the DMA interface and is not reserved for the application. Thus, if the data is not copied or processed prior to the DMA interface requiring

that memory for new sample data, there is the potential for the sample data to be overwritten. The sample rate of each of the active interfaces will determine how quickly the DMA engine fills up the and overwrites previous sample data. To check for an overflow condition, the RF pair timestamp should be monitored (as discussed in the below paragraphs and shown in the code example).

Note: Each successful call to `skiq_receive()` will only return a single packet.

The default behavior of `skiq_receive()` function is non-blocking. If there is no new data available, the function will return immediately with a status of `skiq_rx_status_no_data`. For details on how to configure and use the blocking receive capabilities, refer to *Developing Custom Applications with libsidekiq* (page 26). In the default case of non-blocking receive,the application is responsible for polling the receive interface for packets. There is some overhead associated with calling `skiq_receive()` when no new data is available, so it is recommended to throttle the calls based on when sample data is expected to be available or utilize the blocking receive capability. Additionally, when multiple Rx handles are enabled, the packets of sample data from each of the handles may be interleaved, and the `p_hdl` parameter of the `skiq_receive()` function is populated with the source handle from which the packet was received. The contents of `skiq_receive()` arguments are valid only when `skiq_rx_status_success` is returned.

The `skiq_rx_block_t` structure has fields for the I/Q data and elements of the metadata in the header. Note: future versions of libsidekiq may adjust this header size, so there is a `#define` called `SKIQ_RX_HEADER_SIZE_IN_WORDS` which specifies the size of the header. The metadata contains a timestamp which is incremented at the same rate as the sample rate of that Rx interface. This field is called `rf_timestamp` in the `skiq_rx_block_t` struct. Additionally, the phase coherent RF pair (Rx A1/2 or Rx B1/2) uses the same timestamp reference, or in the case of Sidekiq X4, all 4 phase coherent receivers use the same timestamp reference. This allows the samples from Rx A1 to be phase aligned with samples from Rx A2 (or in the case of Sidekiq X4, Rx A1/A2/B1/B2 are all phase aligned). The metadata also contains the system timestamp associated with the packet (named `sys_timestamp`). The system timestamp is maintained independent of the sample rate and is used across all of the Rx interfaces. The system metadata (currently available only with PCI transport) comprises of the RF IC control output bits, the Rx overload status (not available with all products), and the data source (handle). The data source correlates to the `skiq_rx_hdl_t` definition. The Rx overload bit is set if the Rx overload detection was active for that packet (not available with all products). The RF IC control output bits are defined by both the RF IC control output mode and enable; configured by the `skiq_write_rfic_control_output_config()` API. For details on the modes available, please refer to the "Control Output" section of *[9]* (page 8) (for Sidekiq mPCIe / m.2 / m.2-2280 / Z2 / Z3u), "Monitor Output" section of *[10]* (page 8) (for Sidekiq X2), or "GPIO Monitor Mode" of *[11]* (page 8) (for Sidekiq X4). Note that the `rx_samples.c` test application provides an example of configuring the mode such that the gain of RxA1 is present in the metadata. The user metadata is a 32-bit field available for custom FPGA applications to use if desired. The '`data`' field of the `skiq_rx_block_t` struct consists of I/Q sample data. The fields of the example I/Q packet is shown below. In some applications it may be desirable to change the ordering of the complex (I/Q) samples. For details on how to do so, please refer to *I/Q Ordering Mode* (page 35)

Table 5.4: Rx I/Q Packet Structure

Field	Width (bits)	Description
`rf_timestamp`	64	RF timestamp associated with the received sample block
`sys_timestamp`	64	System timestamp associated with the received sample block
`hdl`	6	Receive handle indicating the receive handle associated with the received sample block
`overload`	1	RF Overload indicating whether or not the RF input was overloaded for the received sample block
`rfic_control`	8	RFIC control word carries metadata from the RFIC, typically the receive gain index
`id`	8	Channel ID used by channelizer (currently unused)
`system_meta`	6	System metadata (unused / reserved)
`version`	3	Packet version field
`user_meta`	32	User metadata typically populated by a custom FPGA build
`int16_t data[]`	16	array of unpacked I/Q samples (16 bits per I or Q sample). Q_0 is data[0], I_0 is data[1], Q_1 is data[2], I_1 is data[3], and so on.

Note: If a sufficiently high sample rate is selected (typically in the range of ~50 Msamples/sec - equivalent to 200 MB/sec - for Sidekiq mPCIe, dependent on the capabilities of the host platform), gaps in the timestamps may occur due to either:

1. the libsidekiq software ring buffer filling up, or
2. the throughput rate of the FPGA → CPU interface being exceeded.

For applications that require continuous data flow, it is imperative to confirm that the timestamp is monotonically increasing at the expected rate to ensure no data is dropped. For recommendations on how to evaluate the Rx throughput performance for a particular host system, refer to *Receive Performance* (page 92).

An example loop that does nothing but read blocks of I/Q data and verifies the timestamps (to ensure that no gaps exist in the data stream) is shown below.

```
/* loop through and acquire the requested # of i/q sample blocks, verifying
    that the timestamp (ts) increments as expected */
while( num_blocks < tot_num_blocks )
{
  status = skiq_receive(card, &hdl, &p_rx_block, &len);
  if( status == skiq_rx_status_success )
  {
    if( p_rx_block != NULL )
    {
      curr_ts = p_rx_block->rf_timestamp;

      if( first_block == true )
      {
        first_block = false;
        next_ts = curr_ts;
        next_ts += ((len/4) - SKIQ_RX_HEADER_SIZE_IN_WORDS);
      }
      else if( curr_ts != next_ts )
      {
```

(continues on next page)

(continued from previous page)

```
        printf("Error: timestamp error expected 0x%016" PRIx64 " but got 0x%016" PRIx64
               ".\n", next_ts, curr_ts);

        return (-1);
      }
      else
      {
        next_ts += ((len/4) - SKIQ_RX_HEADER_SIZE_IN_WORDS);
      }
    }

    num_blocks++;
  }
}
```

Lastly, most parameters associated with a given Rx interface can be updated after the interface has been started. It is not necessary to stop the interface to re-configure an interface. The only exceptions here are that changes to the channel mode, packed mode, I/Q order mode, and data source mode must be configured prior to starting the interface. Also, a change to sample rate will result in the data flow being automatically stopped and restarted once the sample rate has been applied.

Counter Mode with Rx Interface

An Rx interface can be configured to either provide I/Q sample data or counter data. The normal mode of operation is using the I/Q sample mode. However, counter mode can be useful in various test scenarios. The type of data provided can be configured with the `skiq_write_rx_data_src()` function. The data source can be updated at any time but is only applied when streaming is started. When in counter mode, each sample is a 12-bit value for Sidekiq mPCIe, m.2, m.2-2280, Z2 and Matchstiq Z3u and 16-bit for Sidekiq X2, X4, and NV100. When running in counter mode on the Sidekiq, the I sample is odd while the Q sample is an even number. For details on how to validate the counter data, refer to the `verify_data()` function in the `rx_samples.c` test application.

Making `skiq_receive` a blocking call

As of libsidekiq v3.3.0, a call to `skiq_receive()` can be configured to block until an I/Q sample block is available instead of returning immediately. Using a blocking infrastructure can save CPU cycles and provide other processes with time to execute. The addition of a blocking receive call also provides flexibility in how libsidekiq may be leveraged for different use cases.

Note: Not all transports support blocking receive.

The arguments to `skiq_receive()` stay the same and libsidekiq continues to default to non-blocking for `skiq_receive()`. The `skiq_set_rx_transfer_timeout()` API function allows a developer to specify how `skiq_receive()` behaves when samples are not available. For example, if the developer specifies `RX_TRANSFER_NO_WAIT` to `skiq_set_rx_transfer_timeout()`, a call to `skiq_receive()` returns immediately if samples are not available (this is the default behavior). A developer may specify a timeout between 20uS and 1000000uS to `skiq_set_rx_transfer_timeout()`. In this configuration, a call to `skiq_receive()` will return after the specified timeout has elapsed if samples are not available. If the developer specifies `RX_TRANSFER_WAIT_FOREVER` as the timeout, a call to `skiq_receive()` will block indefinitely until samples are available. In both the timeout and `RX_TRANSFER_WAIT_FOREVER` configurations, `skiq_receive()` returns once samples are available at the next opportunity the kernel provides to the associated process.

Note: For improved CPU usage and efficiency in receiving, a non-zero timeout is recommended. Additionally, a timeout that is greater than the inter-sample-block timing at the configured Rx sample rate is also recommended. In most cases, a timeout of 25000uS is sufficient to reap the benefits of a blocking receive.

Caution: when using a non-zero timeout, calling `skiq_stop_rx_streaming()` or `skiq_exit()` from another thread can cause `skiq_receive()` to return without a packet. Be sure to handle that case.

If desired to configure a timeout for the receive call, the API `skiq_get_rx_transfer_timeout()` provides the timeout value or a pre-processor define (`RX_TRANSFER_NO_WAIT`, `RX_TRANSFER_WAIT_FOREVER`, or `RX_TRANSFER_WAIT_NOT_SUPPORTED`). The value `RX_TRANSFER_WAIT_NOT_SUPPORTED` is provided in cases where the transport layer (currently custom or USB) does not support a blocking receive infrastructure.

For an example application that optionally makes use of the blocking receive capabilities, refer to the `rx_samples.c` test application.

Using receive calibration offsets

With the introduction of libsidekiq v4.4.0, Sidekiq units leaving the factory have a per-unit receiver calibration data stored in non-volatile memory. There are several API functions (introduced in libsidekiq v4.0.0) that provide a receive calibration offset based on the RF configuration that may be applied by the user to calculate the calibrated RF energy present in a block of I/Q samples. If a unit does not have stored calibration data, the API functions will fall back to a default data set that represents a given product line (miniPCIe vs m.2 vs Z2, etc), receive handle (`skiq_rx_hdl_t`), and RF port (`skiq_rf_port_t`). The four API functions that provide a calibration offset are as follows.

- `skiq_read_rx_cal_offset()`
- `skiq_read_rx_cal_offset_by_gain_index()`
- `skiq_read_rx_cal_offset_by_LO_freq()`
- `skiq_read_rx_cal_offset_by_LO_freq_and_gain_index()`

These functions provide varying degrees of control when querying for a receive calibration offset. If a parameter is not specifiable, it is taken from the present RF configuration.

In order to use the receive calibration offset G_{radio}, a user should first calculate the baseband power in dB (P_{bb}), then subtract the calibration offset (returned in units of dB) to calculate the RF power (P_{rf}) of the signal in dBm. The equations below further describe how to apply G_{radio}:

$$P_{bb} = 10\ log_{10}\left(\frac{1}{N}\sum_{k=0}^{N} i_k^2 + q_k^2\right)$$
$$P_{rf} = P_{bb} - G_{radio}$$

Using I/Q phase and amplitude calibration

With the introduction of libsidekiq v4.7.0, the Sidekiq X4 FPGA design provides in-line complex multipliers for each receiver handle (Rx A1, Rx A2, Rx B1, and Rx B2; not available with Rx C1/D1) to allow a user to reduce phase and amplitude differences among the receivers. At this release, the Sidekiq X4 units leaving the factory have per-unit I/Q phase and amplitude calibration data stored in non-volatile memory. There exist several API functions that read the calibration settings for a given LO frequency as well as allow a user to either override or supplement the adjustment

values. At this time, if a unit does not have stored calibration data, there is no default data set. The API functions that provide access to this feature are as follows. Refer to the API documentation for additional details on this interface.

- `skiq_read_iq_complex_multiplier()`
- `skiq_read_iq_cal_complex_multiplier()`
- `skiq_read_iq_cal_complex_multiplier_by_LO_freq()`
- `skiq_write_iq_complex_multiplier_absolute()`
- `skiq_write_iq_complex_multiplier_user()`
- `skiq_read_iq_complex_cal_data_present()`

Note that `skiq_read_iq_complex_multiplier()` returns the complex multiplication factor currently in use, while `skiq_read_iq_cal_complex_multiplier()` (note the extra *cal* in the function name), returns the complex multiplication factor as determined by the unit's factory calibration. These two factors may differ if a user has overwritten (`skiq_write_iq_complex_multiplier_absolute()`) or supplemented (`skiq_write_iq_complex_multiplier_user()`) the factor for a given receiver handle.

The factor is automatically updated (or reset) whenever the receive LO frequency is configured. Any user modifications to the multiplication factor must be reapplied after a frequency configuration.

In libsidekiq v4.7.0, these multipliers are always in effect on Sidekiq X4. If a user wishes to disable them, setting an absolute factor of 1+0j is recommended.

5.4.11 Configuring a Tx Interface

There are various aspects of the transmit interface which must be configured prior to starting streaming. These control the behavior of `skiq_transmit()` and are described in detail in the following sections. For recommendations on how to evaluate the Tx throughput performance for a particular host system and the various Tx configuration parameters, refer to *Transmit Performance* (page 92).

Block Size Configuration

The block size is the number of samples included in each transmit packet. A transmit packet consists of the transmit header data as well as the block, which contains the transmit samples (refer to *Tx Packet Structure* (page 45) for the structure of a transmit packet). While in packed mode, the block size refers to the number of words contained in the packet not including the header data.

The block size is adjustable and in general, a larger block size results in higher throughput without underruns or late timestamps. The trade-off of using a larger block size is an increase in latency in the transfer of the transmit packet.

A block size + Tx header size (as defined by `SKIQ_TX_HEADER_SIZE_IN_WORDS`) must be a multiple of 256 words. The block size can be configured with `skiq_write_tx_block_size()`. If an invalid block size is configured, `skiq_start_tx_streaming()` will result in a failure. A few examples of valid block sizes when running in single channel mode are: 1020 (packet size=1020+4=1024), 2044 (packet size=2044+4=2048), and 16380 (packet size=16380+4=16384).

When running in dual channel mode, the block size refers to the number of words contained for each channel. Also, when in dual channel mode, the maximum block size is limited by the FPGA Tx FIFO size. Note: dual channel transmit is supported only with Sidekiq m.2, X2, X4, and NV100. Dual channel transmit can be with A1/A2 or A1/B1 handle pairs. The secondary handle is used in calls to start streaming and `skiq_transmit()`. When running in dual channel, a few example valid packet sizes are: 1022 (packet size=1022 TxA1 samples + 1022 TxA2 samples + 4 header=2048), 2046 (packet size=2046 TxA1 samples + 2046 TxA2 samples + 4 header=4096), and 8190 (8190 TxA1 samples + 8190 TxA2 samples + 4 header=16384)

When running in packed mode (Sidekiq mPCIe, m.2, and m.2-2280 only), care must be used to ensure that the block size contains an integer number of samples and that the block size + header size remains a multiple of 256 words. In packed mode, the block size refers to the number of words contained, not the number of samples. For example, a block size of 252 results in 336 packed samples (252 * `SKIQ_PACKED_SAMPLE_RATIO` = 336), which is a valid packed mode configuration. However, a block size of 508 results in 677.3 samples (508 * `SKIQ_PACKED_SAMPLE_RATIO` = 677.3), which is invalid.

Data Flow Mode

The Tx interface can operate in one of several "data flow modes", as configured by the user application. The default data flow mode is "immediate". In this mode, timestamps are ignored and the data is buffered up and transmitted out as soon as possible. This is useful for applications that do not have a requirement for precise timing.

For applications that require fine grained control of the timing of the transmitted samples, the "with_timestamps" data flow mode should be utilized. Each block of I/Q samples sent down to the Tx interface will need to have a 64-bit timestamp located at `SKIQ_TX_TIMESTAMP_OFFSET_IN_WORDS` into the beginning of the block of data by the application (see `tx_samples.c` in the `test_apps` directory for an example). This timestamp corresponds to the time when the very first sample of the block will be transferred from the FPGA's I/Q sample FIFO to the D/A converters. Each subsequent complex sample in the block will then be transferred to the D/A converters one at a time as the timestamp increments. This continues until the entire data block has been transmitted out. If continuous transmission of I/Q samples is required, it is up to the user application to ensure that this pipeline of data between the user application and the FPGA transmit FIFO is kept filled, thus preventing an underflow. In this mode, the FPGA will not transmit any samples that specify a timestamp that is in the past; the samples will be discarded, the Tx FIFO will be flushed, and the late timestamp counter is incremented (which can be read with the `skiq_read_tx_num_late_timestamps()` function call).

As of libsidekiq v4.6.0, certain bitstreams support an additional mode: "with_timestamps_allow_late_data". Selecting this mode will result in a return value of `-ENOTSUP` if not supported. In this mode, the FPGA will transmit samples as described in the above "with_timestamps" section, but will also transmit samples that have a timestamp that's already past. Please note that in this mode, the late timestamp counter will not be updated, even if samples with late timestamps are transmitted.

Tx Timestamp Clock Source selection

The timestamp source can be configured to either the system or RF (default) clock, when used in conjunction with a timestamp data flow mode. Using a system timestamp may be preferred if an application frequently changes sample rates (which causes the RF timestamp to pause). One caveat for using system timestamps is that it requires a data flow mode of "allow late timestamps mode" due to the implementation.

Configuring the Tx timestamp clock source is supported on systems using FPGA bitstream v3.15.1 and above.

Transfer Mode

The transfer mode of the transmit interface can be configured to operate either synchronously or asynchronously. The transfer mode can be configured with the `skiq_write_tx_transfer_mode()` function.

When running in `skiq_tx_transfer_mode_sync` mode, `skiq_transmit()` blocks if the FPGA transmit FIFO is full until there is space for the next packet of data. The FPGA FIFO is relatively small (as defined by `skiq_fpga_tx_fifo_size_t`), so it may be necessary for users to implement their own external buffering of sample data prior to calling the `skiq_transmit()` call. This mode can be simpler to interface with in that once the `skiq_transmit()` function returns, the data has been transferred to the FPGA and can immediately be freed or reused by the application. However, it only allows for a single packet at a time to be in flight to the FPGA. This results in a potential decrease of throughput efficiency. The `tx_samples.c` application provides an example of how to use the synchronous transfer mode.

An alternative to the synchronous transfer mode is to run in `skiq_tx_transfer_mode_async`, which results in an increased throughput. When running in `skiq_tx_transfer_mode_async` mode, `skiq_transmit()` returns immediately with a status of either `0` or `SKIQ_TX_ASYNC_SEND_QUEUE_FULL`. A maximum of `SKIQ_MAX_NUM_TX_QUEUED_PACKETS` can be queued at given point in time. If a status of `0` is returned, this indicates that the packet was successfully queued for transmission but not necessarily transferred to the FPGA yet. A status of `SKIQ_TX_ASYNC_SEND_QUEUE_FULL` indicates that the software buffer is full and the packet was not queued successfully. The `skiq_transmit()` call must be repeated with this data buffer to transmit it. In order for the application to be notified when the asynchronous transmit operation has been completed by the Sidekiq library, a function pointer must be registered with the `skiq_register_tx_complete_callback()` prior to starting streaming on the Tx interface. Once the buffer has been successfully transferred to the FPGA, the callback function is called and the status of the transmission, a pointer to the data of the completed transmission, and optional user defined data associated specified when initiating the transmission is provided. At this point, it is safe to either reuse or free the buffer. Prior to the callback, the buffered should be considered in use by the Sidekiq library. The `tx_samples_async.c` application provides an example of how to use the asynchronous transfer mode. The `tx_benchmark` application can be used to help assess the performance tradeoffs with the different transfer modes and block sizes on the target host platform, as described in *Receive Performance* (page 92).

5.4.12 Starting the Tx Interface

Once a Tx interface has been properly configured for operation, the interface can then be started to begin data flowing between the CPU and FPGA.

Note: If the Sidekiq card is configured for dual channel mode, both TxA1 and TxA2 interfaces are enabled when `skiq_start_tx_streaming()` is called with `skiq_tx_hdl_A2` as the `hdl` argument. As of libsidekiq v4.13.0 and FPGA bitstream v3.13.0, the Sidekiq X4 can also transmit using TxA1 and TxB1 to provide the user two independently tunable interfaces. At its introduction in libsidekiq v4.17.0, Sidekiq NV100 also supports transmit using TxA1 and TxB1. In that case, the `skiq_tx_hdl_B1` handle needs to be specified as the `hdl` argument to `skiq_start_tx_streaming()`.

Caution: In a dual channel mode transmit configuration, the user must **only** specify the second handle to `skiq_start_tx_streaming()` and `skiq_stop_tx_streaming()` variants. Both Tx interfaces are enabled within each function call whenever the second handle is specified.

An update to the data flow mode / transfer mode / block size is allowed at any time, but is only applied when starting the Tx interface. An example of configuring and starting the interface is shown below.

```
/* configure the data flow mode to use timestamps */
skiq_write_tx_data_flow_mode(card, skiq_tx_hdl_A1,
                             skiq_tx_with_timestamps_data_flow_mode);

/* configure the transfer mode to synchronous */
skiq_write_tx_transfer_mode(card, skiq_tx_hdl_A1,
                            skiq_tx_transfer_mode_sync);

/* configure the size of the block to send */
skiq_write_tx_block_size(card, skiq_tx_hdl_A1, num_samples);

/* begin streaming on the Tx A1 interface */
skiq_start_tx_streaming(card, skiq_tx_hdl_A1);
```

Writing I/Q Samples to a Tx Interface

Once the Tx interface has been started, it is up to the user application to provide the I/Q data to libsidekiq for transmission. The I/Q data is provided to libsidekiq by calling `skiq_transmit()`. The `skiq_transmit()` can block if running in the synchronous transfer mode or return immediately if running the asynchronous transfer mode. For details on the transfer mode, refer to *Transfer Mode* (page 42).

The user application is responsible for populating the sample data prior to calling `skiq_transmit()`. The format of each user-provided transmit packet is shown in *Tx Packet Structure* (page 45). The `skiq_tx_block_t` struct type definition has fields that allow for easy access to the header and I/Q data. The `miscHigh` and `miscLow` fields are reserved for future use. The `timestamp` field is reserved for the transmit timestamp. In the "with_timestamps" data flow mode, the `timestamp` field is used to determine when the FPGA will actually send the data to the D/As. If the FPGA detects that the timestamp specified by software has already passed, the entire FPGA FIFO will be flushed and an error count is incremented. The number of "late timestamps" detected by the FPGA can be queried with the `skiq_read_tx_num_late_timestamps()` API. On certain bitstreams, the "with_timestamps_allow_late" data flow is available that acts much like "with_timestamps" mode, though the FPGA will transmit any samples with a timestamp that has already passed and not increment the "late timestamp" count. The API function `skiq_read_last_tx_timestamp()` provides the caller with the last timestamp the FPGA is acting upon and is useful for debugging applications.

In the "immediate" data flow mode, this timestamp value is ignored and can be set to zero by user applications. It is the user's responsibility to ensure that the FPGA has enough data to transmit. If the FPGA encounters a case where its FIFO is empty and there is no data to transmit while streaming, it will increment an underrun error count. This count can be queried with the `skiq_read_tx_num_underruns()` API.

The sample data (the `data` field of the `skiq_tx_block_t` struct) are 32-bit values, where the most significant 16-bits contains the "I" data, and the least significant 16-bits contain "Q" data. Each "I" and "Q" data word is represented in little endian form as a twos-complement number, sign-extended from the actual width of the D/A converters used in Sidekiq to 16-bits. For the Sidekiq mPCI, m.2, Z2, m.2-2280 and Matchstiq Z3u there are 12 data bits in each sample; for both Sidekiq X2 and Sidekiq X4, there are 14 data bits in each sample; and for Sidekiq NV100, there are 16 data bits in each sample.

The number of samples contained within a single block of data is variable and is configured with the `skiq_write_tx_block_size()` API, as described in *Block Size Configuration* (page 41). The format of the structure of the transmit packet is shown in *Tx Packet Structure* (page 45).

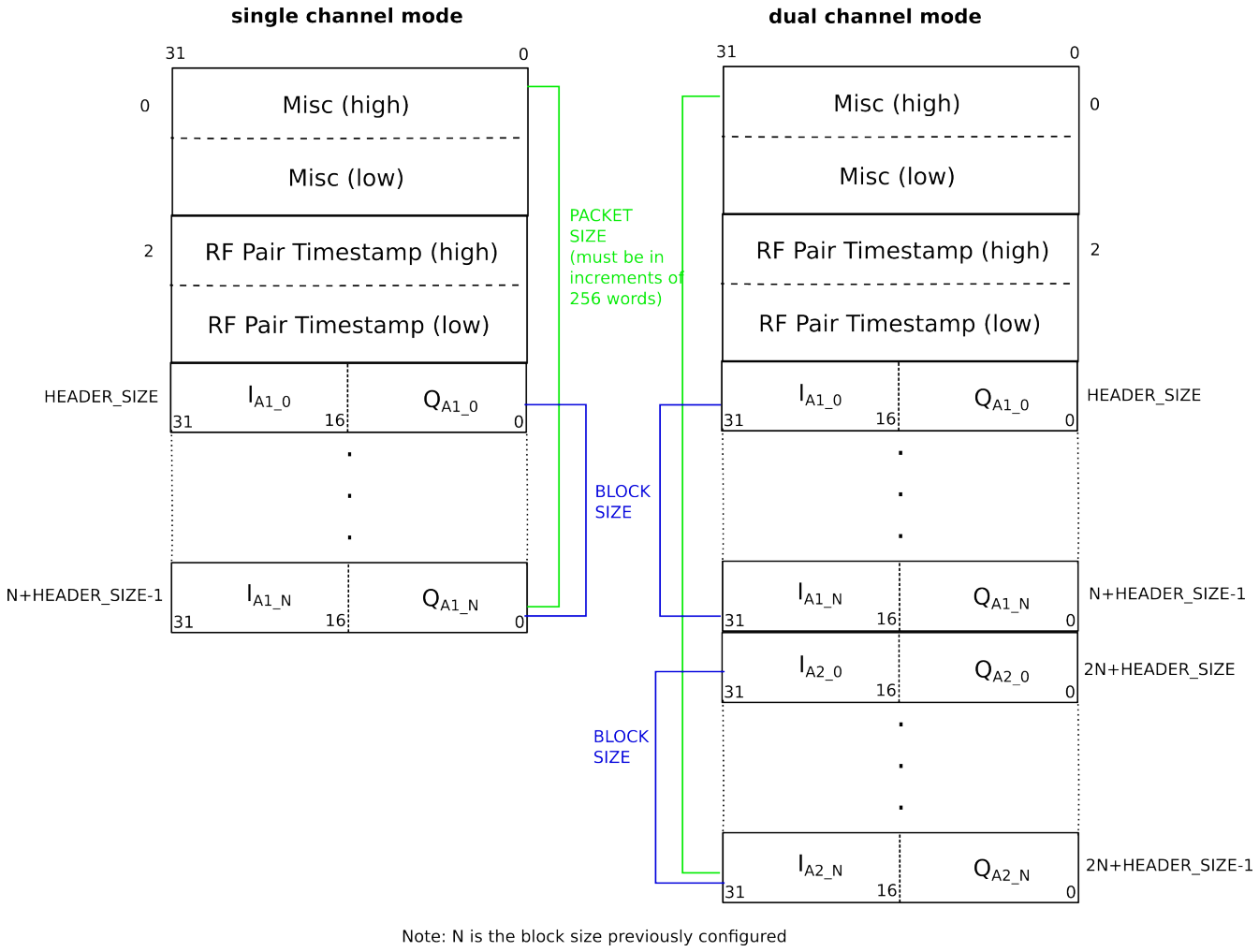

Fig. 5.2: Tx Packet Structure

See the `tx_samples.c` and `tx_samples_async.c` example applications in the `test_apps/` directory for details of how a real-world application uses this interface in both synchronous and asynchronous transfer modes.

5.4.13 Simultaneous use of Tx and Rx Interfaces

Most Sidekiq products support running the Tx and Rx interfaces simultaneously, with the exception of Sidekiq X4; these products support both FDD and TDD applications. Specifically, for an FDD application, the Rx interface can be tuned to one RF frequency, and the Tx interface is tuned to a different RF frequency. Note that Sidekiq X4 does allow for a A* handle to be configured for TX simultaneously to a B* handle configured for RX.

From a software application perspective, a multi-threading library (such as pthreads) can be used to manage the Rx/Tx interfaces in separate threads. See the `fdd_rx_tx_samples.c` test application for an example of performing simultaneous FDD operation.

In the case of TDD operation, the Tx and Rx interfaces can be configured separately, and then proceed through a sequence of starting and stopping the Tx and Rx interfaces as each transmit or receive operation is performed. An example of typical TDD operation can be found in the `tdd_rx_tx_samples.c` test application. Note: the transmit and receive FIFOs are flushed upon restarting the interface, so if the interface is stopped prior to the sample data completing reception or transmission, the data will be flushed.

Both the Rx and Tx interfaces share a common sample rate clock and timestamp, with the exception of certain configurations of the Sidekiq X2 resulting in a configuration where the Tx interface sample rate is twice the Rx sample rate. As a result, it is only necessary to configure the sample rate and reset the timestamp (if desired) for either the Rx or Tx interface.

5.4.14 Stopping and Releasing an Interface

When an application no longer needs to transfer data with a previously started interface, the interface can be stopped which will prevent future data transfers until the next "start" function is executed. Stopping an Rx interface takes place when the `skiq_stop_rx_streaming()` function is called. Stopping a Tx interface takes place when the `skiq_stop_tx_streaming()` function is called. Both the system timestamp and RF pair timestamp continues to increment regardless of whether the system is currently streaming.

Starting and Stopping on 1PPS

Sidekiq can be configured to start and stop streaming on a future 1PPS edge. To start Rx streaming on a 1PPS edge, the function `skiq_start_rx_streaming_on_1pps()` is called by the user application. To stop Rx streaming on a 1PPS edge, the function `skiq_stop_rx_streaming_on_1pps()` is called by the user application. A similar function exists for controlling the streaming operation on Tx (i.e., `skiq_start_tx_streaming_on_1pps()` and `skiq_stop_tx_streaming_on_1pps()`). All of the `skiq_*_streaming_on_1pps()` functions block until the 1PPS occurs.

To ensure that data begins to flow when the 1PPS occurs without any dropout, it is highly recommended that one thread is used to receive/transmit the data, and a separate thread is used to call the `skiq_start_*_streaming_on_1pps` function. Additionally, the thread performing the receiving/transmitting of data should be started prior to the call to start streaming. To ensure that the first packet queued to transmit is the first desired samples, it is recommended to register a callback for when the FPGA is ready to accept samples prior to the 1PPS occurring. The callback can be registered via the `skiq_register_tx_enabled()` API.

In the case of stopping the Rx streaming on a 1PPS, the `skiq_receive()` function needs to continue to be called after `skiq_stop_rx_streaming_on_1pps()` returns. This stop streaming function stops the data from being generated by the FPGA. However, there will be data remaining in the internal FIFOs, so `skiq_receive()` should be called until no data remains. Once there is no data returned from the `skiq_receive()` call, the `skiq_stop_rx_streaming()` function should be called to finalize the disabling of the data flow.

1PPS Source

As of libsidekiq v4.7.0, certain Sidekiq products and revisions allow for configuration of the 1PPS source. Specifically, the source of the 1PPS can be configured to be detected from an alternate pin. This configuration can be updated while running with the `skiq_write_1pps_source()` API. Additionally, the current configuration can be queried with the `skiq_read_1pps_source()` API.

Working with multiple receive handles

As of libsidekiq v4.6.0, the following API functions are available to synchronize or coordinate the starting and stopping of receive streaming. Each of these functions accepts an array of receive handles to start / stop together. It is important to understand that the "multi_immediate" variant does NOT currently synchronize the receive handles, but does offer a convenient way to start and/or stop multiple handles in a single call. As of libsidekiq v4.9.0, the "synchronized trigger" source was introduced that allows for multiple receive handles on a given card to start and/or stop with their RF timestamps synchronized.

Refer to the Sidekiq API documentation for more details on these functions.

- `skiq_start_rx_streaming_multi_immediate()`
- `skiq_start_rx_streaming_multi_on_trigger()`
- `skiq_stop_rx_streaming_multi_immediate()`
- `skiq_stop_rx_streaming_multi_on_trigger()`

For detailed examples on starting/stopping on a 1PPS (receive and/or transmit) or a synchronized trigger (receive only), refer to the test applications `rx_samples_on_trigger.c` and `tx_samples_on_1pps.c`.

5.4.15 Pin Control enable of RFIC signal paths (Sidekiq X4 only)

Since libsidekiq v4.14.0, the Tx/Rx signal paths on X4 can be controlled either through the libsidekiq API or by pins asserted by the FPGA user_app. For minimal latency when switching between receive and transmit, pin control is recommended.

Please consult the Sidekiq API documentation for more details on these functions.

- `skiq_read_rx_rfic_pin_ctrl_mode()`
- `skiq_read_tx_rfic_pin_ctrl_mode()`
- `skiq_write_rx_rfic_pin_ctrl_mode()`
- `skiq_write_tx_rfic_pin_ctrl_mode()`

5.4.16 Clock and Time Management Resources

Sidekiq uses a common reference clock to drive all of the RF/baseband hardware. This reference clock can come from other an on-board temperature compensated voltage controlled oscillator (TCVCXO) or an external reference clock (refer to Hardware User's Manual for specific Sidekiq type). By default, Sidekiq uses the on-board TCVCXO for its reference clock. This on-board TCVCXO has a stability of ± 1 PPM over the temperature range from -30 deg C to +85 deg C. For applications that need to dial in the accuracy of this timing reference even further, libsidekiq provides an API call to warp the timing reference by generating an analog control voltage using an on-board D/A converter (DAC) dedicated to this purpose. The `skiq_write_tcvcxo_warp_voltage()` function is used to warp the control voltage of the oscillator. The reference clock can be adjusted by -1 to +6 ppm by adjusting the DAC voltage. Valid warp voltage ranges are 0.75-2.25V, which corresponds to DAC values between 0 and 1023 for Sidekiq mPCIe / Sidekiq m.2 / Sidekiq Z2. On Sidekiq X2 rev C (unsupported in rev B), Sidekiq X4, Sidekiq Stretch, and Matchstiq Z3u the warp voltage ranges are 0.4-2.4V, or values of 7944-47662. For details on providing an external reference clock, refer to *[6]* (page 8).

In revision C of the mPCIe hardware and all revisions of the m.2, X2, and X4 hardware, the reference clock source is configurable via software. The configuration of the reference clock is a persistent setting. The current source can be queried via the `skiq_read_ref_clock_select()` API. As of libsidekiq v4.7.0, an additional reference clock source of a host, `skiq_ref_clock_host`, can also be configured for specific Sidekiq products. The host reference clock configuration utilizes an alternate clock frequency and input. For details on how to update this configuration, contact Epiq Solutions *[5]* (page 8).

The reference clock source for a card can be temporarily changed on-the-fly, allowing applications to choose the clock source as needed. The source can be changed via the `skiq_write_ref_clock_select()` function. This function will update the current source, however this change will not be stored in memory nor maintained between applications. This is not supported for all Sidekiq products. Products that do not support changing the reference clock source are Sidekiq M.2 and Sidekiq mPCIe cards.

The reference clock frequency for a card can also be temporarily changed on-the-fly, allowing applications to switch between external reference frequencies as needed. The frequency can be changed via the `skiq_write_ext_ref_clock_freq()` function and must be a supported external reference clock frequency per the

card specification. This function will update the expected clock frequency, however this configuration is runtime only and is not stored on the card nor permanent. Please note, this function will also automatically update the reference clock selection to an external reference clock source. The reference clock selection is also not stored on the card nor permanent. Changing the reference clock frequency using `skiq_write_ext_ref_clock_freq()` will stop any ongoing receiving or transmitting. Runtime reference clock frequency switching is only supported on Sidekiq Stretch and Sidekiq NV100 as of libsidekiq v4.17.0.

GPSDO

As of libsidekiq v4.15.0, the GPS Disciplined Oscillator (GPSDO) is available on Sidekiq Stretch when using FPGA bitstream v3.14.1 or later. GPSDO is available for Matchstiq Z3u as of libsidekiq v4.16.0. GPSDO is available for Sidekiq NV100 as of libsidekiq v4.17.0. Its functionality can be enabled with `skiq_gpsdo_enable()` (see *Sidekiq API* (page 68) for API functions related to the GPSDO). When a GPS fix has been obtained by the Sidekiq's on-board GPS, the FPGA uses the GPS unit's 1PPS signal to increase the accuracy of the Sidekiq's on-board oscillator by automatically adjusting the DAC warp voltage. If no GPS fix can be obtained or is lost, the DAC warp voltage is kept at its current value; if no GPS fix is available on startup, the warp voltage is kept at its factory calibrated default value. As the FPGA is now in control of the warp voltage, this prevents its manual adjustment through the `skiq_write_tcvcxo_warp_voltage()` API function. Additionally, there are some sensor peripherals that share a bus with the DAC warp voltage. As such, access to those sensors through API functions may, in some cases, indicate that data is not available (`-EAGAIN`). For example, in the case of Sidekiq Stretch, the temperature sensor measurement may not be available for up to one second after a call to `skiq_gpsdo_enable()` and a call to `skiq_read_temp()` will return `-EAGAIN` if called in that time period. The `skiq_gpsdo_is_locked()` function (available as of libsidekiq v4.17.0) queries the GPSDO control algorithm on the FPGA to check for a lock between the 1PPS signal and the disciplined oscillator.

Starting with FPGA bitstream v3.15.1, the GPSDO algorithm can use one of a few different 1PPS sources for disciplining for on-board reference clock. The GPSDO 1PPS source configuration matches the card's 1PPS source configuration and can be accessed or modified by calling `skiq_read_1pps_source()` or `skiq_write_1pps_source()` respectively. Please note that when `skiq_1pps_source_host` is selected, the GPSDO algorithm only uses the 1PPS when the GPS module additionally indicates a timing fix. If `skiq_1pps_source_external` is selected as the 1PPS source, then GPSDO algorithm uses the 1PPS unconditionally.

Note: The GPSDO algorithm is unique since its execution may persist after a libsidekiq application exits. Support for persistent execution was added to the Matchstiq Z3u and Sidekiq Stretch and Sidekiq NV100 as of libsidekiq v4.17.5. The algorithm resides in the FPGA and does not require software intervention to continue working. The side effect to this behavior is that previous libsidekiq releases (prior to v4.15.0) will not be able to read or write the DAC warp voltage nor, in the case of Sidekiq Stretch, read the temperature sensor measurements at all. However, applications linked against libsidekiq v4.15.0 or later will have alternate access to the DAC warp voltage and temperature sensors as described above.

5.4.17 Timestamp Details

There is both an RF timestamp and System timestamp maintained by the Sidekiq card. The RF timestamp for both receive and transmit are identical and increment at the rate of the sample rate. For each tick of the sample rate clock, the RF timestamp increments by one. For Sidekiq mPCIe, m.2, m.2-2280, Z2, NV100, and Matchstiq Z3u, the System timestamp increments independent of the sample rate. For Sidekiq X2 and Sidekiq X4, the System timestamp increments at a rate relative to the sample rate and continues to increment regardless of any radio configuration changes (with the exception of sample rate). The System timestamp frequency can be queried with the `skiq_read_parameters()` function, and the current frequency value is located in `sys_timestamp_freq` variable of the `skiq_fpga_param_t` data structure.

The timestamps can be reset to zero asynchronously via the `skiq_reset_timestamp()` function. This will reset both the RF and System timestamps. Additionally, if it is desired to set the timestamps to a specific value, the `skiq_update_timestamps()` function can be used. Finally, if it is desired to reset or update the timestamps on a 1PPS edge, then the `skiq_write_timestamp_reset_on_1pps()` or `skiq_write_timestamp_update_on_1pps()` functions can be used. This is useful if it is necessary for the application to have the timestamps synchronized to a 1PPS source.

5.4.18 Automatic Calibration

Automatic calibration may be enabled and performed by Sidekiq by default. Automatic calibration algorithms include DC offset reduction as well as quadrature error correction for both receive and transmit. Depending on the processing being performed by the user radio application, automatic calibration may not be desired.

As of libsidekiq v4.6.0, the ability to disable automatic transmit quadrature calibration algorithm is supported. Additionally, the ability to manually run the Tx quadrature calibration algorithm is supported. The Tx quadrature algorithm may take time to converge and may not be desired to leave in automatic mode. Additionally, execution of the Tx quadrature algorithm results in the appearance of erroneous transmissions in the spectrum while running. To configure the Tx quadrature calibration algorithm mode to run either manually or automatically, the `skiq_write_tx_quadcal_mode()` API can be used. The currently configured mode can be queried with the `skiq_read_tx_quadcal_mode()`. To manually initiate the calibration algorithm to run, the `skiq_run_tx_quadcal()` API can be used. Refer to the `tx_samples.c` application for the APIs associated with this.

As of libsidekiq v4.13.0, the ability to disable automatic receive calibration algorithms is supported with `skiq_write_rx_cal_mode()`. Additionally, the specific calibration types ran can be configured with `skiq_write_rx_cal_type_mask()`. The available calibrations that can be enabled can be queried with `skiq_read_rx_cal_types_avail()`. Finally, if it is desired to manually run the RX calibration, the `skiq_run_rx_cal()` API can be used. Refer to the `rx_samples.c` application for an example use of these APIs.

5.4.19 Receive Stream Mode

A typical use case for the Sidekiq line of products is to receive a great deal of I/Q data as efficiently as possible. This high throughput use case would historically receive 4,096 bytes per block.

As of libsidekiq v4.6.0, the low latency receive stream mode (`skiq_rx_stream_mode_low_latency`) provides a smaller block of I/Q samples from `skiq_receive()` more often and effectively lowers the latency from RF reception to host CPU. This is especially useful when using lower sample rates that would historically take a relative long time to fill up a 4kB I/Q block before delivering the samples to the host CPU and software application.

As of libsidekiq v4.7.0, the balanced stream mode (`skiq_rx_stream_mode_balanced`) is also available. The balanced stream mode is a compromise between the high throughput and low latency stream modes. It results in a reduced throughput relative to the optimized high throughput mode, but produces a larger number of samples per packet than the low latency mode, thus achieving a higher throughput than the low latency mode. Most applications interested in achieving the maximum throughput should use the default high throughput mode, which applications interested in having a minimal latency should use the low latency mode. The balanced mode offers a compromise between the low latency and high throughput options.

Refer to the Sidekiq API for additional details on the receive stream modes available. The API type and functions follow:

- Type: `skiq_rx_stream_mode_t`
- Function: `skiq_read_rx_stream_mode()`
- Function: `skiq_write_rx_stream_mode()`
- Function: `skiq_read_rx_block_size()`

5.4.20 Hotplug

As of libsidekiq v4.14.0, card hotplugging is now supported. Hotplug support will allow the user to connect and remove cards, both physically and logically, during an application's execution. Removing a card that is in use by the application will result in adverse outcomes. New cards will be listed in calls to the `skiq_get_cards()` function. Cards can then be initialized using `skiq_enable_cards()`.

5.4.21 Exiting

When an application is ready to exit, the `skiq_exit()` function should be called. This function ensures that libsidekiq shuts down gracefully. Any libsidekiq function should not be called within a signal handler, as there are various mutexes utilized to control access to libsidekiq and these mutexes may already be locked prior to being attempted to be called from within the context of the signal handler, which may result in deadlock. It is instead recommended to clear a "running" flag within the signal handler and perform the appropriate Sidekiq shutdown within the context of the main application. Refer to the `tx_samples.c` test application for an example of this.

Once `skiq_exit()` has been called, all follow-up calls to libsidekiq will fail with the exception of `skiq_init()` or `skiq_get_cards()`. Calling `skiq_init()` will re-initialize the library and prepare it for usage by a host application.

As of libsidekiq v4.14.0, `skiq_exit()` is called automatically when the application using the library shuts down; this is meant as a safety precaution to ensure that libsidekiq is properly cleaned up. If for some reason this is not desired, the exit handler can be disabled using the `skiq_set_exit_handler_state()` function. This function must be called before libsidekiq is initialized (for example, through `skiq_init()`). Despite the presence of this exit handler, `skiq_exit()` should still be explicitly called in libsidekiq applications to ensure that initialized radios (and the library) are cleaned up and powered down when no longer needed.

5.4.22 Critical Errors

It is possible that within libsidekiq, a critical error may be encountered. Once a critical error condition is detected in libsidekiq, it is no longer safe to continue accessing libsidekiq. The application should be shutdown and the cause of the error should be resolved. Continued operation on the Sidekiq may result in unpredictable / incorrect behavior. The default behavior of libsidekiq when encountering a critical error is to exit. If an application wishes to override the default behavior, a callback function can be registered via the `skiq_register_critical_error_callback()` API. The function registered is then called if libsidekiq encounters a critical error. Note that continued access of libsidekiq after a critical error has occurred can result in undefined behavior and should not be done.

5.5 Using Libsidekiq Remotely

Introduced in libsidekiq v4.16.0 is a mechanism for running libsidekiq applications on a client which connects to a server supported by the network transport interface. Both the Sidekiq Z2 and Matchstiq Z3u support this functionality. The server physically contains the Sidekiq card. An overview of the deployment of such a system is depicted below.

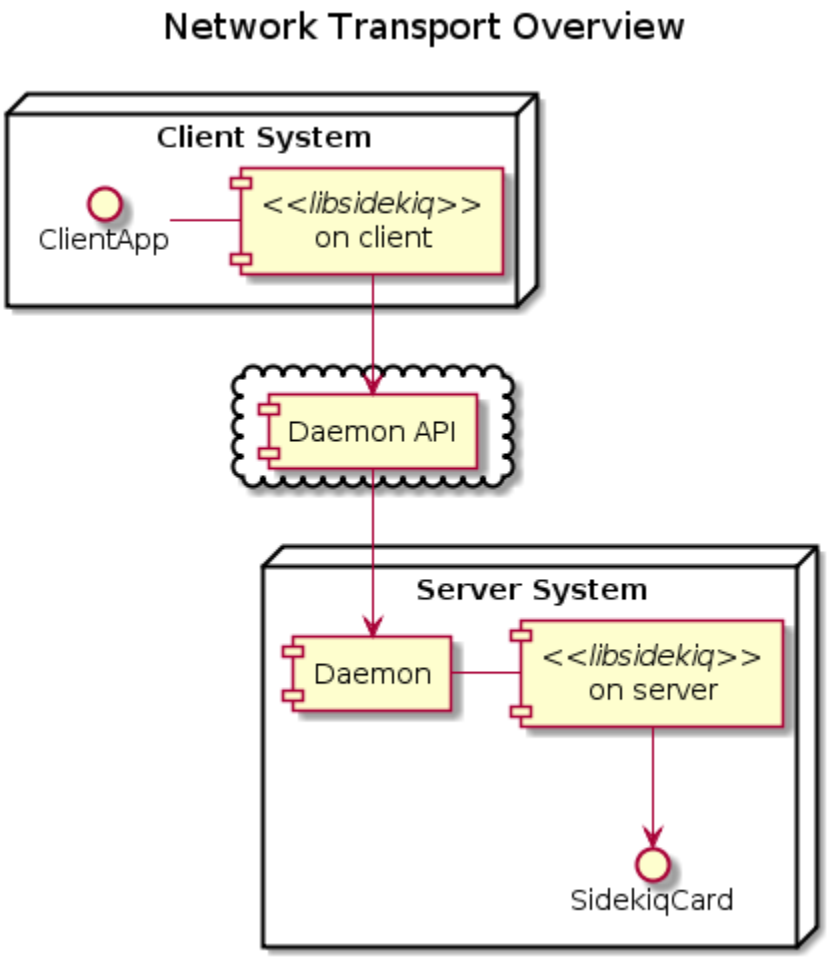

Fig. 5.3: Sidekiq Remote Deployment

The server containing the Sidekiq card allows for applications to be run either remotely or locally on the server itself. Management of the Sidekiq card resource is handled by the daemon. Client(s) communicate with a server via a network interface, using a daemon API to send and receive messages.

5.5.1 Prerequisites

netconfig

After the integration of libtirpc into libsidekiq v4.17.5, network transport now requires a `netconfig` file be present on the host. A default version of the contents of the `netconfig` file are provided below if the host operating system does not include it. Since each operating system may handle this file differently, please contact Epiq Solutions for installation instructions if needed.

Note: In a future release of libsidekiq a patch will be released to remove this requirement for using network transport.

Matchstiq Z3u

The *z3u-usb-net-* package must be at least at least version *1.0.0-1*. To view the currently installed *z3u-usb-net* package installed on the Z3u, run `apt list z3u-usb-net`.

In order to decrease the overall system latency and improve the streaming throughput, it is highly recommended to use the Ethernet Emulation Model (EEM) USB networking protocol. To configure the USB network to use this profile, run `sudo fw_setenv usb_network_profile eem` on the Z3u (refer to the "USB Network Configuration" section of the Matchstiq Z3u Hardware User Manual). Note that after updating the profile, the system must be rebooted for the settings to be applied.

Sidekiq Z2

A minimum BSP of v3.3.0 is required.

Client

Libsidekiq utilizes Open Network Computing (ONC) Remote Procedure Calls (RPC) to provide acceleration of a handful of libsidekiq functions. As a result, the client system most have RPC bind installed. For instance, on a Debian machine (such as Ubuntu 18.04), you can run `sudo apt install rpcbind`.

Only Linux on a x86-64 bit processor is currently supported. For support of alternative operating system support, please contact Epiq Solutions.

5.5.2 Setup

Client System

Connecting to a remote Sidekiq card requires setting environment variables **SKIQ_DAEMON_IP** and **SKIQ_DAEMON_PORT** with the server's IP address and port (default port of 3417).

Server System

The server must be running the daemon process (*skiq_daemon*) prior to accepting client connections. The *skiq_daemon* test application is provided for both the Sidekiq Z2 and the Matchstiq Z3u in the prebuilt applications bundled with a Sidekiq release.

Matchstiq Z3u

For the Matchstiq Z3u, a Debian package, *z3u-skiq-daemon* is provided. This package includes the *skiq_daemon* application as well as a service that is enabled to run automatically upon startup once the package is installed. Note that the card ownership is managed by *skiq_daemon* such that running *skiq_daemon* will not prevent applications from running natively on the Matchstiq Z3u if client does not have the Matchstiq Z3u card enabled. If it is desired to disable *skiq_daemon* from running automatically upon startup, this can be disabled by running `sudo systemctl disable z3u-skiq-daemon` on the Z3u.

5.5.3 Example Usage

Let's consider an example deployment of running a Matchstiq Z3u connected to a Linux laptop. The Linux laptop will be running an x86-64 bit version of libsidekiq.

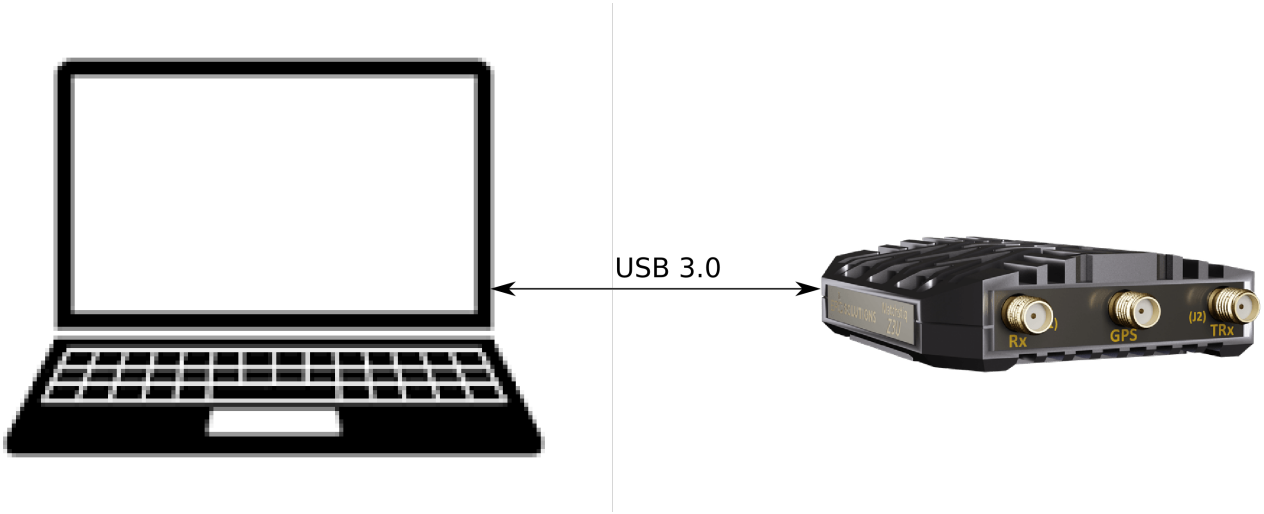

Fig. 5.4: Sidekiq Remote Example Usage

Since the Matchstiq Z3u automatically starts *skiq_daemon* when powered on, there is no additional steps required to setup the Matchstiq Z3u.

On the Linux laptop, we must configure the daemon IP address and port number. By default, the Matchstiq Z3u's IP address is 192.168.0.15. Additionally, the default port number of *skiq_daemon* is 3417. To configure the IP address and port number, on the laptop, we can run:

```
$ export SKIQ_DAEMON_IP=192.168.0.15
$ export SKIQ_DAEMON_PORT=3417
```

Now, also on the Linux laptop, we can run the *version_test* application.

```
$ ./version_test
1 card(s) found: 0 in use, 1 available!
Card IDs currently used     :
Card IDs currently available: 0
Info: initializing 1 card(s)...
SKIQ[18197]: <INFO> libsidekiq v4.16.0 (gXXXXXXXXX)
version_test[18197]: <INFO> Sidekiq card 0 is serial number=9X0J, Z3U (rev B) (part ES032201-B0-00)
version_test[18197]: <DEBUG> got port 3417, ip 192.168.0.15 card=0
version_test[18197]: <DEBUG> Allocating new hash table
version_test[18197]: <DEBUG> Allocating new connection 5620507011113330880
version_test[18197]: <DEBUG> Creating new RPC client
version_test[18197]: <DEBUG> Saving server card 0 (uid=5620507011113330880) to client card 0
version_test[18197]: <WARNING> FPGA capabilities indicate no support for reading/writing flash for card 0
version_test[18197]: <INFO> Sidekiq card 0 FPGA v3.14.0, (date 20081217, FIFO size unknown)
version_test[18197]: <INFO> Sidekiq card 0 is configured for an internal reference clock
************************************************************
* libsidekiq v4.16.0
************************************************************
************************************************************
```

(continues on next page)

(continued from previous page)

```
* Sidekiq Card 0
Card
    accelerometer present: true
    part type: Z3U
    part info: ES032201-B0-00
    serial: 9X0J
    xport: network
FPGA
    version: 3.14.0
    git hash: 0x1eefb308
    build date (yymmddhh): 20081217
    tx fifo size: unknown
RF
    reference clock: internal
    reference clock frequency: 40000000 Hz

version_test[18197]: <INFO> Unlocking card 0
version_test[18197]: <DEBUG> Hash table empty, destroying...
```

We can see that our remotely connected Matchstiq Z3u was detected by the *version_test* application running on the Linux x86 host.

5.5.4 Detailed Network and Port Use

The *skiq_daemon* application uses both TCP and UDP sockets for use. The probe socket exists on port number 3417 and can be overwritten by launching the *skiq_daemon* with a different port. For example, if the probe port number is desired to be 4000, you can start the *skiq_daemon* application with: `./skiq_daemon -p 4000`. The client will need to specify the daemon port as **SKIQ_DAEMON_PORT** as 4000 (`export SKIQ_DAEMON_PORT=4000`). When a card is in use by libsidekiq, a TCP socket connection is established between the client and server. The port number used can vary and typically begins from the **SKIQ_DAEMON_PORT**. The port in use for the control is established during the probe procedure of the card. Additionally, a TCP and UDP socket connection are established to support RPC. The port number in use to support RPC varies. Finally, when streaming samples, a UDP socket is used to stream the sample data. The port number used in streaming can vary and typically begins from the **SKIQ_DAEMON_PORT**.

5.5.5 Limited Capabilities

The following features / capabilities are not supported in libsidekiq v4.17.4 via remote operation using the network transport:

- reprogramming the RFIC via *skiq_prog_rfic_from_file()*
- reprogramming the FPGA via *skiq_prog_fpga_from_file()*
- **accessing the flash via**
  - *skiq_verify_fpga_config_from_flash()*
  - *skiq_save_fpga_config_to_flash_slot()*
  - *skiq_verify_fpga_config_in_flash_slot()*
- **frequency hopping via**
  - *skiq_write_rx_freq_tune_mode()*
  - *skiq_read_rx_freq_tune_mode()*

  - *skiq_write_tx_freq_tune_mode()*
  - *skiq_read_tx_freq_tune_mode()*
  - *skiq_write_rx_freq_hop_list()*
  - *skiq_read_rx_freq_hop_list()*
  - *skiq_write_tx_freq_hop_list()*
  - *skiq_read_tx_freq_hop_list()*
  - *skiq_write_next_rx_freq_hop()*
  - *skiq_write_next_tx_freq_hop()*
  - *skiq_perform_rx_freq_hop()*
  - *skiq_perform_tx_freq_hop()*
  - *skiq_read_curr_rx_freq_hop()*
  - *skiq_read_curr_tx_freq_hop()*
  - *skiq_read_next_rx_freq_hop()*
  - *skiq_read_next_tx_freq_hop()*
- **RX streaming on 1PPS or a trigger source via**
  - *skiq_start_rx_streaming_on_1pps()*
  - *skiq_start_rx_streaming_multi_on_trigger()*
  - *skiq_stop_rx_streaming_on_1pps()*
- **RX stream configuration via**
  - *skiq_set_rx_transfer_timeout()*
  - *skiq_get_rx_transfer_timeout()*
  - *skiq_write_rx_stream_mode()*
  - *skiq_read_rx_stream_mode()*
- Only a single remote card can be used
- Only a single handle can be used
- RF port selection is not supported on Z2

5.6 Configuring Sample Rate / Channel Bandwidth

5.6.1 API Ordering Dependency

For all platforms, it's recommended to configure the sample rate before tuning the carrier or LO frequency. Failure to do so may result in an invalid request for LO tuning due to the relationship between LO tune range and sample rate. For example, tuning the LO frequency to 50MHz and then requesting a sample rate of 100Msps would result in error:

```
Info: configured Rx LO freq to 50000000 Hz
Error: failed to set Rx sample rate or bandwidth(using default from last config file)...status is -22
```

5.6.2 Configuring Sample Rate / Channel Bandwidth on Multiple Handles

Libsidekiq offers multiple API functions to configure the receive sample rate and bandwidth, such as `skiq_write_rx_sample_rate_and_bandwidth()` and `skiq_write_rx_sample_rate_and_bandwidth_multi()`. When configuring multiple receive handles, the `skiq_write_rx_sample_rate_and_bandwidth_multi()` function is preferred as it is more performant than calling `skiq_write_rx_sample_rate_and_bandwidth()` multiple times.

5.6.3 Sidekiq mPCIe, m.2, Stretch (m.2-2280), Z2, and Matchstiq Z3u only

This section describes the process of configuring the channel bandwidth in the Analog Devices AD9361/4 RFIC using predefined FIR filter coefficients, the dependency of sample rate on bandwidth, and the additional filter configuration settings used for a specific sample rate. It does not cover how the FIR is programmed but rather captures the selection criteria and characteristics of the FIR coefficients. The same procedure is for both configuration the receive channel bandwidth as well as the transmit channel bandwidth. The receive and transmit channel bandwidths can be configured independently, but the sample rate can not.

The requested channel bandwidth along with the sample rate is used to select and configure the digital FIR coefficients. The ratio of requested channel bandwidth to sample rate is used to calculate the desired passband percentage of the FIR filter. If the desired passband percentage does not match any of the passband percentages of the precomputed FIR filters, the precomputed FIR filter with the next incremented step in the passband percentage is selected. In other words, if precomputed FIR coefficients in increments of 10% are available and a passband percentage of 65% is desired, then the FIR coefficients resulting in a passband of 70% would be selected.

Sample Rate and FIR Selection

As mentioned in the overview, the selection of the FIR filter coefficients on the percentage of requested channel bandwidth to sample rate. Specifically, the passband percentage is computed as

$$passband\_percent = (bandwidth/sample\_rate) * 100$$

In addition to the calculation of the passband percentage, the sample rate also impacts decimation (or interpolation) factor built in to the FIR filter stage. The specific decimation / interpolation factor of the FIR as it relates to sample rate is shown in *Decimation / Interpolation Factor* (page 56).

Table 5.5: Decimation / Interpolation Factor

Sample Rate (in samples/sec)	FIR Decimation / Interpolation	Filter Lineup	Total Decimation/Interpolation
233000-13300000	4*	RX/TX DEC3/INT3 enabled RX/TX HB2 enabled RX HB1 enabled	RX 48, TX 24
13300000-23000000	2	RX/TX DEC3/INT3 enabled RX/TX HB2 enabled RX HB1 enabled	RX 24, TX 12
23000000-40000000	2	RX/TX HB3 enabled RX/TX HB2 enabled RX HB1 enabled	RX 16, TX 8
40000000-46000000	2	RX/TX DEC3/INT3 enabled RX HB2 enabled	RX 12, TX 6
46000000-61440000	2	RX/TX HB3 enabled RX HB2 enabled	RX 8, TX 4

Note: When the TX FIR interpolation is configured to a value of 4, the TX FIR coefficients are automatically double relative to other settings. This ensures a consistent output power level when moving from interpolation settings of 4 to 2 (ex. changing sample rates from <13.3Msps to >13.3Msps).

Since the decimation factor is part of the FIR filter configuration, the actual passband percentage is impacted by the decimation factor. Specifically, the actual passband percentage of a specific filter is

$$actual\_passband = filter\_passband * decimation\_factor$$

The impact of the decimation factor is accounted for when selecting the predefined FIR filter based on the desired channel bandwidth.

Number of Filter Taps

The number of taps available is limited by the RF IC and the configured sample rate. The RF IC can calculate 16 taps per clock cycle, and depending on the sample rate, the resulting number of taps available for the FIR is either 64, 96, or 128. The number of taps used for specific sample rates are shown in *Number of Filter Taps* (page 57).

Table 5.6: Number of Filter Taps

Sample Rate (in samples/sec)	# of RX Filter Taps	# of TX Filter Taps
233000 – 40000000	128	128
40000000 – 46000000	96	96
46000000 – 61440000	64	64

Filter Selection Example

Suppose a sample rate of 10Msps and a channel bandwidth of 6.5MHz is requested.

$$passband\_percent = (6500000/10000000) * 100 = 65$$

With a sample rate of 10Msps, the decimation factor is 4. In order to achieve an actual passband of 6.5MHz, the FIR filter passband is calculated as

$$\begin{aligned} filter\_passband &= passband\_percent/decimation\_factor \\ &= 65/4 \\ &= 16.25 \end{aligned}$$

However, there is not a FIR filter with a passband of 16.25. As a result, the filter selected is the next one available, greater than the requested passband. In this case, a filter passband of 16.5 is selected.

Additionally, since the sample rate is <40Msps, the maximum of 128 filter taps can be used. Therefore, the filter used in this case is `fir_128_tap_165_passband`. The actual resulting bandwidth is calculated as follows

$$\begin{aligned} actual\_passband &= filter\_passband * decimation\_factor * sample\_rate \\ &= 16.5 * 4 * 10000000 \\ &= 6.6\text{MHz} \end{aligned}$$

Custom Filter Coefficients

The ability to load custom FIR coefficients is supported through the `skiq_write_rfic_rx_fir_coeffs()` / `skiq_write_rfic_tx_fir_coeffs()` API functions. The decimation (interpolation) factor of the FIR and the

number of taps available is determined by the sample rate and cannot be changed. Therefore, the sample rate must be configured prior to the custom FIR configuration. The FIR configuration parameters can be queried via the `skiq_read_rfic_rx_fir_config()` / `skiq_read_rfic_tx_fir_config()` APIs. If custom FIR coefficients are used, the bandwidth reported via `skiq_read_rx_sample_rate_and_bandwidth()` / `skiq_read_tx_sample_rate_and_bandwidth()` are no longer valid.

Filter Passband Available

The FIR filter passbands currently available are summarized in *Available Filters* (page 58). Note the FIR passband listed is for a decimation factor of 1.

Table 5.7: Available Filters

Start Passband	End Passband	Step Size	Number of Filters
0.1%	0.1%	N/A	1
0.5%	50%	0.5%	100

Analog Filtering

In addition to the filtering described above, Sidekiqs based on the AD9361/4 RFIC allow users to configure analog filters present on the Rx/Tx paths. These low pass filters are located right before the ADC for the Rx path and after the DAC for the Tx path. By default, the analog filter bandwidth is automatically configured via `skiq_write_rx_sample_rate_and_bandwidth()`, or `skiq_write_tx_sample_rate_and_bandwidth()`, but users may override the configuration by calling `skiq_write_rx_analog_filter_bandwidth()`, or `skiq_write_tx_analog_filter_bandwidth()` API functions after configuring the sample rate and bandwidth.

Sidekiq mPCIe and Matchstiq Z3u

When operating in dual channel mode, the maximum supported sample rate of Sidekiq mPCIe and Matchstiq Z3u is limited to 30.72 Msps. In single channel mode, the full maximum sample rate of 61.44 Msps is supported.

5.6.4 Sidekiq X2 and X4

The Sidekiq X2 and X4 RF transceivers support a wide variety of sample rate and bandwidth combinations out of the box with libsidekiq. The following tables list the rates that are directly achievable and can be configured with the libsidekiq API. In addition to the provided rates, both decimation and user generated profiles can be utilized to meet specific application needs. Please contact Epiq Solutions via the support forums *[5]* (page 8) if there is more information required regarding capability and configuration of sample rates / bandwidth.

Built-in Profiles

The next subsections capture the sample rate and bandwidth settings supported as of libsidekiq v4.12.0. These profiles and configuration parameters were generated using Analog Devices' AD9371/AD9379 Filter Wizard.

Sidekiq X2 Built-in Profiles

The following two tables show the full list of available receive and transmit sample rates available in libsidekiq v4.12.0 for a Sidekiq X2. Some sample rates have been available in previous releases and are noted as such.

Table 5.8: Sidekiq X2 Receive Sample Rates

RX Input Rate (Msps)	RF Bandwidth (MHz)	Supported Handle(s)	Since
245.76	200	B1	v4.10.0
245.76	100	B1	v4.10.0
153.6	100	A1/A2/B1	v4.10.0
122.88	100	A1/A2/B1	v4.10.0
100	82	A1/A2/B1	v4.10.0
73.728	60.456 / 30.228	A1/A2/B1	v4.10.0
61.44	50	A1/A2/B1	v4.10.0
61.44	25	A1/A2/B1	v4.10.0
50	41	A1/A2/B1	v4.10.0
36.864	30.228	A1/A2	v4.10.0
30.72	25 / 20 / 18	A1/A2	v4.10.0

Table 5.9: Sidekiq X2 Transmit Sample Rates

Tx Output Rate (Msps)	RF Bandwidth (MHz)	Supported Handle(s)	Since
153.6	100	A1/A2	v4.10.0
122.88	100	A1/A2	v4.10.0
100	82	A1/A2	v4.10.0
73.728	60.456	A1/A2	v4.10.0
61.44	50	A1/A2	v4.10.0
50	41	A1/A2	v4.10.0

Sidekiq X4 Built-in Profiles

The following two tables show the full list of available receive and transmit sample rates available in libsidekiq v4.12.0 for a Sidekiq X4. Some sample rates have been available in previous releases and are noted as such.

Table 5.10: Sidekiq X4 Receive Sample Rates

Rx Input Rate (Msps)	RF Bandwidth (MHz)	Supported Handle(s)	Since
500	450 / 400	C1/D1	v4.11.1
491.52	450 / 400	C1/D1	v4.11.1
250	200	A1/A2/B1/B2/C1/D1	v4.10.0
250	100	A1/A2/B1/B2/C1/D1	v4.11.1
245.76	200	A1/A2/B1/B2/C1/D1	v4.9.0
245.76	100	A1/A2/B1/B2/C1/D1	v4.11.1
200	164	A1/A2/B1/B2/C1/D1	v4.12.0
153.6	100	A1/A2/B1/B2/C1/D1	v4.11.1
122.88	100	A1/A2/B1/B2/C1/D1	v4.9.0
122.88	72 / 64 / 61.44	A1/A2/B1/B2/C1/D1	v4.11.1
100	82	A1/A2/B1/B2/C1/D1	v4.9.0
76.8	30.72	A1/A2/B1/B2/C1/D1	v4.11.1
73.728	60.456 / 30.228	A1/A2/B1/B2/C1/D1	v4.11.1
61.44	50	A1/A2/B1/B2/C1/D1	v4.9.0
61.44	25	A1/A2/B1/B2/C1/D1	v4.11.1
50	41	A1/A2/B1/B2/C1/D1	v4.10.1
50	20	A1/A2/B1/B2/C1/D1	v4.11.1

Table 5.11: Sidekiq X4 Transmit Sample Rates

Tx Output Rate (Msps)	RF Bandwidth (MHz)	Supported Handle(s)[1]	Since
500	450 / 400	A1/A2/B1/B2	v4.12.0
491.52	450 / 400	A1/A2/B1/B2	v4.12.0
250	200	A1/A2/B1/B2	v4.10.0
250	100	A1/A2/B1/B2	v4.11.1
245.76	200	A1/A2/B1/B2	v4.9.0
245.76	100	A1/A2/B1/B2	v4.11.1
200	164	A1/A2/B1/B2	v4.12.0
153.6	100	A1/A2/B1/B2	v4.11.1
122.88	100	A1/A2/B1/B2	v4.9.0
122.88	72 / 64 / 61.44	A1/A2/B1/B2	v4.11.1
100	82	A1/A2/B1/B2	v4.9.0
76.8	30.72	A1/A2/B1/B2	v4.11.1
73.728	60.456 / 30.228	A1/A2/B1/B2	v4.11.1
61.44	50	A1/A2/B1/B2	v4.9.0
61.44	25	A1/A2/B1/B2	v4.11.1
50	41	A1/A2/B1/B2	v4.10.1
50	20	A1/A2/B1/B2	v4.11.1

Note: Transmit only applications at rates >= 250Msps additionally requires configuring RxC1 or RxD1 to the desired sample rate.

[1] Streaming transmit samples over PCIe is only available for handle A1 or handle pairs A1/A2 and A1/B1

Options for Sample Decimation

Starting with libsidekiq **v4.10.0** and FPGA bitstream **v3.12.0**, both Sidekiq X2 and Sidekiq X4 have build options that allow the use of sample decimation. The build options are defined in more detail in each product's respective PDK documentation. The use of the decimator happens automatically when a receive sample rate is configured by the user. The sample rate selection algorithm attempts to find the best profile and decimation settings to meet the requested sample rate and RF bandwidth. The equivalent RF bandwidth after the decimation stage is the minimum of the RF bandwidth before the stage and the decimated sample rate. So an RF bandwidth of 40MHz for a signal sampled at 50Msps, will have an equivalent RF bandwidth of 25MHz after one stage of decimation since ($50 / 2 < 40$).

The default build configuration for the decimation offers the functionality on RxA1 and RxA2 with up to 6 stages (i.e. decimate by 64) available. For example, if 50 Msamples/sec is available and the receive handle supports decimation of $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$, $\frac{1}{16}$, $\frac{1}{32}$, and $\frac{1}{64}$, then 25, 12.5, 6.25, 3.125, 1.5625, and 0.78125 Msps are also available.

Runtime Loaded User Generated Profiles

In addition to the built-in profiles, X2 and X4 users can create custom profiles using the Analog Devices profile generation tool. At the time of the libsidekiq v4.17.0 release, there is no support for generating or using custom profiles with Sidekiq NV100. The following sections outline the procedure for creating and importing a user-generated profile at runtime.

Determine Dev Clock Settings for ADI Profile Tool

Use the Sidekiq sample rate utility `test_sample_rate` test application (as shown below) to determine the device clock settings. Below is an example of running the application with a desired sample rate of 73.728 MHz.

```
$ /tmp/test_sample_rate -c 1 -r 73728000
Info: initializing card 1...
SKIQ[32448]: <INFO> libsidekiq 4.8.255-dev (g684832db)
test_sample_rate[32448]: <INFO> Sidekiq card 1 is serial number=7T25, hardware reserved (rev 1), product␣
↪reserved (X2) (part ES020201-B2-00)
test_sample_rate[32448]: <INFO> Sidekiq card 1 FPGA v3.11.0, (date 18121917, FIFO size 64k)
test_sample_rate[32448]: <INFO> Sidekiq card 1 is configured for host reference clock
test_sample_rate[32448]: <INFO> Loading calibration data for Sidekiq X2, card 1
Info: exact sample rate requested (73728000 Hz) is possible!
Debug: VCXO 153600000 FPGA div 2

Info: Use dev clock frequency 147456000 with a divider of 2

test_sample_rate[32448]: <INFO> unlocking card 1
```

Create New Profile

Once the appropriate Dev Clock for Sidekiq has been determined, the AD9371 (Sidekiq X2) or ADRV9009 (Sidekiq X4) Filter Wizard can be used to generate a profile that can be loaded as outlined below.

1. Launch the Filter Wizard tool (note: v1.10 is the currently supported version for X2, v2.4 for X4).
2. If using Sidekiq X4, set the `Part Number` field to ADRV9009 (n/a for X2).
3. Configure the `Tx Profile`, `ORx Profile` (`skiq_rx_hdl_B1` for X2), `Rx Profile`, and `SnRx Profile` fields to the desired sample rate and filter characteristics. Currently, RX / ORx / TX sample rates must match. The below example uses a sample rate of 73.728 Msps.

4. Press the `Generate Profiles` button and ensure that the profiles are valid as indicated by the status bar.
5. Configure the `Ref Clock Divider` and `Device Clock (MHz)` settings to the values generated by the `test_sample_rate` application. These fields are located in the bottom left hand side of the main screen. In our example, the `Ref Clock Divider` is set to `2` and the `Device Clock` is set to `147.456 MHz`.
6. Generate the profile file with the *Output Profiles to File* button.

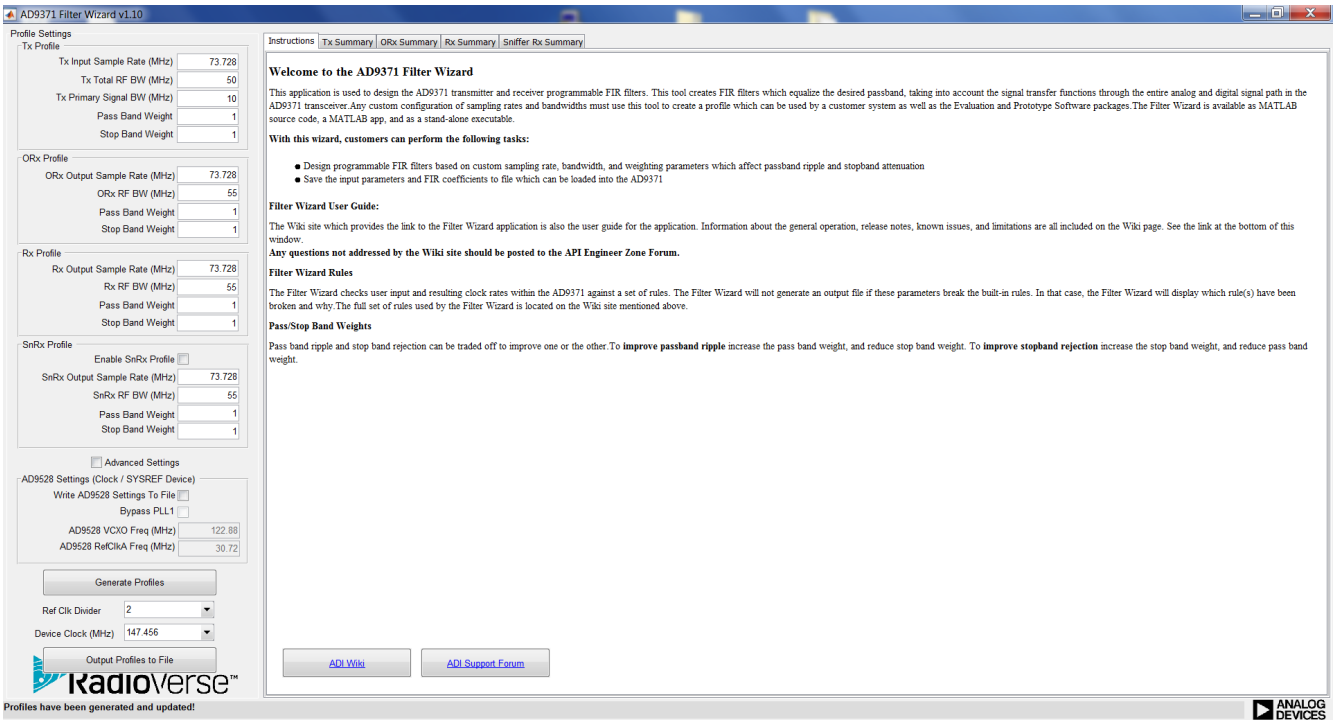

Fig. 5.5: Analog Devices Profile Generator (Sidekiq X2)

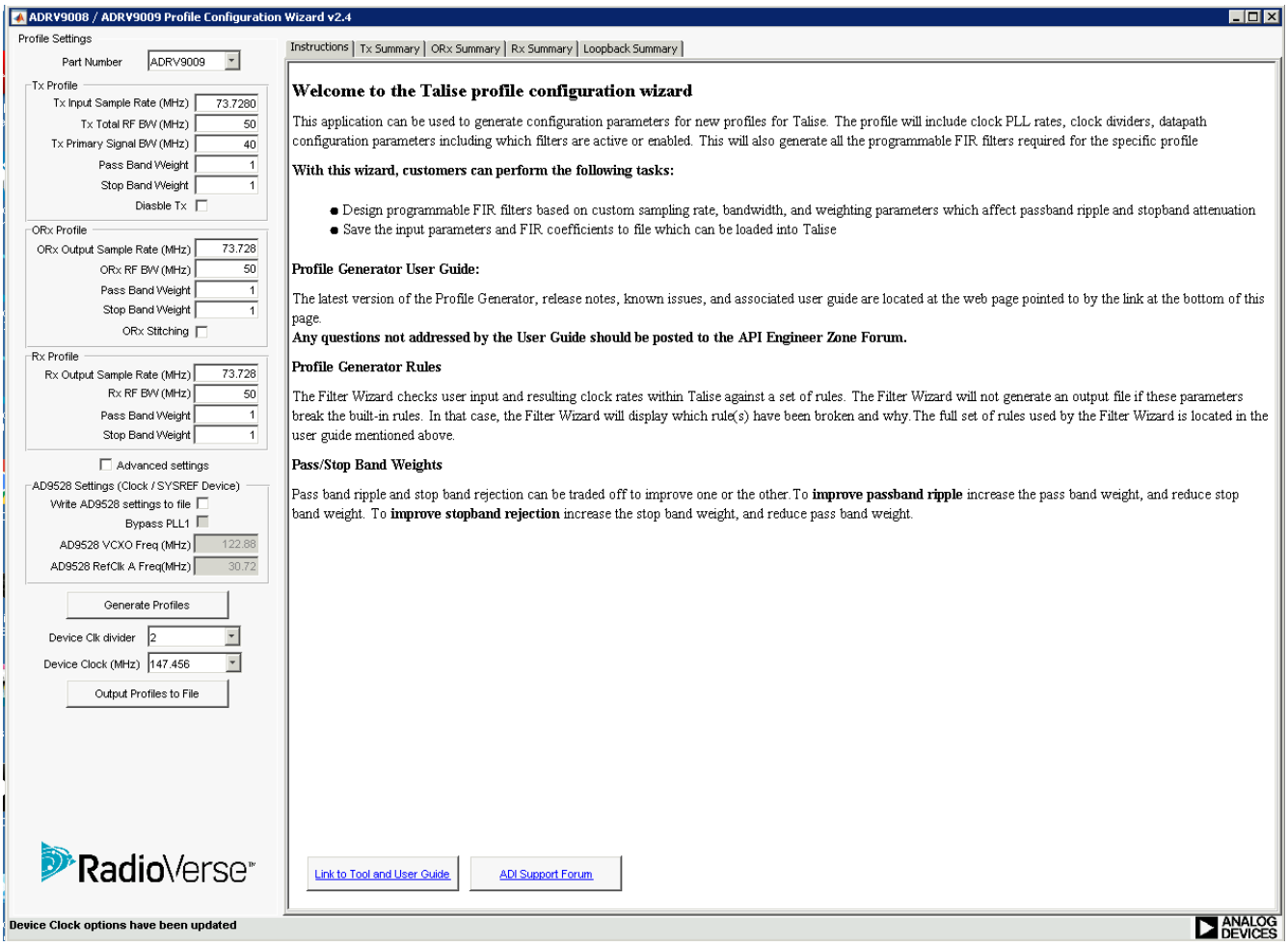

Fig. 5.6: Analog Devices Profile Generator (Sidekiq X4)

Configure Sidekiq with Profile

To configure the radio with the profile created, after performing the initialization and default configuration with the `skiq_init()` API, the `skiq_prog_rfic_from_file()` API. This configures the radio with the sample rate defined in the profile as well as configuring the various filter settings. This process results in a full reset and re-initialization of the radio, so any previous radio configuration (outside of the sample rate and bandwidth) must be applied after configuring from the profile. It is recommended to perform the programming from the file immediately after initialization, prior to any radio configuration. Note that modifying the sample rate and/or bandwidth settings with the `skiq_write_rx_sample_rate_and_bandwidth()` or `skiq_write_tx_sample_rate_and_bandwidth()` will result any previously loaded profile to be overwritten.

5.6.5 Sidekiq NV100

The Sidekiq NV100 RF transceivers support a wide variety of sample rate and bandwidth combinations out of the box with libsidekiq. The following table lists the rates that are directly achievable and can be configured with the libsidekiq API. Even though the NV100 has two receivers and two transmitters, there are clocking constraints between handles that restrict sample rate configuration. As of libsidekiq v4.17.0, both receivers must be configured to the same sample rate. The same restriction applies to both transmitters.

Due to limitations of the interpolation/decimation rate supported in the datapath, the Sidekiq NV100 (using the ADRV9004 RFIC) has a list of "dead" zones in which certain sample rates cannot be configured. Please see the "Dead Zone Frequency Ranges" table of the ADRV9001 User Guide *[15]* (page 8) for more information. Please contact Epiq Solutions via the support forums *[5]* (page 8) if there are questions regarding capability and configuration of sample rates and RF bandwidths. Users may also request additional sample rates via the support forums.

Sidekiq NV100 Built-in Profiles

The following table shows the full list of available receive and transmit sample rates available in libsidekiq v4.17.0 for a Sidekiq NV100. If a sample rate is listed here, it is available for both receive and transmit paths. At this time, all receivers and transmitters are configured for the same sample rate (unless the user is configuring from the list of disparate rates). However, the RF receive bandwidth is configurable per-handle between 5% and 80% in 0.5% increments.

Table 5.12: Sidekiq NV100 Sample Rates

Rx/Tx Sample Rate (Msps)
0.541667
1.92
2.4576
2.8
3.84
4
4.9152
5.6
7.68
9.8304
10
11.2
15.36
16
20
21.6667
22
23.04
30.72
40
61.44

The following table shows the full list of disparate sample rates available in libsidekiq v4.17.0 for a Sidekiq NV100. If a sample rate combination is listed there, the receivers can be configured at the input rate while the transmitters can be configured at the output rate.

Table 5.13: Sidekiq NV100 Sample Rates

Input Rate (Msps)	Output Rate (Msps)
1.4	5.6
11.2	5.6

The RF receive bandwidth is configurable per-handle between 5% and 80% in 0.5% increments with additional values above 80% listed in the subsequent table.

Table 5.14: Sidekiq NV100 Supported Rx Bandwidth Percentages

Bandwidth % of Sample Rate
3
5
5.5
.
. (Between 5 and 80% possible in 0.5% increments)
.
75.5
80
86
89
95
96
99

5.7 Example X4 Use Cases: Rx

This section outlines common use cases possible with Sidekiq X4, demonstrating how they can be achieved with the supplied test applications. For users developing an application from scratch, the equivalent sequence of API calls is provided. See the `rx_samples_minimal.c` and `rx_samples.c` sample applications in the `test_apps/` directory for example usage.

Caution: Receiving at high sample rates can be RAM & disk intensive. Performance is dependent on the host platform.

5.7.1 Receive: single channel, up to 200MHz IBW

```
test_apps/rx_samples_minimal -c 0 -d /tmp/iq_output -f 850000000 -r 245760000 -b 200000000 --handle=A1
```

- where handle can be any of the following: `A1,A2,B1,B2,C1,D1`

Note: libsidekiq will use the provided sample rate and bandwidth arguments to select the RFIC profile best meeting the constraints. If the selected rate/bandwidth is not an exact match, it may be desirable to define and import a custom profile. Please see the *Runtime Loaded User Generated Profiles* (page 61) section for more details.

The same can be achieved using the following sequence of API functions:

```
skiq_init()
skiq_write_rx_sample_rate_and_bandwidth(card, handle, rate, bandwidth)
skiq_write_rx_LO_freq(card, handle, frequency)
skiq_start_rx_streaming_multi_immediate()
while(receiving)
   skiq_receive(card, handle, rx_block, len)
skiq_stop_rx_streaming_multi_immediate()
skiq_exit()
```

5.7.2 Receive: single channel, up to 400MHz IBW

```
test_apps/rx_samples_minimal -c 0 -d /tmp/iq_output -f 850000000 -r 491520000 -b 400000000 --handle=C1
```

- where handle can be any of the following: `C1,D1`

The same can be achieved using the following sequence of API functions:

```
skiq_init()
skiq_write_rx_sample_rate_and_bandwidth(card, handle, rate, bandwidth)
skiq_write_rx_LO_freq(card, handle, frequency)
skiq_start_rx_streaming_multi_immediate()
while(receiving)
    skiq_receive(card, handle, rx_block, len)
skiq_stop_rx_streaming_multi_immediate()
skiq_exit()
```

5.7.3 Receive: Two phase coherent channels up to 200MHz IBW

```
test_apps/rx_samples_minimal -c 0 -d /tmp/iq_output -f 850000000,850000000 -r 250000000,250000000 -b
200000000,200000000 --handle=A1,A2
```

- where handle can be any two of the following: `A1,A2,B1,B2`

The same can be achieved using the following sequence of API functions:

```
skiq_init()
skiq_write_rx_sample_rate_and_bandwidth_multi(card, handles, nr_handles, rates, bandwidths)
skiq_write_rx_LO_freq(card, handle #1)
skiq_write_rx_LO_freq(card, handle #2)
skiq_start_rx_streaming_multi_immediate()
while(receiving)
   skiq_receive(card, handle, rx_block, len)
skiq_stop_rx_streaming_multi_immediate()
skiq_exit()
```

Caution: Phase coherent receive cannot be used in the conjunction with the decimator. A warning will be issued when this condition is encountered, but please be aware of the limitation.

5.7.4 Receive: Two independently tunable channels different sample rate* up to 200MHz IBW

```
test_apps/rx_samples_minimal -c 0 -d /tmp/iq_output -f 750000000,850000000 -r 250000000,250000000 -b
61440000,122880000 --handle=A1,B2
```

- where handle can be two of the following: `A1,A2,B1,B2`, with the constraint that one handle must be from the "A" group and one from "B".

The same can be achieved using the following sequence of API functions:

```
skiq_init
skiq_write_rx_sample_rate_and_bandwidth_multi(card, handles, nr_handles, rates, bandwidths)
skiq_write_rx_LO_freq(card, handle #1)
skiq_write_rx_LO_freq(card, handle #2)
skiq_start_rx_streaming_multi_immediate()
```

(continues on next page)

(continued from previous page)

```
 while(receiving)
   skiq_receive(card, handle, rx_block, len)
skiq_stop_rx_streaming_multi_immediate()
skiq_exit()
```

Note: * Different sample rates can be used per handle, but with the following caveats:

1. The lower rate must be on a handle supporting decimation. (`A1 or A2`)
2. Rates must be < 153.6Msps
3. The lower rate must be a factor of the higher rate

Caution: Phase coherent receive cannot be used in this scenario since decimation is active.

5.8 Example NV100 Use Cases: Rx

This section outlines common use cases possible with Sidekiq NV100, demonstrating how they can be achieved with the supplied test applications. For users developing an application from scratch, the equivalent sequence of API calls is provided. See the `rx_samples_minimal.c` and `rx_samples.c` sample applications in the `test_apps/` directory for example usage.

Caution: Receiving at high sample rates can be RAM & disk intensive. Performance is dependent on the host platform.

5.8.1 Receive: single channel, up to 50MHz IBW

The following example captures samples at 61.44Msps with a bandwidth of 50MHz, at a center frequency of 850MHz.

```
test_apps/rx_samples_minimal -c 0 -d /tmp/iq_output -f 850000000 -r 61440000 -b 50000000 --handle=A1
```

- where handle can be any of the following: `A1,A2,B1`

Note: libsidekiq will use the provided sample rate and bandwidth arguments to select the RFIC profile best meeting the constraints. Please contact Epiq Solutions if additional rates are desired.

The same can be achieved using the following sequence of API functions:

```
skiq_init(card)
skiq_write_rx_sample_rate_and_bandwidth(card, handle, rate, bandwidth)
skiq_write_rx_LO_freq(card, handle, frequency)
skiq_start_rx_streaming_multi_immediate()
while(receiving)
   skiq_receive(card, handle, rx_block, len)
skiq_stop_rx_streaming_multi_immediate()
skiq_exit()
```

5.8.2 Receive: two phase coherent channels, up to 50MHz IBW

The following example captures phase coherent samples for two channels at 61.44Msps with a bandwidth of 50MHz, at a center frequency of 850MHz.

```
test_apps/rx_samples_minimal -c 0 -d /tmp/iq_output -f 850000000,850000000 -r 61440000,61440000 -b
50000000,50000000 --handle=A1,A2
```

The same can be achieved using the following sequence of API functions:

```
skiq_init(card)
skiq_write_rx_sample_rate_and_bandwidth_multi(card, handles, nr_handles, rates, bandwidths)
skiq_write_chan_mode(chan_mode_dual)
skiq_write_rx_LO_freq(card, handle #1 or handle #2)
skiq_start_rx_streaming_multi_immediate()
while(receiving)
   skiq_receive(card, handle, rx_block, len)
skiq_stop_rx_streaming_multi_immediate()
skiq_exit()
```

Note: `skiq_write_rx_sample_rate_and_bandwidth_multi()` is the preferred API function for configuring multiple handles. Please refer to *Configuring Sample Rate / Channel Bandwidth* (page 55) for more information.

5.8.3 Receive: two independently tunable channels, same sample rate, up to 50MHz IBW

The following example captures samples for two channels at 61.44Msps with a bandwidth of 50MHz, tuned to 750MHz and 850MHz respectively.

```
test_apps/rx_samples_minimal -c 0 -d /tmp/iq_output -f 750000000,850000000 -r 61440000,61440000 -b
50000000,50000000 --handle=A1,B1
```

The same can be achieved using the following sequence of API functions:

```
skiq_init(card)
skiq_write_rx_sample_rate_and_bandwidth_multi(card, handles, nr_handles, rates, bandwidths)
skiq_write_rx_LO_freq(card, handle #1)
skiq_write_rx_LO_freq(card, handle #2)
skiq_start_rx_streaming_multi_immediate()
 while(receiving)
   skiq_receive(card, handle, rx_block, len)
skiq_stop_rx_streaming_multi_immediate()
skiq_exit()
```

5.9 Sidekiq API

The Doxygen output is included in the SDK bundle as HTML and PDF. Please refer to `sidekiq_sdk_current/doc/html` or `sidekiq_sdk_current/doc/Sidekiq_API_4.17.0.pdf` for information regarding the API.

5.10 FPGA user_app examples

5.10.1 Transmitting samples from FPGA memory

Starting with FPGA PDK reference v3.12.0, the FPGA offers a way to allow signal playback (transmit) from its internal memories (BRAM) independent of transport. There is an accompanying test application (`tx_samples_from_FPGA_RAM`) included in the Sidekiq SDK bundle to showcase this FPGA user_app interface. It uses FPGA user registers and libsidekiq public API functions to load samples from file(s) to each desired transmit handle's associated FPGA memory and begin playing back those circular sample buffers. On the Sidekiq X4, this test application and the user_app interface may be used to demonstrate transmitting 4 channels in a phase coherent manner.

The software test application relies on support from the FPGA user_app to have those Tx Internal Memory Interfaces at the appropriate addresses and sizes. If a user changes the user_app implementation, it is up to them to make changes in the software test application accordingly.

The `tx_samples_from_FPGA_RAM` application takes most of the same arguments that the `tx_samples` application does. The `--source` argument copies the sample contents from the file into ALL specified transmit handles, effectively transmitting the sample waveform. The `--prefix` argument is a convenience argument to specify a file prefix used to copy sample contents from different files into their corresponding transmit handles' internal memory interface in the FPGA user_app.

Section 7.3.7 of the Sidekiq X2 / X4 FPGA Developers Manual (v3.12.0) discusses this transmit memory interface from the FPGA's point of view. At v3.12.0 of the PDK, this user_app addition is only available with stock Sidekiq X2 / X4 FPGA bitstreams and may be disabled during synthesis if desired by an FPGA developer. It is up to the user to ensure that both sides of this interface are working should any changes be made. This addition may not be enabled by default in future PDKs.

6 Hosts & Platforms

6.1 Windows Sidekiq Development

As of libsidekiq **v4.4.0**, the Sidekiq mPCIe and Sidekiq M.2 hardware is supported under Windows 7 and includes a signed device driver. The remainder of this section shows the steps necessary to install the SDK and associated device drivers under Windows 7. The Windows installer is available at the same location as the Linux release bundles on the Epiq Support forum.

Note: Libsidekiq no longer supports Windows 7 as of libsidekiq **v4.11.0** as Microsoft ended Windows 7 support on January 14, 2020.

As of libsidekiq **v4.6.0**, the existing Sidekiq mPCIe and Sidekiq M.2 hardware is supported under Windows 10 as well as Windows 7. In addition, the Sidekiq X2 hardware is also supported. Installation of the SDK under Windows 10 is very similar to that of the Windows 7 installation.

6.1.1 Install the SDK

The Windows installer is a self-extracting and executing file. When run, it will present a series of dialog boxes similar to those which follow.

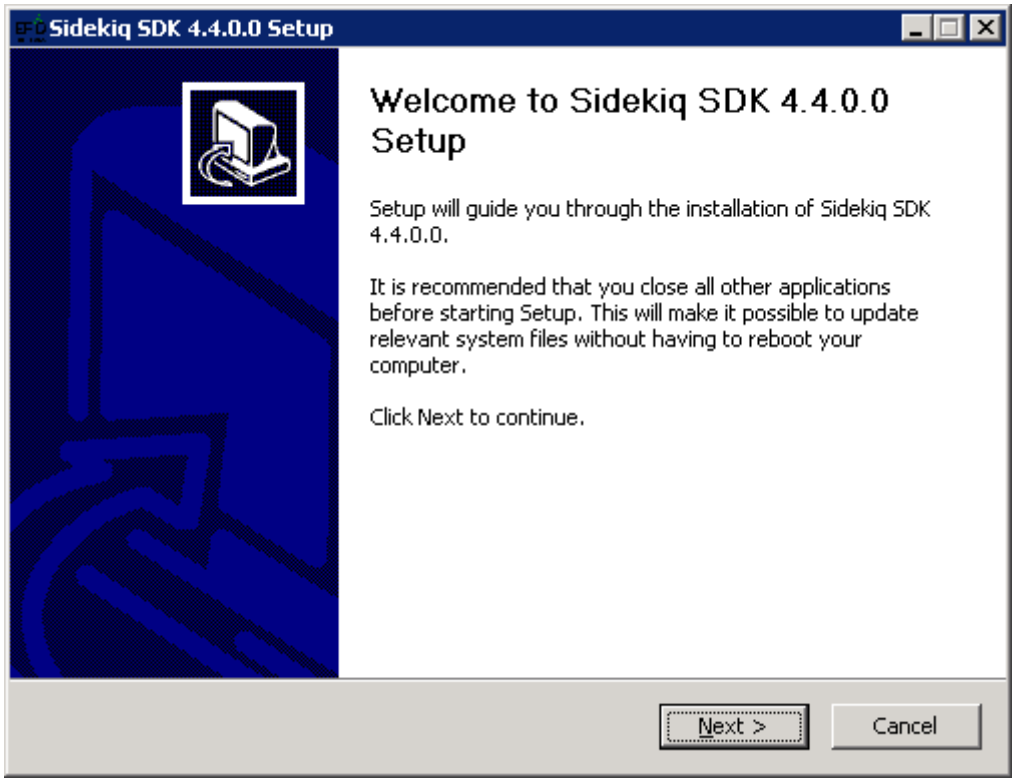

Fig. 6.1: Windows Installer Dialog

Choose *Next*.

Choose the components to install. It is recommended to install all of the components. The VC++ Redistributable libraries may already be installed if the target machine also has an installation of Microsoft Visual Studio present.

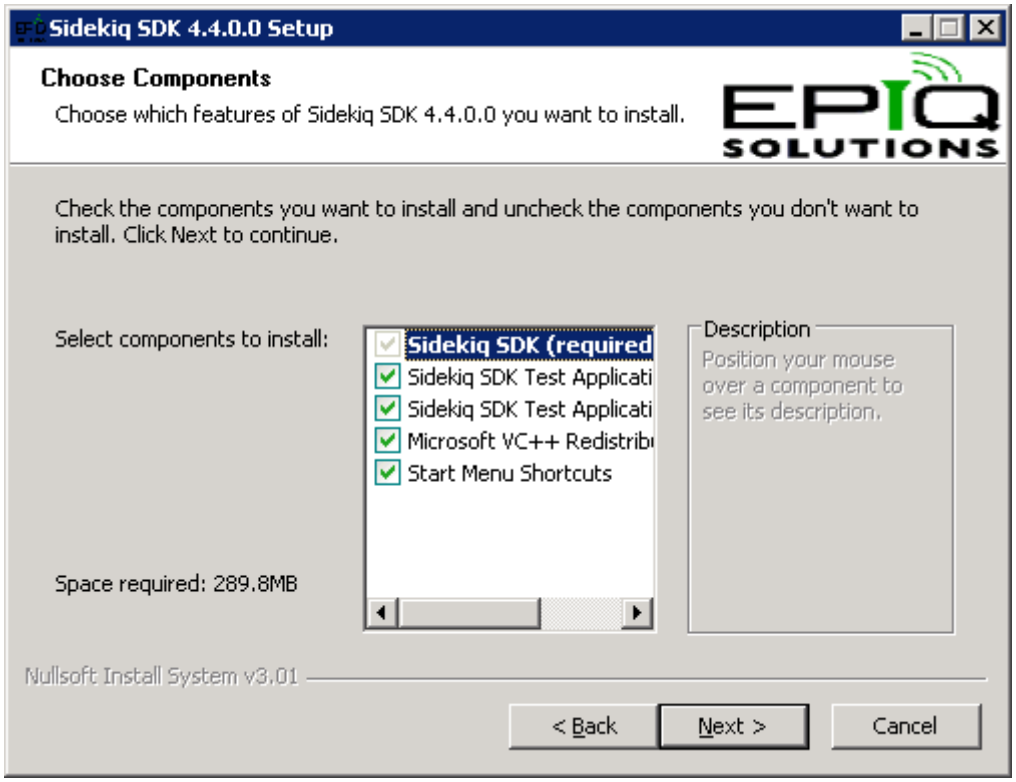

Fig. 6.2: Windows Installer Components Dialog

Choose the installation directory and wait for the installation to complete.

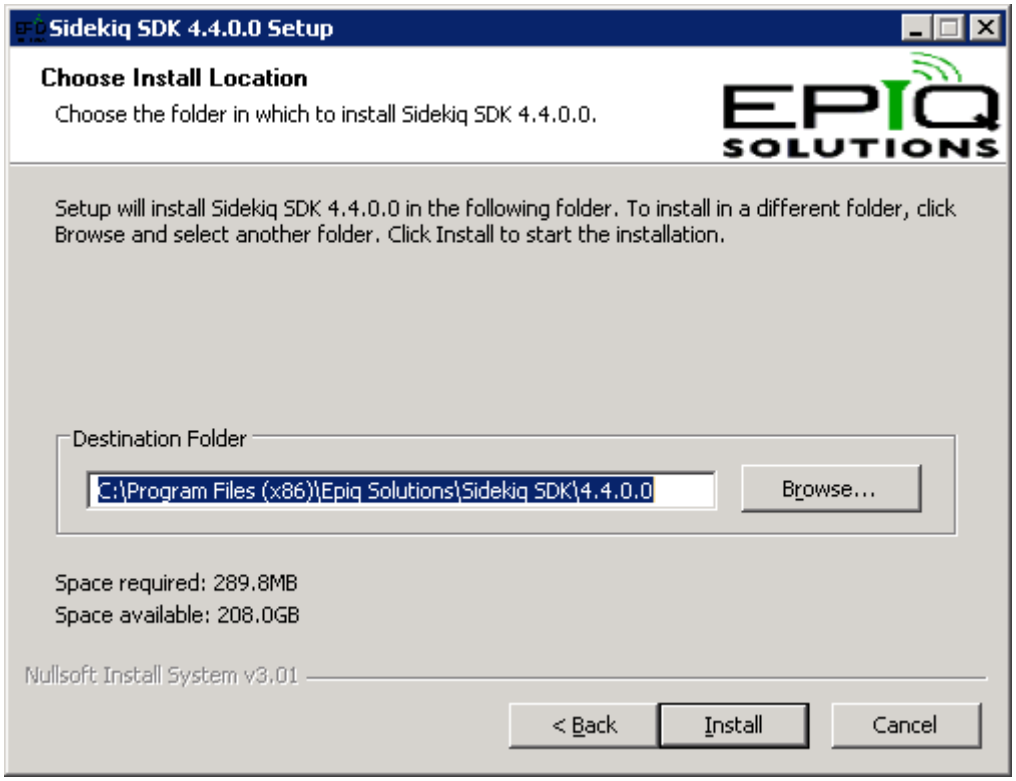

Fig. 6.3: Windows Installer Path Dialog

If the VC++ Redistributables component was selected for installation, the installer will launch another installation of the VC++ libraries.

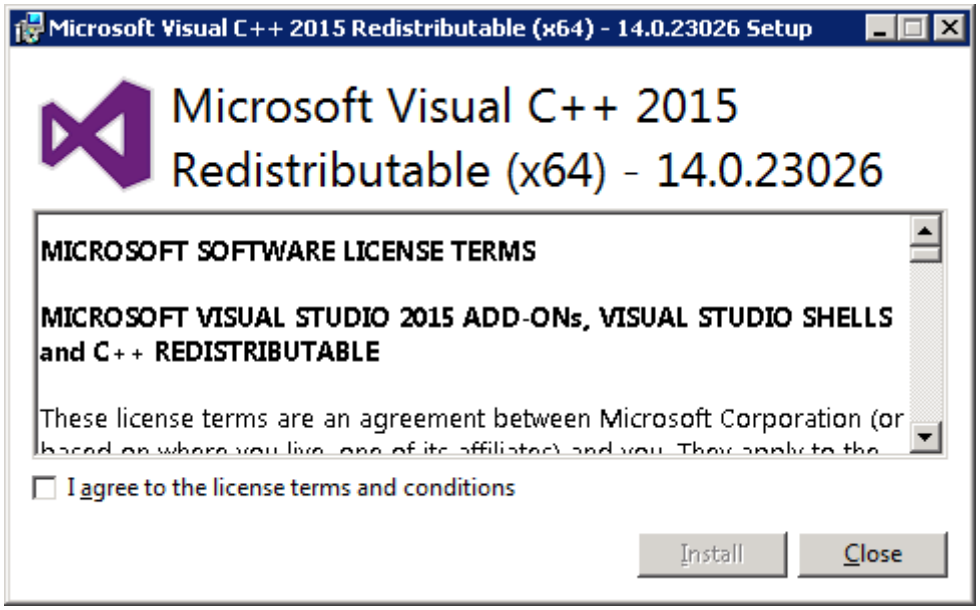

Fig. 6.4: Windows MSVC++ Redistributable Dialog

6.1.2 Sidekiq Device Configuration

When the Sidekiq is inserted into the computer, the Device Manager will detect the hardware but no driver will be loaded. The M.2 or PCIe interface for the Sidekiq will be reported as a "*PCI Data Acquisition and Signal Processing Controller*" and the USB interface will be reported as a "*Sidekiq*", as shown in *Windows Detection of Sidekiq* (page 74).

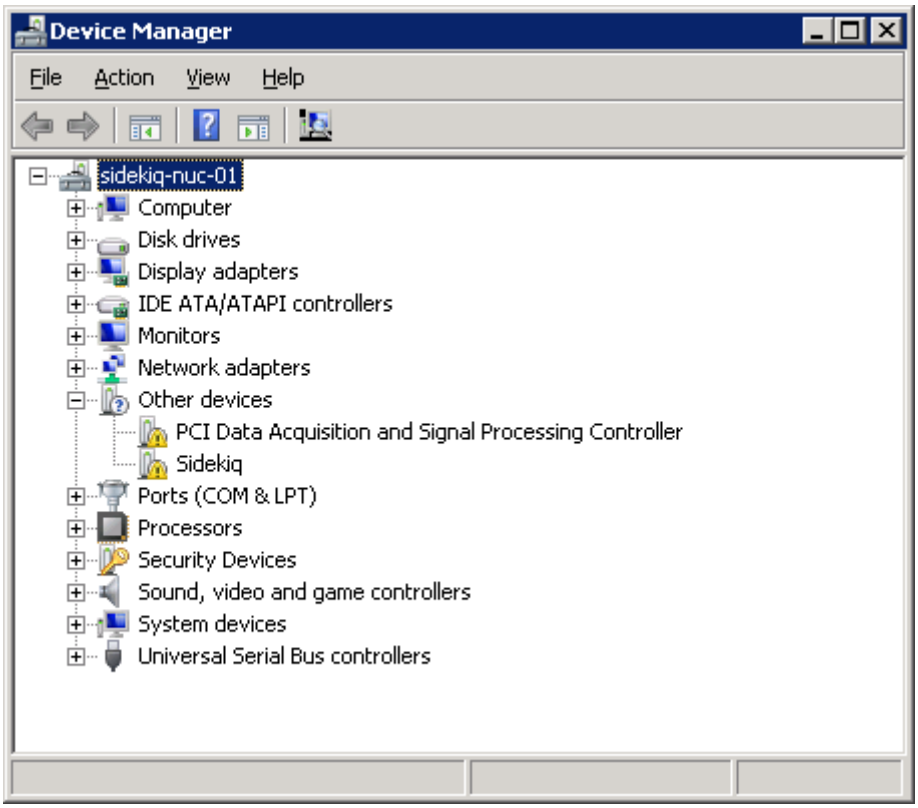

Fig. 6.5: Windows Detection of Sidekiq

Windows® will prompt to select a driver, or by right-clicking on the "*PCI Data Acquisition and Signal Processing Controller*" and the "*Sidekiq*" one may choose to install or "*Update Driver Software...*" for each interface.

Select the Sidekiq PCIe Driver Software

When prompted to search for the device driver software, choose "*Browse my computer for driver software*", as shown in *Windows Driver Search* (page 75).

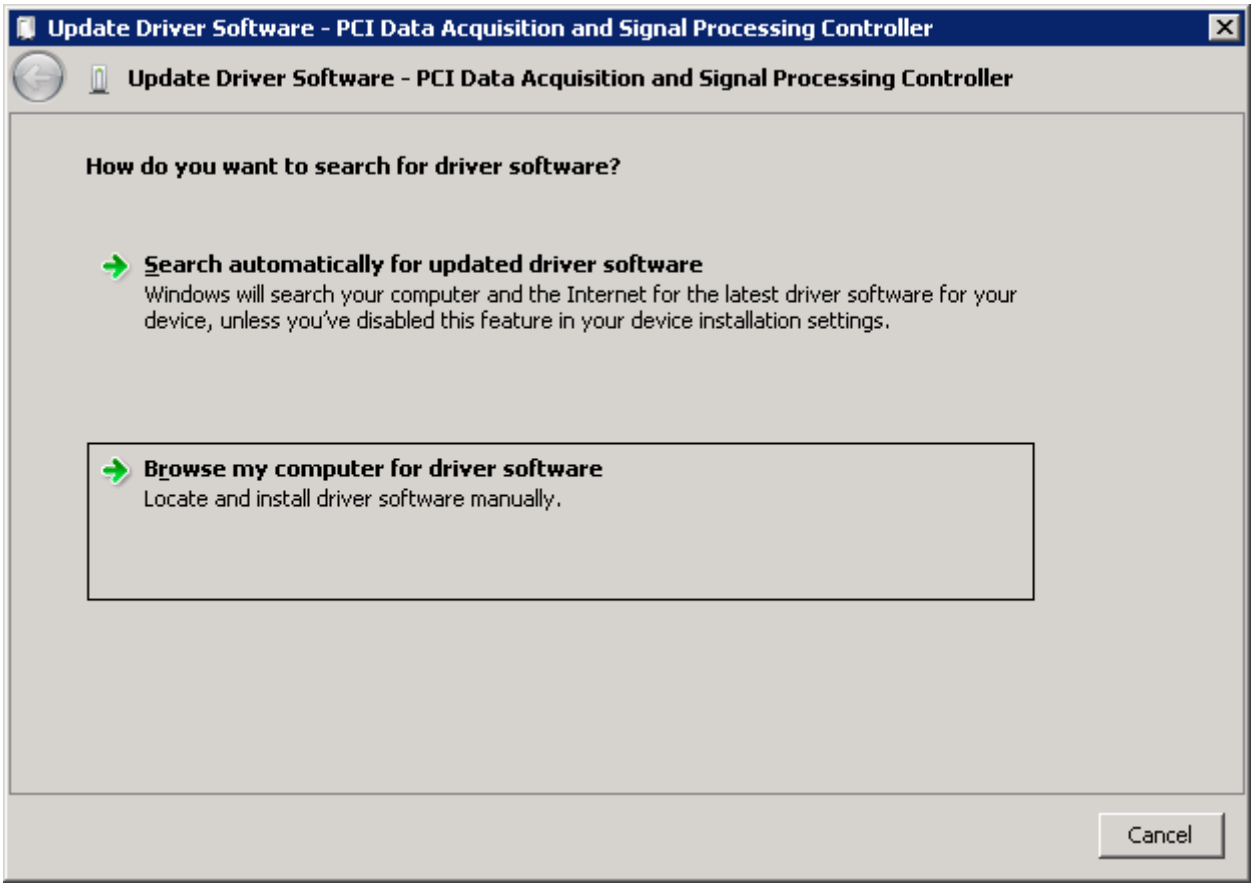

Fig. 6.6: Windows Driver Search

Next, choose the path to the driver package, containing the `DMADriver.sys` file, provided by Epiq Solutions. For example, refer to *Windows Driver Package Selection* (page 76).

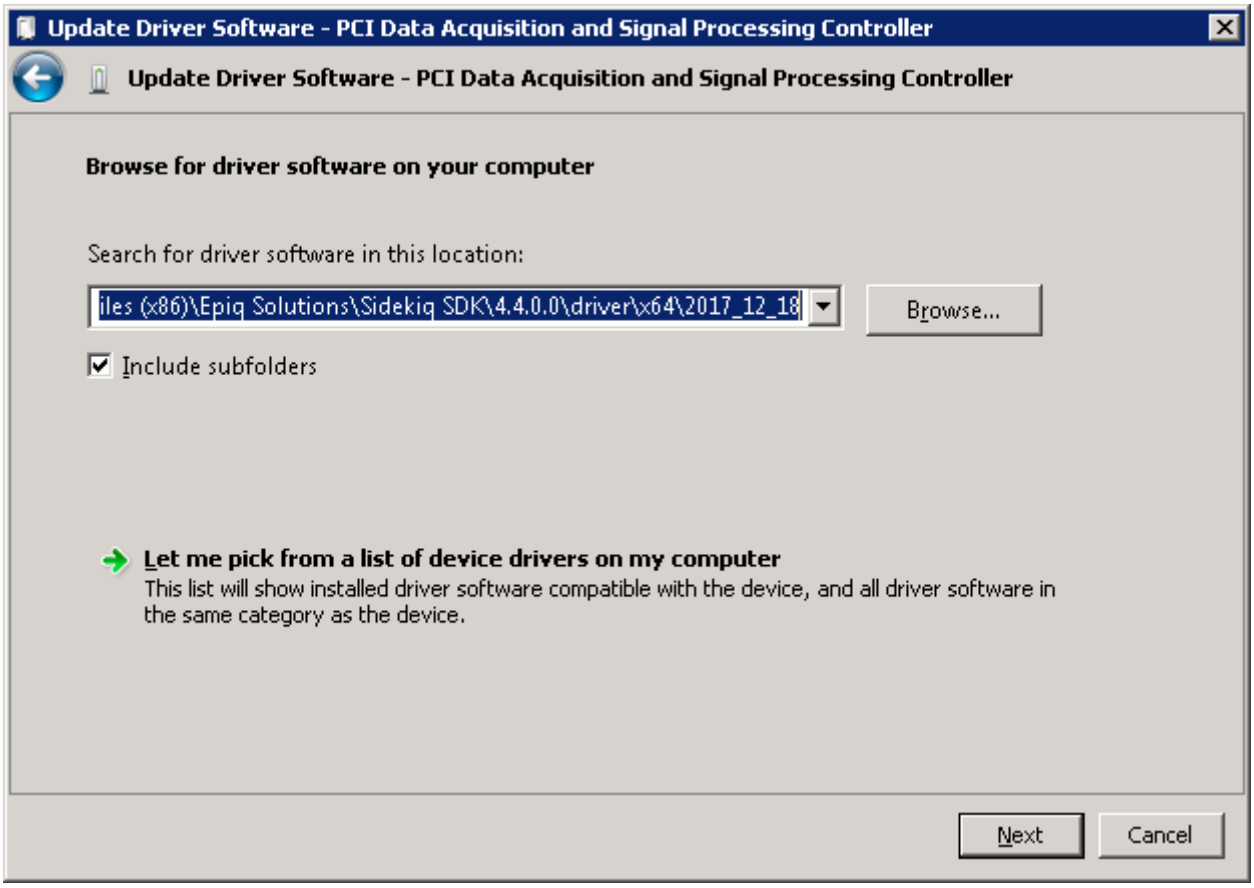

Fig. 6.7: Windows Driver Package Selection

Agree to install the driver. The Device Manager will then rename the "*PCI Data Acquisition and Signal Processing Controller*" as a "*PCI Express DMA Device*", as shown in *Windows Successful DMA Driver Installation* (page 77).

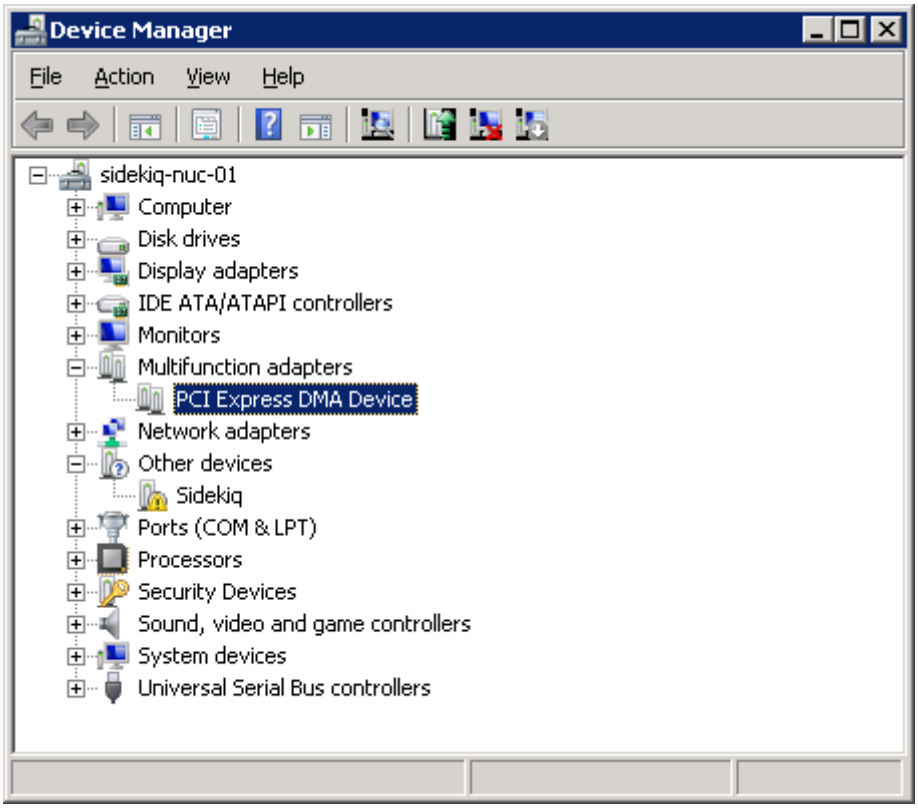

Fig. 6.8: Windows Successful DMA Driver Installation

Select the USB Driver Software

Similarly, the "Sidekiq" device driver for the USB interface may be installed by selecting the path to its Setup Information file (`*.inf`), which is installed in the `usb_driver` sub-directory of the Sidekiq SDK installation directory. Refer to to see how both Sidekiq devices should appear in Device Manager.

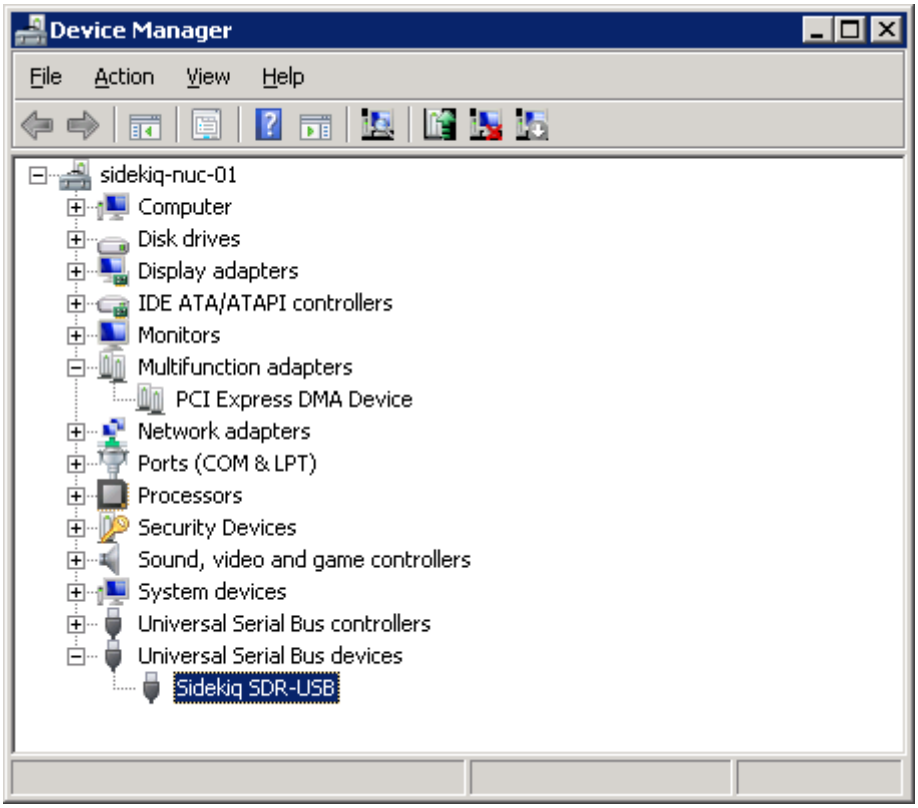

Fig. 6.9: Windows Successful USB Driver Installation

It should now be possible to execute applications built against the SDR library (`libsidekiq__mingw64.a`) to communicate with the device hardware.

6.1.3 Windows Development Tools

The MinGW-64 or Microsoft Visual Studio compilers may be used with libsidekiq to write applications to control the Sidekiq radio. These toolchains are available from the following, respective, links:

http://www.msys2.org/

https://www.visualstudio.com/

MinGW-64

The MinGW-64 compiler is installed from the MSYS2 environment using the `pacman` package manager. The test applications were built using MinGW gcc version 7.2.0. Since MinGW-64 is designed to support the GCC compiler on Windows systems, the user can also leverage libsidekiq's `arg_parser` library to allow easier working with command line arguments in applications.

Visual Studio

The Visual Studio test applications have been built using Visual Studio 2015 Community Edition. Applications built with Visual Studio only need to link against `libsidekiq__mingw64.a` and cannot use, nor need to use, `arg_parser.a`

Note: The Visual Studio solution files may require the paths to the header files and library to be updated.

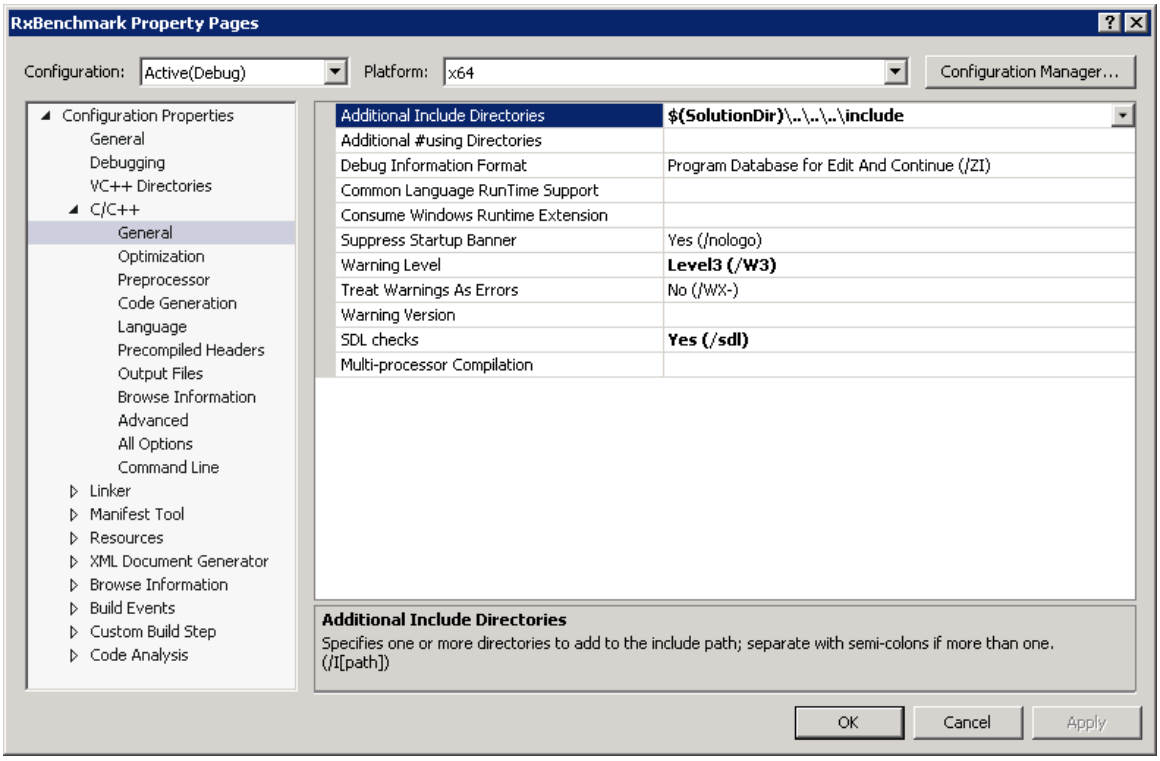

Fig. 6.10: Visual Studio - Additional Include Directories

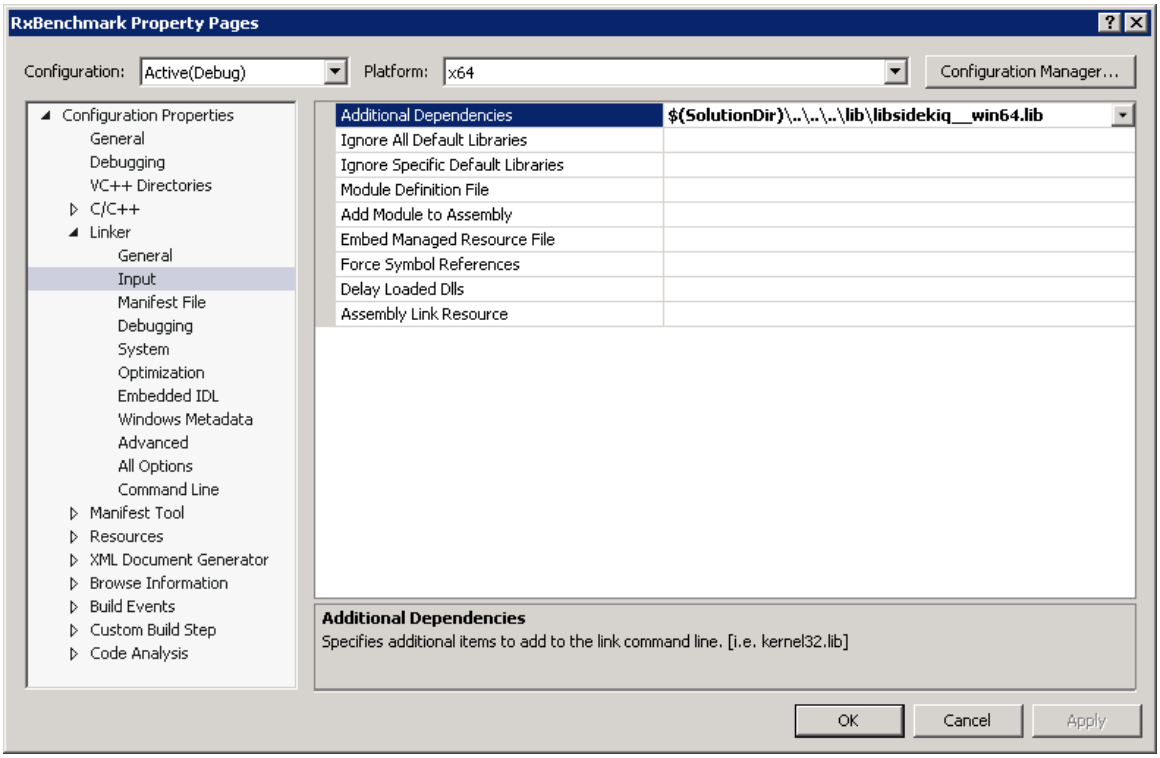

Fig. 6.11: Visual Studio - Additional Dependencies

6.2 Developing for Alternative Host Platforms

Sidekiq can be used on a wide variety of host platforms. The following sections provide a brief overview of how to use libsidekiq on the various platforms. For detailed information on a specific host platform, please contact Epiq Solutions' Support, *[5]* (page 8).

6.2.1 Supported Architectures

The Sidekiq SDK currently supports host platforms with the following CPU architectures: 64-bit x86, 32-bit x86, ARM Cortex A9 (specifically validated on Freescale's i.MX6), and 64-bit ARM (specifically validated on NVIDIA's Jetson TX1/TX2/Xavier). Prebuilt test applications for each of the architectures are located within their own subdirectory under the `prebuilt_apps` directory of the SDK and also available for download, *[5]* (page 8). For example, the binaries for a 64-bit x86 processor is located at: `sidekiq_sdk_current/prebuilt_apps/x86_64.gcc`.

6.2.2 Building Test Applications

The Sidekiq SDK provides versions of libsidekiq for each of the supported architectures. Additionally, the test applications can be compiled for any of the these architectures. The steps for building a specific host platform are as follows:

1. Change to the directory where the SDK was previously uncompressed.

```
$ cd /home/sidekiq/sidekiq_sdk_v4.17.x/
```

2. Change directories to the `test_apps` directory and run make, using `BUILD_CONFIG` to specify the host target.

As of Sidekiq SDK release v3.0.0, a libc version of 2.11 or greater is required on x86 64-bit and 32-bit platforms. Using a previous version of libc may result in linking errors during build. Verification of the version of libc on the build system can be performed by locating libc.so.6, executing it, and taking note of the version. As a point of reference, Ubuntu 12.04 and later, as well as CentOS 6.7 and later currently satisfy this requirement.

Deploying Applications

As of Sidekiq SDK release v4.0.0, both glib-2.0 and libusb open source libraries are leveraged within libsidekiq. As of Sidekiq SDK release v4.17.5, the libtirpc open source library is also leveraged within libsidekiq. Correspondingly, the glib-2.0 (LGPL) and libusb (LGPL) license agreements are maintained by linking to them dynamically and the libtirpc (BSD) license agreement is maintained by acknowledging it in this documentation. There are two approaches to deploying applications built against libsidekiq, glib-2.0, libusb, and libtirpc: one approach is used when the application is executed on the build machine (i.e. the SDK is installed), while the other approach is used when the application is executed on a target machine (i.e. the SDK is not installed).

In the first approach, the SDK contains all of the requisite dynamic libraries in the directory `$SDK/lib/support/$BUILD_CONFIG/usr/lib/epiq`. The `$SDK/test_apps/Makefile` has all of the necessary flags for building and executing on the build machine. After final linking, applications reference the glib-2.0, libusb, and libtirpc libraries in the SDK's preceding support directory.

In the second approach, the necessary glib-2.0, libusb, and libtirpc libraries are required to be installed on the target machine at a known location: `/usr/lib/epiq`. The SDK provides these libraries as packages for all supported platforms in the form of DEB and RPM files. Use *Linker Flags for glib-2.0, libusb, and libtirpc when deploying applications on target machines* (page 81) to determine which package should be installed on the target machine.

Note:

A run-time package is not required for the Matchstiq S10 as long as the unit is updated to support Sidekiq SDK release v3.0.0, v4.0.0 for libusb, or v4.17.5 for libtirpc as the libraries

are included by default.

With the introduction of glib-2.0, libusb, and libtirpc as support libraries, there are additional linker flags required when building and linking with libsidekiq. All of the required flags are captured in the `test_apps/Makefile`, but are repeated here for clarity. *These flags are used whenever deploying applications to a target machine.* Depending on the `BUILD_CONFIG`, use these compiler flags (assuming SDK references its current location):

Table 6.1: Linker Flags for glib-2.0, libusb, and libtirpc when deploying applications on target machines

BUILD_CONFIG	Linker Flags

x86_64.gcc	`-L $(SDK)/lib/support/$(BUILD_CONFIG)/usr/lib/epiq \`
	`-lglib-2.0 -lusb-1.0 -ltirpc -lpthread -lrt -lm \`
	`-Wl,--enable-new-dtags \`
	`-Wl,-rpath,/usr/lib/epiq`

x86_32.gcc	`-L $(SDK)/lib/support/$(BUILD_CONFIG)/usr/lib/epiq \`
	`-lglib-2.0 -lusb-1.0 -ltirpc -lpthread -lrt -lm \`
	`-Wl,--enable-new-dtags \`
	`-Wl,-rpath,/usr/lib/epiq`

Continued on next page

Table 6.1 – continued from previous page

BUILD_CONFIG	Linker Flags

aarch64.gcc6.3	`-L $(SDK)/lib/support/$(BUILD_CONFIG)/usr/lib/epiq \`
	` -lglib-2.0 -lusb-1.0 -ltirpc -lpthread -lrt -lm \`
	` -Wl,--enable-new-dtags \`
	` -Wl,-rpath,/usr/lib/epiq`

arm_cortex-a9.gcc4.8_gnueabihf_linux	`-L $(SDK)/lib/support/$(BUILD_CONFIG)/usr/lib/epiq \`
	` -lglib-2.0 -lusb-1.0 -ltirpc -lpthread -lrt -lm \`
	` -Wl,--enable-new-dtags \`
	` -Wl,-rpath,/usr/lib/epiq`

arm_cortex-a9.gcc4.8_uclibc_openwrt	`-L $(SDK)/lib/support/$(BUILD_CONFIG)/usr/lib/epiq \`
	` -lglib-2.0 -lintl -liconv -lusb-1.0 -ltirpc \`
	` -lpthread -lrt -lm \`
	` -Wl,--enable-new-dtags \`
	` -Wl,-rpath,/usr/lib/epiq`

arm_cortex-a9.gcc4.9.2_gnueabi	`-L $(SDK)/lib/support/$(BUILD_CONFIG)/usr/lib/epiq \`
	` -liio -lz -lxml2 -ltirpc -lpthread -lrt -lm \`
	` -Wl,--enable-new-dtags \`
	` -Wl,-rpath,/usr/lib/epiq`

arm_cortex-a9.gcc7.2.1_gnueabihf *(supported starting in v4.9.5)*	`-L $(SDK)/lib/support/$(BUILD_CONFIG)/usr/lib/epiq \`
	` -liio -ltirpc -lpthread -lrt -lm \`
	` -Wl,--enable-new-dtags \`
	` -Wl,-rpath,/usr/lib/epiq`

arm_cortex-a9.gcc5.2_glibc_openwrt	`-L $(SDK)/lib/support/$(BUILD_CONFIG)/usr/lib/epiq \`
	` -lglib-2.0 -lintl -liconv -lusb-1.0 -ltirpc \`
	` -lpthread -lrt -lm \`
	` -Wl,--enable-new-dtags \`
	` -Wl,-rpath,/usr/lib/epiq`

Table 6.2: Run-time Library Packages

Distro Packaging Type	BUILD_CONFIG	Run-time Library Package
Debian	x86_64.gcc	sidekiq-shared-libs-x86-64.gcc_4.17.x_amd64.deb
	x86_32.gcc	sidekiq-shared-libs-x86-32.gcc_4.17.x_amd64.deb
Redhat (RPM)	x86_64.gcc	sidekiq-shared-libs-x86_64.gcc-4.17.x-1.x86_64.rpm
	x86_32.gcc	sidekiq-shared-libs-x86_32.gcc-4.17.x-1.x86_64.rpm

Note: A developer may replace glib-2.0, libusb, or libtirpc at their discretion with the understanding that libsidekiq is built and tested with glib version 2.42.1, libusb-1.0.20, and libtirpc-1.3.3.

6.2.3 Additional Dependencies

In order for a Sidekiq card to be functional on a specific host platform, there are additional drivers (`dmadriver.ko`, `skiq_platform_device.ko`, and `pci_manager.ko`) that libsidekiq depends on. The driver modules are pre-installed on the Sidekiq PDK laptop and any updates are included with the system update. If an alternative host platform is being targeted, please contact Epiq Solutions Support, *[5]* (page 8), for driver availability and support for the alternative host platform.

Note: In order to use the GPS / UART functionality of a Sidekiq Stretch, the user must also make certain that `sidekiq_uart.ko` and `sidekiq_gps.ko` are available for the target platform.

Sidekiq Stretch UART

The Sidekiq Stretch's on-board GPS can provide NMEA-0183 messages through a UART device on the host system. When both the `sidekiq_uart.ko` kernel module and the `dmadriver.ko` (v5.3.0 or later) kernel module are loaded, a UART character device file is available as `/dev/ttySKIQ_UART<card>` where `<card>` is the Stretch's card index. This device file may be used directly in any application (whether it uses libsidekiq or not) to receive NMEA-0183 messages. This device file may also be used in conjunction with `gpsd` and `gpsmon`.

Sidekiq Stretch GPS sysfs

Control and status monitoring of Sidekiq Stretch's on-board GPS is provided through several sysfs entries on the host system. When both the `sidekiq_uart.ko` kernel module and the `dmadriver.ko` (v5.3.0 or later) kernel module are loaded, several sysfs entries are available in `/sys/fs/skiq_gps/<card>` where `<card>` is the Stretch's card index. These entries are accessible from any application, whether it uses libsidekiq or not. A short summary of the available entries as of v0.0.2 of `sidekiq_gps` are as follows:

- `ant_bias_en` – enables (1) or disables (0) the antenna bias (see the Sidekiq Stretch Hardware User's Manual for more details)
- `has_fix` – GPS has a fix (1) or does not have a fix (0)
- `power_en_n` – enables (0) or disables (1) power to the GPS module
- `reset` – controls the RESET line to the GPS module - (1) holds the device in reset, while (0) allows it to run

6.2.4 Setting up Sidekiq on New Host PC

The following is a brief overview of how to configure a new host PC with Sidekiq. For details on configuring a Windows Host PC, refer to *Windows Sidekiq Development* (page 70).

Linux Host PC Requirements

The following are requirements for the Host PC.

- Must be running Linux.
- Must use a 64-bit processor.
- Must be running supported kernel version (see below tables)

– In CentOS 7.6, the 3.10.0-957 kernel series (base through 12.2 releases) have incorrectly marked a required kernel symbol related to DMA transactions as GPL. At this time, we are unable to provide kernel modules for this kernel series as our DMA driver has a proprietary license. *UPDATE*: Starting with release 21.2, the 3.10.0-957 kernel series has fixed this issue. The EL Repo has fixed this issue starting with release 5.1 in the 3.10.0-957 kernel series.

Table 6.3: Supported Ubuntu Kernels

Ubuntu (x86 64-bit)	
Low Latency Kernels • 4.4.0-116-lowlatency • 5.3.0-45-lowlatency	**Generic Kernels** • 4.4.0-21-generic through 4.4.0-210-generic • 4.8.0-34-generic through 4.8.0-58-generic • 4.10.0-14-generic through 4.10.0-42-generic • 4.11.0-13-generic and 4.11.0-14-generic • 4.12.0-13-generic • 4.13.0-16-generic through 4.13.0-45-generic • 4.15.0-13-generic through 4.15.0-208-generic • 4.18.0-13-generic through 4.18.0-25-generic • 5.0.0-15-generic through 5.0.0-63-generic • 5.3.0-19-generic through 5.3.0-76-generic • 5.4.0-26-generic through 5.4.0-146-generic • 5.8.0-23-generic through 5.8.0-63-generic • 5.11.0-16-generic through 5.11.0-18-generic • 5.11.0-22-generic through 5.11.0-49-generic • 5.13.0-21-generic through 5.13.0-52-generic • 5.15.0-25-generic through 5.15.0-69-generic • 5.19.0-28-generic through 5.19.0-41-generic • 4.13.0-1031-oem • 4.15.0-1043-oem • 4.19.0-041900rc3-generic

Table 6.4: Supported CentOS 6.x Kernels

CentOS 6.x
• 2.6.32-279.el6.x86_64 series up to 22.1 • 2.6.32-358.el6.x86_64 series up to 23.2 • 2.6.32-431.el6.x86_64 series up to 29.2 • 2.6.32-504.el6.x86_64 series up to 30.3 • 2.6.32-573.el6.x86_64 series up to 26.1 • 2.6.32-642.el6.x86_64 series up to 15.1 • 2.6.32-696.el6.x86_64 series up to 30.1 • 2.6.32-754.el6.x86_64 series up to 33.1 • 3.10.83-1.el6.elrepo.x86_64 • 3.10.92-1.el6.elrepo.x86_64

Table 6.5: Supported CentOS 7.x Kernels

CentOS 7.x
• 3.10.0-123.el7.x86_64 series up to 20.1 • 3.10.0-229.el7.x86_64 series up to 20.1 • 3.10.0-327.el7.x86_64 series up to 36.3 • 3.10.0-514.el7.x86_64 series up to 26.2 • 3.10.0-693.el7.x86_64 series up to 21.1 • 3.10.0-862.el7.x86_64 series up to 14.4 • 3.10.0-957.el7.x86_64 series from **21.2 to 27.2** • 3.10.0-957.el7.centos.plus.x86_64 from **5.1 to 27.2** • 3.10.0-1062.el7.x86_64 series up to 18.1 • 3.10.0-1062.1.1.el7.centos.plus.x86_64 series up to 18.1 • 3.10.0-1127.el7.x86_64 series up to 19.1 • 3.10.0-1127.el7.centos.plus.x86_64 series up to 19.1 • 3.10.0-1160.el7.x86_64 series up to 42.2 • 3.10.0-1160.el7.centos.plus.x86_64 series up to 42.2 • 4.4.39-1.el7.elrepo.x86_64 • 4.4.139-1.el7.elrepo.x86_64 • 4.4.198-1.el7.elrepo.x86_64 • 4.16.9-1.el7.elrepo.x86_64 • 5.1.15-1.el7.elrepo.x86_64

Table 6.6: Supported CentOS 8.x Kernels

CentOS 8.x
• 4.18.0-80.el8.x86_64 • 4.18.0-80.1.2.el8_0.x86_64 series up to 11.2 • 4.18.0-147.el8.x86_64 • 4.18.0-147.0.3.el8_1.x86_64 series up to 8.1 • 4.18.0-147.3.1.el8_1.centos.plus.x86_64 • 4.18.0-147.5.1.el8_1.centos.plus.x86_64 • 4.18.0-193.el8.x86_64 • 4.18.0-193.1.2.el8_2.x86_64 series up to 19.1 • 4.18.0-193.6.3.el8_2.centos.plus.x86_64 series up to 19.1 • 4.18.0-240.el8.x86_64 • 4.18.0-240.1.1.el8_3.x86_64 series up to 22.1 • 4.18.0-240.1.1.el8_3.centos.plus.x86_64 series up to 22.1 • 4.18.0-305.3.1.el8.x86_64 series up to 19.1 • 4.18.0-305.3.1.el8.centos.plus.x86_64 series up to 19.1 • 4.18.0-305.25.1.el8_4.centos.plus.x86_64 • 4.18.0-348.el8.x86_64 • 4.18.0-348.2.1.el8_5.x86_64

Table 6.7: Supported Fedora and Debian Kernels

Fedora	Debian
• 3.17.4-301.fc21.x86_64 • 4.13.12-200.fc26.x86_64 • 4.13.9-200.fc26.x86_64 • 4.13.9-300.fc27.x86_64 • 4.14.3-300.fc27.x86_64 • 4.14.5-300.fc27.x86_64 • 5.0.9-301.fc30.x86_64	• 3.16.0-4-amd64

Table 6.8: Supported NVIDIA Jetson TX1 / TX2 Kernels

NVIDIA Jetson TX1	NVIDIA Jetson TX2
• JetPack 3.1 (L4T R28.1) • JetPack 3.2.1 (L4T R28.2) • JetPack 3.3 (L4T R28.2) • JetPack 4.4.1 (L4T R32.4.4) • JetPack 4.5 (L4T R32.5) • JetPack 4.5.1 (L4T R32.5.1) • JetPack 4.6 (L4T R32.6.1) • JetPack 4.6.1 (L4T R32.7.1) • JetPack 4.6.2 (L4T R32.7.2) • JetPack 4.6.3 (L4T R32.7.3)	• JetPack 3.1 (L4T R28.1) • JetPack 3.2.1 (L4T R28.2.1) • JetPack 3.3 (L4T R28.2.1) • JetPack 4.2 (L4T R32.1) • JetPack 4.2.1 (L4T R32.2) • JetPack 4.2.2 (L4T R32.2.1) • JetPack 4.2.3 (L4T R32.2.1) • JetPack 4.3 (L4T R32.3.1) • JetPack 4.4 (L4T R32.4.3) • JetPack 4.4.1 (L4T R32.4.4) • JetPack 4.5 (L4T R32.5) • JetPack 4.5.1 (L4T R32.5.1) • JetPack 4.6 (L4T R32.6.1) • JetPack 4.6.1 (L4T R32.7.1) • JetPack 4.6.2 (L4T R32.7.2) • JetPack 4.6.3 (L4T R32.7.3)

Table 6.9: Supported NVIDIA Jetson Xavier/Orin Kernels

NVIDIA Jetson Xavier/Orin
• JetPack 4.1.1 (L4T R31.1) • JetPack 4.2 (L4T R32.1) • JetPack 4.2.1 (L4T R32.2) • JetPack 4.2.2 (L4T R32.2.1) • JetPack 4.2.3 (L4T R32.2.1) • JetPack 4.3 (L4T R32.3.1) • JetPack 4.4 (L4T R32.4.3) • JetPack 4.4.1 (L4T R32.4.4) • JetPack 4.5 (L4T R32.5) • JetPack 4.5.1 (L4T R32.5.1) • JetPack 4.6 (L4T R32.6.1) • JetPack 4.6.1 (L4T R32.7.1) • JetPack 4.6.2 (L4T R32.7.2) • JetPack 4.6.3 (L4T R32.7.3) • JetPack 5.0.0 (L4T R34.1.0) • JetPack 5.0.1 (L4T R34.1.1) • JetPack 5.0.2 (L4T R35.1.0) • JetPack 5.1.0 (L4T R35.2.1) • JetPack 5.1.1 (L4T R35.3.1)

Configuring A New Host

1. Install Sidekiq card in new host PC
2. Download and install the latest Sidekiq update from Epiq Solutions support forum, *[5]* (page 8). Refer to *Installation Procedure* (page 22) for details on what the update procedure installs.
3. Download and install ERA for Epiq Solutions support forum, *[5]* (page 8).

6.2.5 Developing for the Matchstiq S1x and S2x

The Matchstiq S1x and S2x platforms (see https://epiqsolutions.com/rf-transceiver/matchstiq-s/ for more details) are small form factor software defined radio platforms that have internal Sidekiq card(s). This means libsidekiq is used to develop custom applications for the Matchstiq S1x platform.

The Sidekiq SDK already supports the processor architecture of the Matchstiq S1x and S2x platforms, however a cross-compiler is needed to support building custom applications. The toolchain may be downloaded from the Epiq Solutions Support forum using the following link:

https://support.epiqsolutions.com/viewtopic.php?p=6395

The toolchain is offered in either a Debian or RPM format and is installed at a specific location on the development machine and symlinked as expected by the Sidekiq SDK. The toolchain was created by OpenWRT's ImageBuilder framework.

To verify correct toolchain installation, navigate to the Sidekiq SDK directory and compile the test applications specifying a `BUILD_CONFIG` of `arm_cortex-a9.gcc5.2_glibc_openwrt`. The applications may then be copied to the Matchstiq S1x or S2x unit and executed (as shown in the next code block). Before executing applications on the Matchstiq, it is imperative no other application is using the Sidekiq card.

First, copy the executable to the Matchstiq using the *scp* command.

```
linux$ cd sidekiq_sdk_v4.17.x/test_apps

linux$ make BUILD_CONFIG=arm_cortex-a9.gcc5.2_glibc_openwrt

...
<build output truncated>
...

linux$ scp bin/version_test root@192.168.2.140:/tmp/
```

Then, shell into the Matchstiq and execute the application.

```
linux$ ssh root@192.168.2.140
root@OpenWrt-sn760B:~# /tmp/version_test
1 card(s) found: 0 in use, 1 available!
Card IDs currently used     :
Card IDs currently available: 0
Info: initializing 1 card(s)...
SKIQ[20100]: <INFO> libsidekiq v4.17.x (gd0f29444f)
version_test[20100]: <INFO> Sidekiq card 0 is serial number=45822, hardware MPCIE C (rev C), product␣
↪SKIQ-MPCIE-002 (PCIe) (part ES004206-C0-00)
version_test[20100]: <INFO> Sidekiq card 0 firmware v2.9
version_test[20100]: <INFO> Sidekiq card 0 FPGA v3.13.1, (date 20060518, FIFO size 16k)
version_test[20100]: <INFO> Sidekiq card 0 is configured for an internal reference clock
version_test[20100]: <INFO> Loading calibration data for Sidekiq PCIe, card 0
**********************************************************
* libsidekiq v4.17.x
**********************************************************
**********************************************************
* Sidekiq Card 0
  Card
        accelerometer present: true
        part type: PCIe
        part info: ES004206-C0-00
        serial: 45822
        xport: PCIe
        GPSDO: not supported by card
  FPGA
        version: 3.13.1
        git hash: 0x0a4725c4
        build date (yymmddhh): 20060518
        tx fifo size: 16k samples
  FW
        version: 2.9
  RF
        reference clock: internal
        reference clock frequency: 40000000 Hz

version_test[20100]: <INFO> Unlocking card 0
```

6.2.6 Developing for the NVIDIA Jetson TX1/TX2/Xavier

The Sidekiq SDK already supports the processor architecture of the NVIDIA Jetson, however a cross-compiler is needed to support building custom applications. The toolchain may be downloaded from Linaro using the following

link:

https://releases.linaro.org/components/toolchain/binaries/6.3-2017.05/aarch64-linux-gnu/

To verify correct toolchain installation, navigate to the Sidekiq SDK directory and compile the test applications specifying a `BUILD_CONFIG` of `aarch64.gcc6.3`. The applications may then be copied to the NVIDIA Jetson unit and executed.

Note: On the NVIDIA Jetson TX1 / TX2 platform, FPGA programming or kernel module reloading causes a kernel panic after PCI coherent memory is exhausted. In its default configuration, it may happen in as few as two times. Refer to this post (https://devtalk.nvidia.com/default/topic/1023501/pci_alloc_consistent-memory-leak-on-tx2-/) for details on how to patch the Jetson's TX2 kernel.

6.2.7 Developing for the Sidekiq Z2

The Sidekiq Z2 platform (see https://epiqsolutions.com/rf-transceiver/sidekiq-z2/ for more details) is a small form factor Linux system (including ARM CPU and Xilinx FPGA using the Zynq) and software defined radio platform that utilizes libsidekiq for SDR functionality. This means libsidekiq is used to develop custom applications for the Sidekiq Z2 platform.

The Sidekiq SDK already supports the processor architecture of the Sidekiq Z2, however a cross-compiler is needed to support building custom applications. The Z2 is loaded with a Board Support Package (BSP); depending on which version of the Z2 BSP is targeted, a different cross-compiler and toolchain are required. Please note that as of version 3.0.0 of the Z2 BSP, the cross-compiler has changed and applications compiled for the 2.0.0 BSP will not work on the 3.0.0 BSP (and vice-versa). Both toolchains may be downloaded from the Epiq Solutions Support forum:

Z2 BSP v2.0 and below	https://support.epiqsolutions.com/viewtopic.php?f=148&t=2455
Z2 BSP v3.0 and above	https://support.epiqsolutions.com/viewtopic.php?f=148&t=3189

The toolchains are offered in either a Debian or RPM format and are installed at a specific location on the development machine and symlinked as expected by the Sidekiq SDK. These toolchain were downloaded from the Xilinx SDK and packaged by Epiq for distribution.

To verify correct toolchain installation, navigate to the Sidekiq SDK directory and compile the test applications specifying a `BUILD_CONFIG` of `arm_cortex-a9.gcc4.9.2_gnueabi` (for v2.0.0 or below of the BSP) or `arm_cortex-a9.gcc7.2.1_gnueabihf` (for v3.0.0 and above of the BSP). The applications may then be copied to the Sidekiq Z2 unit and executed (as shown in the next code block). Before executing applications on the Sidekiq Z2, it is imperative no other application is using the Sidekiq card.

First, copy the executable to the Sidekiq Z2 using `scp`.

```
linux$ cd sidekiq_sdk_v4.17.x/test_apps
linux$ make BUILD_CONFIG=arm_cortex-a9.gcc7.2.1_gnueabihf
...
<build output truncated>
...
linux$ scp bin/version_test root@192.168.3.1:/tmp/
```

Then, shell into the Sidekiq Z2 and execute the application.

```
linux$ ssh root@192.168.3.1

root@sidekiqz2:~# /tmp/version_test
SKIQ[1163]: <INFO> Need to perform full initialization
SKIQ[1163]: <INFO> Performing detection of cards
```

```
SKIQ[1163]: <INFO> Sidekiq card detection completed successfully!
SKIQ[1163]: <INFO> Preliminary initialization complete, continue full initialization
1 card(s) found: 0 in use, 1 available!
Card IDs currently used     :
Card IDs currently available: 0
Info: initializing 1 card(s)...
SKIQ[1163]: <INFO> libsidekiq v4.17.x (gddb0a7bf7)
version_test[1163]: <INFO> Sidekiq card 0 is serial number=8028, Z2 (rev B) (part ES023201-B0-00)
version_test[1163]: <WARNING> FPGA capabilities indicate no support for reading/writing flash for␣
↪card 0
version_test[1163]: <INFO> Sidekiq card 0 FPGA v3.14.1, (date 21042019, FIFO size unknown)
version_test[1163]: <INFO> Sidekiq card 0 is configured for an internal reference clock
version_test[1163]: <INFO> Loading calibration data for Sidekiq Z2, card 0
******************************************************
* libsidekiq v4.17.x
******************************************************
******************************************************
* Sidekiq Card 0
  Card
        accelerometer present: false
        part type: Z2
        part info: ES023201-B0-00
        serial: 8028
        xport: custom
        GPSDO: not supported by card
  FPGA
        version: 3.14.1
        git hash: 0x0cb0dec6
        build date (yymmddhh): 21042019
        tx fifo size: unknown
  RF
        reference clock: internal
        reference clock frequency: 40000000 Hz

version_test[1163]: <INFO> Unlocking card 0
```

6.2.8 Developing for the Matchstiq Z3u

The Matchstiq Z3u platform (see https://epiqsolutions.com/rf-transceiver/matchstiq-z for more details) is a small form factor software defined radio platform that has a wideband transceiver supported via libsidekiq. This means libsidekiq is used to develop custom applications for the Matchstiq Z3u platform.

The Sidekiq SDK already supports the processor architecture of the Matchstiq Z3u platforms, however a cross-compiler is needed to support building custom applications. A minimum of Sidekiq SDK v4.15.0 is required for any Matchstiq Z3u development. The toolchain may be downloaded from Linaro using the following link:

https://releases.linaro.org/components/toolchain/binaries/6.3-2017.05/aarch64-linux-gnu/

To verify correct toolchain installation, navigate to the Sidekiq SDK directory and compile the test applications specifying a `BUILD_CONFIG` of `aarch64.gcc6.3`. The applications may then be copied to the Matchstiq Z3u unit and executed.

First, copy the executable from the host to the Matchstiq Z3u using `scp`.

```
host$ cd sidekiq_sdk_v4.17.x/test_apps
host$ make BUILD_CONFIG=aarch64.gcc6.3
```

```
...
<build output truncated>
...
host$ scp bin/version_test sidekiq@192.168.0.15:/tmp/
```

Then, shell into the Sidekiq Z3u and execute the application.

```
host$ ssh sidekiq@192.168.0.15

sidekiq@z3u:~$ /tmp/version_test
1 card(s) found: 0 in use, 1 available!
Card IDs currently used    :
Card IDs currently available: 0
Info: initializing 1 card(s)...
SKIQ[3396]: <INFO> libsidekiq v4.15.0
version_test[3396]: <INFO> Sidekiq card 0 is serial number=9X0J, Z3U (rev B) (part ES032201-B0-00)
version_test[3396]: <WARNING> FPGA capabilities indicate no support for reading/writing flash for␣
↪card 0
version_test[3396]: <INFO> Sidekiq card 0 FPGA v3.14.0, (date 20081217, FIFO size unknown)
version_test[3396]: <INFO> Sidekiq card 0 is configured for an internal reference clock
version_test[3396]: <INFO> Loading calibration data for Sidekiq Z3U, card 0
******************************************************
* libsidekiq v4.15.0
******************************************************
******************************************************
* Sidekiq Card 0
  Card
      accelerometer present: true
      part type: Z3U
      part info: ES032201-B0-00
      serial: 9X0J
      xport: custom
  FPGA
      version: 3.14.0
      git hash: 0x1eefb308
      build date (yymmddhh): 20081217
      tx fifo size: unknown
  RF
      reference clock: internal
      reference clock frequency: 40000000 Hz
version_test[3396]: <INFO> Unlocking card 0
```

6.3 Assessing Throughput Performance

Since the Sidekiq card can be used in a wide variety of host systems, the throughput capabilities of the Sidekiq varies greatly. Included with the prebuilt Sidekiq test applications are a set of benchmarking test applications. These benchmarking applications can be used to determine the theoretical maximum throughput performance of a particular host system. Additionally, various Sidekiq parameters can be adjusted to observe impact in performance.

6.3.1 Receive Performance

The `rx_benchmark` application provides insight into a receive-only application's maximum theoretical benchmark on a specific host system. The user can specify the number of handles specified to use and whether the blocking receive capability is used. The application then monitors timestamp errors for each enabled channel to validate receive throughput performance.

If a desired run time or minimum error count is desired, the `--threshold` and `--time` parameters can be used for the application. If the threshold criteria was met, a `0` is returned by the application.

6.3.2 Receive Performance Example

Here is an example of running the benchmark application with both RX A1 and A2 enabled at a rate of 20Msps for 5 seconds. The return code of `0` indicates that the threshold criteria was met.

```
$ ./rx_benchmark --card 0 --handle both --rate 20000000 --threshold 1 --time 5
rx_benchmark[27478]: <INFO> libsidekiq v4.0.0, RF IC library 2.5.2
rx_benchmark[27478]: <INFO> Sidekiq card 0 is serial number=30281, hardware MPCIE C, product SKIQ-MPCIE-001
rx_benchmark[27478]: <INFO> Sidekiq card 0 is configured for an internal reference clock
rx_benchmark[27478]: <INFO> Sidekiq card 0 firmware v2.2
rx_benchmark[27478]: <INFO> Sidekiq card 0 FPGA v3.3, (date 16122919, FIFO size 3)
rx_benchmark[27478]: <INFO> card 0: number of tx channels supported 1, number of rx channels supported 2
Receive throughput: 148 MB/s (# RxA1 timestamp errors 0) (# RxA2 timestamp errors 0)
Receive throughput: 153 MB/s (# RxA1 timestamp errors 0) (# RxA2 timestamp errors 0)
Receive throughput: 153 MB/s (# RxA1 timestamp errors 0) (# RxA2 timestamp errors 0)
Receive throughput: 153 MB/s (# RxA1 timestamp errors 0) (# RxA2 timestamp errors 0)
Receive throughput: 153 MB/s (# RxA1 timestamp errors 0) (# RxA2 timestamp errors 0)
rx_benchmark[27478]: <INFO> unlocking card 0
$ echo $?
0
```

Here is an example of running the benchmark application with both RX A1 and A2 enabled at a rate of 30Msps for 5 seconds. The return code of `1` indicates that the threshold criteria was not met.

```
$ ./rx_benchmark --card 0 --handle both --rate 30000000 --threshold 1 --time 5
rx_benchmark[27504]: <INFO> libsidekiq v4.0.0, RF IC library 2.5.2
rx_benchmark[27504]: <INFO> Sidekiq card 0 is serial number=30281, hardware MPCIE C, product SKIQ-MPCIE-001
rx_benchmark[27504]: <INFO> Sidekiq card 0 is configured for an internal reference clock
rx_benchmark[27504]: <INFO> Sidekiq card 0 firmware v2.2
rx_benchmark[27504]: <INFO> Sidekiq card 0 FPGA v3.3, (date 16122919, FIFO size 3)
rx_benchmark[27504]: <INFO> card 0: number of tx channels supported 1, number of rx channels supported 2
Receive throughput: 191 MB/s (# RxA1 timestamp errors 4898) (# RxA2 timestamp errors 4902)
Receive throughput: 198 MB/s (# RxA1 timestamp errors 9968) (# RxA2 timestamp errors 9981)
Receive throughput: 198 MB/s (# RxA1 timestamp errors 15041) (# RxA2 timestamp errors 15063)
Receive throughput: 198 MB/s (# RxA1 timestamp errors 20107) (# RxA2 timestamp errors 20148)
Receive throughput: 198 MB/s (# RxA1 timestamp errors 25176) (# RxA2 timestamp errors 25228)
rx_benchmark[27504]: <INFO> unlocking card 0
$ echo $?
1
```

6.3.3 Transmit Performance

The `tx_benchmark` application provides insight into a transmit-only application's maximum theoretical benchmark on a specific host system. The user can specify the number of threads to use (`1` implies synchronous transmit mode,

more than 1 selects asynchronous transmit mode) and the block size. This allows for a user to evaluate how different transmit configuration options impact the overall transmit throughput on a system.

If a desired run time or minimum error count is desired, the `--threshold` and `--time` parameters can be used for the application. If the threshold criteria was met, a `0` is returned by the application.

Transmit Performance Example

Here is an example of running the `tx_benchmark` application in asynchronous transmit mode with 4 threads, a block size of 16380, and at a rate of 45Msps for 5 seconds. The return code of `0` indicates that the threshold criteria was met.

```
$ ./tx_benchmark --block-size 16380 --threshold 5 --time 5 --threads 4 --rate 45000000
Info: number of samples is 16380 (65536 bytes)
tx_benchmark[27680]: <INFO> libsidekiq v4.0.0, RF IC library 2.5.2
tx_benchmark[27680]: <INFO> Sidekiq card 0 is serial number=30281, hardware MPCIE C, product SKIQ-MPCIE-001
tx_benchmark[27680]: <INFO> Sidekiq card 0 is configured for an internal reference clock
tx_benchmark[27680]: <INFO> Sidekiq card 0 firmware v2.2
tx_benchmark[27680]: <INFO> Sidekiq card 0 FPGA v3.3, (date 16122919, FIFO size 3)
tx_benchmark[27680]: <INFO> card 0: number of tx channels supported 1, number of rx channels supported 2
Setting sample rate to 45000000
Send throughput: 175 MB/s (# underruns 1)
Send throughput: 171 MB/s (# underruns 1)
Send throughput: 171 MB/s (# underruns 1)
Send throughput: 171 MB/s (# underruns 1)
Send throughput: 171 MB/s (# underruns 1)
Sending complete
Cleaning up
tx_benchmark[27680]: <INFO> unlocking card 0
$ echo $?
0
```

Here is an example of running the `tx_benchmark` application in synchronous transmit mode with 4 threads, a block size of 16380, and at a rate of 45Msps for 5 seconds. The return code of `1` indicates that the threshold criteria was not met.

```
$ ./tx_benchmark --block-size 16380 --threshold 5 --time 5 --threads 1 --rate 45000000
Info: number of samples is 16380 (65536 bytes)
tx_benchmark[27690]: <INFO> libsidekiq v4.0.0, RF IC library 2.5.2
tx_benchmark[27690]: <INFO> Sidekiq card 0 is serial number=30281, hardware MPCIE C, product SKIQ-MPCIE-001
tx_benchmark[27690]: <INFO> Sidekiq card 0 is configured for an internal reference clock
tx_benchmark[27690]: <INFO> Sidekiq card 0 firmware v2.2
tx_benchmark[27690]: <INFO> Sidekiq card 0 FPGA v3.3, (date 16122919, FIFO size 3)
tx_benchmark[27690]: <INFO> card 0: number of tx channels supported 1, number of rx channels supported 2
Setting sample rate to 45000000
Send throughput: 159 MB/s (# underruns 5140)
Send throughput: 152 MB/s (# underruns 11176)
Send throughput: 154 MB/s (# underruns 16406)
Send throughput: 155 MB/s (# underruns 21748)
Send throughput: 155 MB/s (# underruns 27068)
Sending complete
Cleaning up
tx_benchmark[27690]: <INFO> unlocking card 0
$ echo $?
1
```

6.3.4 Transceive

The `xcv_benchmark` application provides insight into a transceiver application's maximum theoretical benchmark on a specific host system. The user can specify the number of threads to use for transmission (`1` implies synchronous transmit mode, more than `1` selects asynchronous transmit mode), the transmit block size, and whether to receive samples with the blocking mode. This allows for a user to evaluate how different streaming configuration options impact the overall throughput on a specific host system. Both transmit underruns as well as receive timestamp errors are monitored to determine the performance of the system.

If a desired run time or minimum error count is desired, the `--threshold` and `--time` parameters can be used for the application. If the threshold criteria was met, a `0` is returned by the application.

Transceive Performance Example

Here is an example of running the benchmark application in asynchronous transmit mode with 4 TX threads, a TX block size of 16380, using the blocking receive mode, and at a rate of 30Msps for 5 seconds. The return code of `0` indicates that the threshold criteria was met.

```
$ ./xcv_benchmark --threads 4 --block-size 16380 --blocking --rate 30000000 --time 5 --threshold 5
xcv_benchmark[28256]: <INFO> libsidekiq v4.0.0, RF IC library 2.5.2
xcv_benchmark[28256]: <INFO> Sidekiq card 0 is serial number=30281, hardware MPCIE C, product SKIQ-MPCIE-001
xcv_benchmark[28256]: <INFO> Sidekiq card 0 is configured for an internal reference clock
xcv_benchmark[28256]: <INFO> Sidekiq card 0 firmware v2.2
xcv_benchmark[28256]: <INFO> Sidekiq card 0 FPGA v3.3, (date 16122919, FIFO size 3)
xcv_benchmark[28256]: <INFO> card 0: number of tx channels supported 1, number of rx channels supported 2
Send throughput: 117 MB/s (# underruns 3)
Receive throughput: 115 MB/s (# timestamp errors 0)
Send throughput: 114 MB/s (# underruns 3)
Receive throughput: 115 MB/s (# timestamp errors 0)
Send throughput: 114 MB/s (# underruns 3)
Receive throughput: 115 MB/s (# timestamp errors 0)
Send throughput: 114 MB/s (# underruns 3)
Receive throughput: 115 MB/s (# timestamp errors 0)
Send throughput: 114 MB/s (# underruns 3)
Receive throughput: 115 MB/s (# timestamp errors 0)
Waiting for packets to complete transfer, num_pkts 9210, num_complete 9160
Packet send completed!
xcv_benchmark[28256]: <INFO> unlocking card 0
$ echo $?
0
```

Here is an example of running the benchmark application in asynchronous transmit mode with 1 TX thread (synchronous mode), a TX block size of 16380, using the blocking receive mode, and at a rate of 30Msps for 5 seconds. The return code of `1` indicates that the threshold criteria was not met.

```
$ ./xcv_benchmark --threads 1 --block-size 16380 --blocking --rate 30000000 --time 5 --threshold 5
xcv_benchmark[28267]: <INFO> libsidekiq v4.0.0, RF IC library 2.5.2
xcv_benchmark[28267]: <INFO> Sidekiq card 0 is serial number=30281, hardware MPCIE C, product SKIQ-MPCIE-001
xcv_benchmark[28267]: <INFO> Sidekiq card 0 is configured for an internal reference clock
xcv_benchmark[28267]: <INFO> Sidekiq card 0 firmware v2.2
xcv_benchmark[28267]: <INFO> Sidekiq card 0 FPGA v3.3, (date 16122919, FIFO size 3)
xcv_benchmark[28267]: <INFO> card 0: number of tx channels supported 1, number of rx channels supported 2
Send throughput: 114 MB/s (# underruns 191)
Receive throughput: 115 MB/s (# timestamp errors 0)
Send throughput: 114 MB/s (# underruns 472)
Receive throughput: 115 MB/s (# timestamp errors 0)
```

(continues on next page)

(continued from previous page)

```
Send throughput: 114 MB/s (# underruns 1048)
Receive throughput: 115 MB/s (# timestamp errors 0)
Send throughput: 114 MB/s (# underruns 1056)
Receive throughput: 115 MB/s (# timestamp errors 0)
Send throughput: 114 MB/s (# underruns 1534)
Receive throughput: 115 MB/s (# timestamp errors 0)
Packet send completed!
xcv_benchmark[28267]: <INFO> unlocking card 0
$ echo $?
1
```

6.4 DKMS

6.4.1 What is DKMS?

In general, Linux drivers (kernel modules) need to be compiled to run with a specific version of the Linux kernel. While many Linux distributions ship updated drivers alongside new versions of the kernel, any third-party or user-added modules are not recompiled by default. Upon rebooting the system to use the new kernel version, these modules can not load due to a version mismatch and require manual installation. As this can be annoying and time consuming, the Dynamic Kernel Module Support (DKMS) framework was developed by the Linux community.

DKMS is a tool for Linux-based systems that allows kernel modules to be automatically recompiled upon the installation of a new kernel version. This helps to ensure that the kernel modules are always available for the running kernel without needing to download them or request a new version from the vendor.

6.4.2 What systems does DKMS work on?

While DKMS works on most major Linux distributions, most do not include it in a default installation and require the installation of extra software packages. For example, Ubuntu requires the installation of the `dkms` package from the official repositories while CentOS requires the addition of the third-party EPEL repository before installing the `dkms` package.

6.4.3 How is Epiq using DKMS?

Sidekiq software defined radios use several kernel modules to communicate between the host and the radio card.

In the past, Sidekiq software installs have included pre-compiled versions of the kernel modules for a list of specific kernel versions - usually the latest kernel versions of the long-term support branches of popular Linux distributions. While this is adequate at time of a libsidekiq release, most Linux distributions frequently release newer kernel version requiring Epiq to supply updated drivers upon request (on our support forums, https://support.epiqsolutions.com).

As of libsidekiq v4.15.0, all of the Epiq kernel modules provide optional DKMS support. Precompiled kernel modules will continue to be bundled with libsidekiq releases, and Epiq will certainly supply updated modules upon request. Customers may also install the DKMS packaged versions of these modules to ensure that appropriate kernel modules are always used without the need for manual recompilation.

6.4.4 Are there any licensing requirements for using DKMS support?

Several of Epiq's kernel modules are licensed under the GPL and source code and DKMS support are freely provided. These modules include:

- pci_manager
- sidekiq_uart
- sidekiq_gps

Some of Epiq's kernel modules contain proprietary and/or sublicensed code. These modules include:

- dmadriver
- skiq_platform_driver

As DKMS modules recompile as needed, they require the source code of the module to be bundled in the DKMS package. Therefore, as of libsidekiq v4.15.0, DKMS support for these modules requires the purchase of a sublicense for the DMA Driver source code. Please contact the Epiq sales department (sales@epiqsolutions.com) or your account executive for more information on sublicensing.

6.4.5 How are the Epiq DKMS modules installed?

With the exception of DKMS modules requiring a license, all other DKMS modules are included in the default libsidekiq install (v4.15.0 and higher). As these are optional components, they must be manually installed and enabled to take advantage of them.

Please note that not all Linux distributions come with the DKMS subsystem installed by default; please check your distribution's documentation for more information on DKMS and installation / configuration.

The DKMS packages reside in the drivers/dkms of the libsidekiq install (typically ~/sidekiq_image_current/). Each DKMS module has its own package, and all DKMS packages should be installed to ensure that all of the modules are kept up-to-date. There are several ways to install the DKMS modules:

- Automatically install all of the appropriate packages:
  - Run the install-dkms-packages.sh script, which installs the proper packages for the currently running Linux Distribution
- Manually install packages appropriate for OS through its package manager:
  - Each module has its own package file
    - deb/*.deb for Ubuntu & Debian-based systems
    - rpm/*.rpm for CentOS / Fedora / RHEL-based systems
  - Packages can be installed from the command line (dpkg or yum, depending on Linux distribution) or through the GUI
- Manually install packages through the source packages:
  - Uncompress the driver source package using the tar and/or xz command(s)
    - Source packages may be found in the source/ directory
  - Run the install-dkms.sh script found inside the uncompressed directory (will likely need to run as root user or via sudo)

In order to upgrade the DKMS modules - for example, when a new version of libsidekiq comes out - please follow the above steps for installing the DKMS module. If source packages were originally used to install the DKMS modules, please remove them (see directions below) before installing the new versions.

Installation of the sublicensed DMA Driver DKMS module should use the same instructions as either of the manual steps the above; please note that this module bundles both the dma_driver and skiq_platform_device into one package.

6.4.6 How to check the status of the Epiq DKMS modules?

From the command line, run the command `dkms status`. Depending on which module(s) were installed, one or more of the following should be listed:

- `sidekiq_gps`
- `sidekiq_uart`
- `pci_manager`
- `dmadriver`
- `skiq_platform_device`

This command verifies that the listed DKMS modules are installed and enabled, and lists the current source version and the kernel version(s) that the modules were automatically built for.

6.4.7 How are the Epiq DKMS modules removed?

If the OS packages were installed (e.g. through the `.deb` or `.rpm` installation steps listed above), then remove the packages through the OS's package manager. This can typically be done through the GUI or on the command line (typically `apt` or `yum`).

If the installation was done through the source packages, run the `remove-dkms.sh` scripts found in each of the compressed source packages (will likely need to run as `root` user or via `sudo`).

6.4.8 How are the Epiq DKMS modules loaded?

By default, the libsidekiq installation automatically attempts to load the needed kernel modules on system startup using the `load_sidekiq_drivers.sh` script (found in the `drivers/` directory of the libsidekiq install). With the addition of DKMS support, if one of the pre-built kernel modules cannot be found for the currently running kernel, the system will attempt to load the DKMS-based modules (if installed).

6.5 Advanced Topics

6.5.1 Adjusting the DMA Ring Buffer Packet Count (Linux only)

Sidekiq radios that use the PCI / Thunderbolt bus as a transport interact with the kernel running on the host system through the DMA Driver kernel module. This kernel module handles the memory transfer of I/Q samples from the radio directly into the host's memory using Direct Memory Access (DMA); the kernel module preallocates a chunk of host memory and organizes it into a ring buffer, each element of which contains a "packet" of I/Q sample data from the radio.

By default, the number of I/Q sample data packets is set to:

- 1024 for Matchstiq S1x / S2x radios
- 2048 for all other PCI / Thunderbolt based radios

(Please note that Sidekiq Z2 and Matchstiq Z3u do not use the DMA Driver kernel module)

Each ring buffer entry is 4096 bytes, so this effectively means that the DMA Driver on a host using 2048 ring buffer packets has 8MB of I/Q sample buffer space before the sample buffer runs out of space and overflows (leading to the `skiq_rx_status_error_overrun` condition when calling `skiq_receive()`). If the specified receive sample rate is fairly low and the application using the Sidekiq reads samples frequently enough, this may be a reasonable amount of

buffer space. However, higher sample rate applications may require more available buffer space in order to capture samples without frequent overruns. For example, with a 100Msps sample rate the default 8MB ring buffer can hold roughly 20 ms worth of I/Q sample data; the user application must consistently read samples from the radio at a faster rate than this to ensure that there are no overruns in the sample buffer.

One option to help prevent overruns is to increase the sample buffer size, which can be done by increasing the number of entries in the DMA ring buffer; this can be done by specifying the `RingBufferPacketCount` module parameter when loading the DMA Driver module. Using the `modinfo` command, here is an example of the module parameters of the DMA Driver on an x86 host:

```
username@host:~/sidekiq_image_current/driver/5.4.0-39-generic$ modinfo dmadriver.ko
filename:       /home/username/sidekiq_image_current/driver/5.4.0-39-generic/dmadriver.ko
version:        5.4.1.0
description:    Northwest Logic PCI Express DMA Driver
license:        Proprietary
author:         Pro Code Works, LLC
srcversion:     AE21B1718769B0830391B7C
alias:          pci:v000019AAd00002280sv*sd*bc*sc*i*
alias:          pci:v000019AAd00005832sv*sd*bc*sc*i*
alias:          pci:v000019AAd00007021sv*sd*bc*sc*i*
alias:          pci:v000019AAd00007011sv*sd*bc*sc*i*
alias:          pci:v000019AAd0000E004sv*sd*bc*sc*i*
depends:        skiq_platform_device
retpoline:      Y
name:           dmadriver
vermagic:       5.4.0-39-generic SMP mod_unload
parm:           UseMSI:Use MSI [1 = TRUE|0 = FALSE] (default 1) (int)
parm:           UseMSIX:Use MSIX [1 = TRUE|0 = FALSE] (default 0) (int)
parm:           UseMSIMulti:Use MSI Multi-Vector [1 = TRUE|0 = FALSE] (default 1) (int)
parm:           UseIntCtrl:Use Interrupt Control [1 = TRUE|0 = FALSE] (default 1) (int)
parm:           DMADescriptorsPerEngine:Number of DMA Descriptors per Engine (int)
parm:           RingBufferPacketCount:Number of packets in the ring buffer (default 2048)
```

(Note that the `RingBufferPacketCount` parameter displays the default number of ring buffer entries selected for the host.)

In order to increase the number of ring buffer entries, this number can be increased. For example:

```
username@host:~/$ sudo insmod $HOME/sidekiq_image_current/driver/$(uname -r)/dmadriver.ko␣
↪RingBufferPacketCount=16384
```

Setting `RingBufferPacketCount=16384` increases the sample buffer size by eight times over the default settings (to 64MB) and provides approximately 160 ms of I/Q sample buffer space (given the 100Msps example given above). However, this also dramatically increases the amount of dedicated kernel memory needed on the host system (as DMA operations happen in kernel space), which may not be available on certain hosts. If the `RingBufferPacketCount` value is set too high, the desired amount of buffer space may not be able to be allocated which can result in the module loading but failing to allocate the requested amount of memory. This error will show up in the system logs:

```
[  735.453649] DMAD: NorthWest Logic PCI Express DMA Driver 5.4.2.0
[  735.453650] DMAD:   Build (Sep 30 2020-21:54:17)
[  735.453650] DMAD:  Message logging enabled.
[  735.453675] DMAD: Config: Setting ring buffer packet count to 128000
[  735.453676] DMAD: Warning: DMADescriptorsPerEngine value (8192) too low for ring buffer packet count;␣
↪expanding to 512000
[  735.456140] DMAD: Config: Device name is DMAD0
[  735.456170] DMAD 0000:02:00.0: enabling device (0000 -> 0002)
[  735.458788] DMAD: Config: Bar 0, pMemRangePhys=0xdf010000, BarCfg=0xdf010000
[  735.458789] DMAD: Config: BAR[0] Set to Register Type at Addr 0x000000000dd1c813
```

(continues on next page)

(continued from previous page)

```
[  735.458790] DMAD: Config: BAR[0] Register 32-bit PhysAddr=0xDF010000 VirtAddr=000000000dd1c813 Len=0x10000
[  735.458791] DMAD: Config: Bar 1, pMemRangePhys=0xdf002000, BarCfg=0xdf002000
[  735.458791] DMAD: Config: BAR[1] Set to Mem Type
[  735.458792] DMAD: Config: BAR[1] Memory   32-bit PhysAddr=0xDF002000 VirtAddr=00000000a2c89dac Len=0x2000
[  735.458793] DMAD: Config: Bar 2, pMemRangePhys=0xdf000000, BarCfg=0xdf000000
[  735.458793] DMAD: Config: BAR[2] Set to Mem Type
[  735.458794] DMAD: Config: BAR[2] Memory   32-bit PhysAddr=0xDF000000 VirtAddr=000000000b4b9284 Len=0x2000
[  735.458803] DMAD: Config: Found a Packet Send Type DMA Engine
[  735.458803] DMAD: Config: Allocating 512000 descriptors sizeof 32
[  735.458865] DMAD: ERROR: Unable to allocate 16384000 DMA descriptors for DMA channel 0
[  735.458865] DMAD: ERROR: DMA initialization for adapter 0 failed, cannot use device
[  735.458940] DMAD: ERROR: Unable to attach to adapter
```

If this error occurs, it is likely that the PCI transport will be inaccessible and the Sidekiq card will fail to be detected over PCI (on Sidekiq mPCIe and m.2 cards, the USB transport will still continue to function). However, as the DMA Driver is a separate component, no warning about this condition will be shown from libsidekiq; therefore, when experimenting with this parameter, it is important to verify through the system logs that the DMA Driver module loaded without errors. When this error occurs, It is recommended to unload the driver (via the `rmmod dmadriver`) command and attempt to reload it using a smaller value for `RingBufferPacketCount`.

Be aware that the following message in the system logs is not fatal, but it does indicate that the `DMADescriptorsPerEngine` parameter is being automatically adjusted to accommodate the specified `RingBufferPacketCount` parameter:

```
username@host:~/$ dmesg
...
[1307015.430356] DMAD: Config: Setting ring buffer packet count to 8192
[1307015.430356] DMAD: Warning: DMADescriptorsPerEngine value (8192) too low for ring buffer packet count;␣
↪expanding to 32768
```

As this is an advanced configuration setting, as of libsidekiq v4.15.0 there is no default way to adjust the number of ring buffer entries when the DMA Driver is loaded on system startup. One option is to unload and reload the DMA Driver kernel module with the desired ring buffer size after system startup. Another option would be to modify the `../sidekiq_image_current/driver/load_sidekiq_driver.sh` file to include the `RingBufferPacketCount` parameter when loading the DMA Driver (it is highly recommended to make a backup copy of this script before editing it).

6.5.2 Configuring Sidekiq Drivers Using a Driver Configuration File

As of the libsidekiq v4.17.2 System Release, driver parameters can be automatically set using a per-module configuration file[1]. The `modinfo -F parm <module_name>` command lists modifiable parameters which can be loaded from a configuration file. The man page for `modinfo` can be used for more info. The name of the configuration file should match that of the module being configured; for instance, configuration options for the 'dmadriver' module should be placed into 'dmadriver.conf'. The configuration file needs to be saved in the `.../sidekiq_image_current/driver/driver_config` folder. Two examples of configuring module parameters can be found within the preinstalled example configuration file (`.../sidekiq_image_current/driver/driver_config/sidekiq_drivers.conf.example`):

```
### x86 device driver example configuration
options dmadriver DMADescriptorsPerEngine=8192
options dmadriver RingBufferPacketCount=2048

### ARM device driver example configuration
options dmadriver DMADescriptorsPerEngine=4096
options dmadriver RingBufferPacketCount=1024
```

[1] This functionality is not available on Matchstiq S1x and S2x products.

7 Hardware Information

7.1 Detailed RF Port Configuration

The tables below provides a summary of RF ports and configurations available per Sidekiq product, revision and RF mode. For details on configuring the RF port and mode, refer to *RF Port Configuration* (page 34).

Table 7.1: RF Port Mode

Product	Supports Fixed?	Supports TRX?
Sidekiq mPCIe	Yes	No
Sidekiq M.2 rev B	Yes	No
Sidekiq M.2 rev B TDD	No	Yes
Sidekiq M.2 rev C	Yes	Yes
Sidekiq X2	Yes	No
Sidekiq Z2	Yes	Yes
Sidekiq X4	Yes	No
Matchstiq Z3u	Yes	Yes
Sidekiq NV100	Yes	Yes

Table 7.2: RF Port Mapping for Sidekiq mPCIe, M.2, and Stretch

	Sidekiq mPCIe	Sidekiq M.2 rev B	Sidekiq M.2 rev B TDD	Sidekiq M.2 rev C	Sidekiq Stretch
`skiq_rx_hdl_A1`	Jxxx_RX1	J2	J2*	J2	J2 or J1*
`skiq_rx_hdl_A2`	Jxxx_TX1RX2	J3	J3*	J3	N/A
`skiq_tx_hdl_A1`	Jxxx_TX1RX2	J1	J2*	J1 or J2*	J1
`skiq_tx_hdl_A2`	N/A	J4	J3*	J4 or J3*	N/A

* - requires the RF port to be configured to TRX via `skiq_write_rf_port_config()`. RX or TX is controlled via `skiq_write_rf_port_operation()`.

Table 7.3: RF Port Mapping for Sidekiq X2 and X4

	Sidekiq X2	Sidekiq X4
skiq_rx_hdl_A1	J2	J2
skiq_rx_hdl_A2	J1	J3
skiq_rx_hdl_B1	J3	J6
skiq_rx_hdl_B2	N/A	J7
skiq_rx_hdl_C1	N/A	J3
skiq_rx_hdl_D1	N/A	J7
skiq_tx_hdl_A1	J4	J4
skiq_tx_hdl_A2	J5	J1
skiq_tx_hdl_B1	N/A	J8
skiq_tx_hdl_B2	N/A	J5

Table 7.4: RF Port Mapping for Sidekiq Z2 and Matchstiq Z3u

	Sidekiq Z2	Matchstiq Z3u
skiq_rx_hdl_A1	J1 or J2 or J3**	J1 or J2***
skiq_rx_hdl_A2	N/A	J2
skiq_tx_hdl_A1	J1	J2
skiq_tx_hdl_A2	N/A	N/A

** - requires the RF port to be configured to TRX via `skiq_write_rf_port_config()`. RX or TX is controlled via `skiq_write_rf_port_operation()`.

*** - J2 when in TRx mode, J1 when in fixed mode, mode configured with `skiq_write_rf_port_config()`

Table 7.5: RF Port Mapping for Sidekiq NV100

	Sidekiq NV100
skiq_rx_hdl_A1	J1
skiq_rx_hdl_A2	J2 or J1*
skiq_rx_hdl_B1	J2 or J1*
skiq_tx_hdl_A1	J1
skiq_tx_hdl_B1	J2

* - RxB1 may be routed to J1, but only when operating in fixed mode (i.e. not TRX)

7.2 FPGA Programming

7.2.1 Transport Layer Requirements

In order to make use of a given Sidekiq, it is imperative to have the FPGA configured with the correct transport implementation. For example, an application that is designed to use the PCIe transport interface requires the FPGA to be loaded with an image implementing the PCIe components necessary for I/Q streaming. By default, a Sidekiq is configured with a PCIe-capable FPGA image. Applications designed to make use of PCIe as the primary transport layer will therefore require no additional actions in order to properly function. If another transport layer is required, such as USB, and/or the original PCIe image was erased, Epiq Solutions provides pre-generated FPGA image files to each customer within the Sidekiq image that can be saved into on board flash memory or dynamically loaded into the FPGA RAM at run time. Currently, the following files are provided in `sidekiq_image_current/fpga/`.

- `sidekiq_image_mpcie_xport_pcie_latest.bin`: Configures the FPGA on a mini PCIe Sidekiq card for use over PCIe.
- `sidekiq_image_mpcie_xport_usb_latest.bin`: Configures the FPGA on a mini PCIe Sidekiq card for use over USB.
- `sidekiq_image_m2_xport_pcie_latest.bit`: Configures the FPGA on an m.2 Sidekiq card for use over PCIe.
- `sidekiq_image_m2_xport_usb_latest.bit`: Configures the FPGA on an m.2 Sidekiq card for use over USB.
- `sidekiq_image_x2_xport_pcie_latest.bin`: Configures the FPGA on Sidekiq X2 card for use over PCIe on the HiTech Global K-800 FMC Carrier (https://www.xilinx.com/products/boards-and-kits/1-96z8ax.html)
- `sidekiq_image_x2_xcku115_xport_pcie_latest.bin`: Configures the FPGA on Sidekiq X2 card for use over PCIe on the HiTech Global K-800 FMC Carrier (with Xilinx Kintex Ultrascale KU115) (https://www.xilinx.com/products/boards-and-kits/1-96z8ax.html)
- `sidekiq_image_x4_xport_pcie_latest.bin`: Configures the FPGA on Sidekiq X4 card for use over PCIe on the HiTech Global K-800 FMC Carrier (https://www.xilinx.com/products/boards-and-kits/1-96z8ax.html)
- `sidekiq_image_x4_xcku115_xport_pcie_latest.bin`: Configures the FPGA on Sidekiq X4 card for use over PCIe on the HiTech Global K-800 FMC Carrier (with Xilinx Kintex Ultrascale KU115) (https://www.xilinx.com/products/boards-and-kits/1-96z8ax.html)
- `sidekiq_image_m2_2280_xport_pcie_latest.bit`: Configures the FPGA on a Sidekiq Stretch card for use over PCIe.
- `sidekiq_image_nv100_xport_pcie_latest.bit`: Configures the FPGA on a Sidekiq NV100 card for use over PCIe
- `sidekiq_image_z3u_latest.bit`: Configures the FPGA on a Matchstiq Z3u for use over custom transport

Additionally, it is possible to create custom FPGA images for Sidekiq that implement user defined signal processing routines over either the PCIe or USB transport layer. For further details regarding custom FPGA images, refer to associated Sidekiq product's FPGA Developer's Manual.

7.2.2 Updating the FPGA

The FPGA image can be loaded into the RAM (mPCIe / m.2 / Z2 / Z3u for full reconfiguration and Stretch for partial reconfiguration) or it can be loaded from the flash. The FPGA image is automatically loaded from flash when the Sidekiq is powered up. A new image (full or partial) can be loaded into RAM for Sidekiq mPCIe, m.2, Z2, Stretch, or Matchstiq Z3u via the `prog_fpga` test application or with the `skiq_prog_fpga_from_file()` API.

In addition to modifying just the runtime FPGA configuration, the FPGA image loaded from flash can be updated with the `store_user_fpga` test application or via the `skiq_save_fpga_config_to_flash()` function [*]. Additionally, the FPGA configuration can be reloaded via the `skiq_prog_fpga_from_flash()` API (not supported for Sidekiq Z2 or

Matchstiq Z3u). The FPGA configuration that is stored in flash is automatically loaded from flash when the Sidekiq card is powered up.

At this time, the Annapolis Micro Systems' WILDSTAR (WB3XZD) FMC Carrier's FPGA bitstream can only be updated by using Vivado and the Xilinx Virtual Cable interface. Please refer to the FPGA PDK documentation for further details.

When developing an application that updates the FPGA, it is highly suggested to take steps to prevent function calls that write to the FPGA or flash memory (such as the `skiq_prog_fpga_from_file()` or `skiq_save_fpga_config_to_flash()` API calls) from being interrupted and causing a partial write to occur. For UNIX-based systems, this involves masking and unmasking the `SIGINT` and `SIGTERM` signals such that these signals – which would normally halt execution of the program – are delayed until after the FPGA programming is finished. Also, it is advised to temporarily mask the `SIGINT` and `SIGTERM` signals before calling the `skiq_init()` API call. `skiq_init()` may create additional threads that can receive and process signals unless specified by the parent thread (as threads inherit signal masks from their parent thread). See the `prog_fpga.c` test application source code for more details.

* - storing to flash not supported for Sidekiq Z2 and Matchstiq Z3u.

7.2.3 FPGA Images in Flash *

All Sidekiq products' flash memory have enough capacity to store at least two FPGA images: a golden (or fallback) image and a user image. As long as the user FPGA image is valid, this will be the image loaded from flash upon either power up or when the user requested that the FPGA image is reloaded from flash via the `skiq_prog_fpga_from_flash()` function.

If the user FPGA image stored in flash is not a valid configuration, the Sidekiq will automatically fallback to configure the FPGA with the golden image stored in flash. The golden FPGA image does not provide full Sidekiq capabilities. However, it does provide the ability to access the flash via the FPGA to store a new user FPGA image in flash. A new user FPGA image can be stored to flash via the `store_user_fpga` test application or via the `skiq_save_fpga_config_to_flash()` function. A golden FPGA image must already be present in flash prior to updating the user flash image. The presence of the golden image in flash can be tested via the `skiq_read_golden_fpga_present_in_flash()`. Updating an FPGA image saved in flash can be performed via the PCIe or USB interfaces as long as a valid golden image has already been programmed.

The golden FPGA image is a fallback option in case the user FPGA image in flash is either corrupted or incomplete. There is no ability for the user to program their own golden FPGA bitstream. The golden FPGA image is programmed either in the factory or while applying a Sidekiq system update if a golden image has not already been saved to flash.

* - storing to flash not supported for Sidekiq Z2 and Matchstiq Z3u.

FPGA Configuration Flash Slots

Starting in libsidekiq v4.12.0, for certain Sidekiq products, there are additional storage locations in the on-board flash for FPGA bitstreams. This means that multiple FPGA bitstreams can be stored in flash and the FPGA can be configured from any slot that contains a valid bitstream. Each flash configuration slot contains the FPGA bitstream and has 64 bits of metadata associated with the slot. The user may use this metadata to create a mapping between the stored bitstream and its intended purpose. For example, the user can store an abbreviated hash of the bitstream in the metadata so that a full dump of the flash contents is not necessary when verifying what bitstream is stored in the config slot.

There are six new API functions that provide access to the flash configuration slots and they are listed here with a brief description:

- `skiq_prog_fpga_from_flash_slot()` – provides the caller the ability to configure the FPGA from the specified slot

- `skiq_save_fpga_config_to_flash_slot()` – stores an FPGA bitstream at the specified slot
- `skiq_verify_fpga_config_in_flash_slot()` – verifies the contents of the flash configuration slot against the specified `FILE stream`
- `skiq_read_fpga_config_flash_slot_metadata()` – reads the 64-bit metadata associated with the specified flash configuration slot
- `skiq_find_fpga_config_flash_slot_metadata()` – iterates through the available flash configuration slots to find the specified metadata
- `skiq_read_fpga_config_flash_slots_avail()` – provides the caller with the number of available slots for a given card

As of libsidekiq v4.12.0, there are three Sidekiq products that may have more than one configuration slot available. The Sidekiq Stretch (M.2-2280) has **six** configuration slots available. The Sidekiq X2 and Sidekiq X4, when part of the HTG-K800 FMC carrier (Xilinx KU060 only), each have **two** configuration slots available. The Sidekiq NV100 has **six** configuration slots available as of libsidekiq v4.17.0.

7.3 Power consumption states (mPCIe, m.2)

As of libsidekiq v3.5.0, the Sidekiq has three distinct power consumption states: Idle, RX, and RX/TX. Each state is entered / exited automatically by way of existing API function calls and initialization levels. Using an initialization level of `skiq_xport_init_level_full` in a call to `skiq_init()` brings the RFIC out of low power mode and into the RX state. However, using an initialization level of `skiq_xport_init_level_basic` in a call to `skiq_init()` does not change the RFIC power state.

The Sidekiq transmit line-up stays powered down after a call to `skiq_init()` until `skiq_write_tx_LO_freq()` is called for the first time. It will then only be powered down (along with the rest of the RFIC) on a call to `skiq_exit()`. *Sidekiq Power State Consumption (in Watts)* (page 105) shows the different power consumption states and their associated transitions.

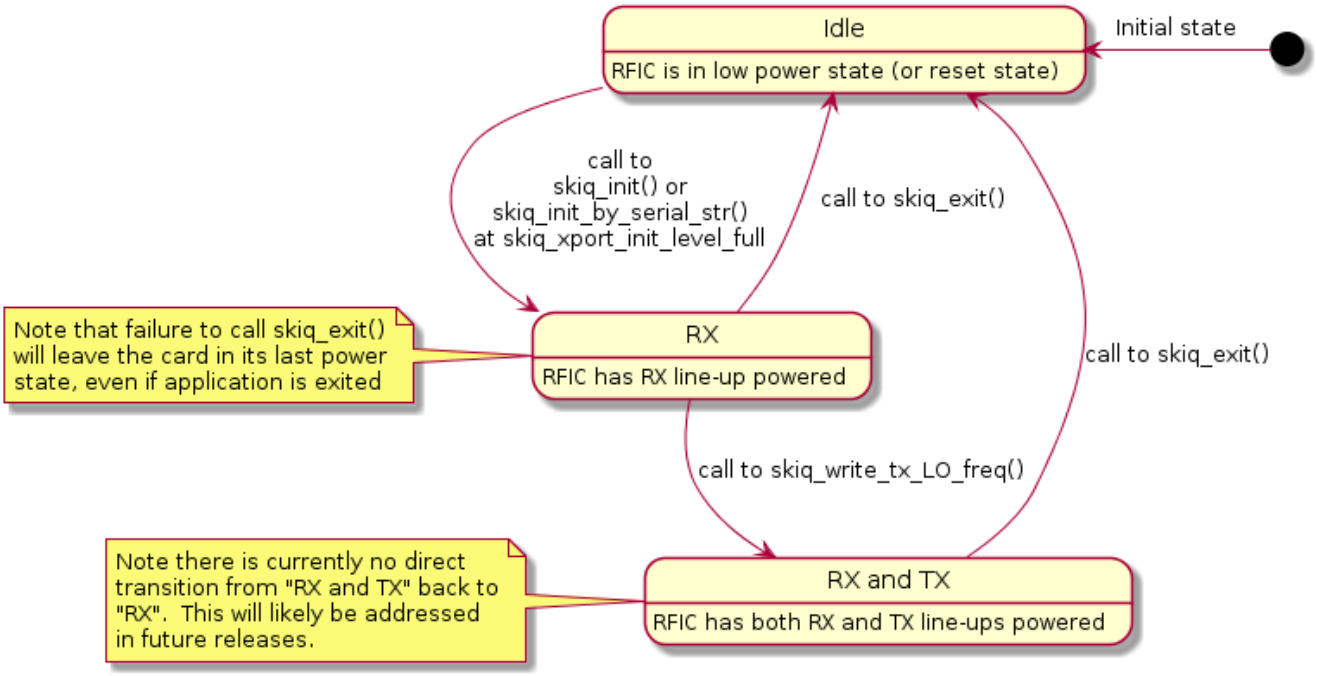

Fig. 7.1: Sidekiq power state transitions

There can be significant power consumption savings related to the three distinct states. outlines a typical example of both a miniPCIe Sidekiq card and an m.2 Sidekiq card in the various power consumption states.

Note: These power consumption measurements will vary based on sample rate, transmit LO frequency, and transmit attenuation.

The Idle state measurements were performed after a clean application shutdown. The RX state measurements were performed using `rx_benchmark` with a sample rate of 45Msps. The RX/TX state measurements were performed using `tx_samples_async` in four different configurations:

- fc=850MHz, tx_atten=0, sample_rate=10MHz, bw=10MHz, block-size=16380
- fc=850MHz, tx_atten=359, sample_rate=10MHz, bw=10MHz, block-size=16380
- fc=3.85GHz, tx_atten=0, sample_rate=10MHz, bw=10MHz, block-size=16380
- fc=3.85GHz, tx_atten=359, sample_rate=10MHz, bw=10MHz, block-size=16380

In libsidekiq version 3.4.1 and before, the Idle state's power consumption varied depending on the previous state (RX or "RX / TX") since the RFIC did not enter a low power state.

Table 7.6: Sidekiq Power State Consumption (in Watts)

Hardware	libsidekiq version	Idle	RX	RX / TX
Sidekiq miniPCIe	v3.4.1 and before	2.0 to 2.3	2.2	2.1 to 2.4
	v3.5.0 and after	1.1	2.0	2.1 to 2.4
Sidekiq m.2	v3.4.1 and before	2.4 to 2.6	2.5	2.4 to 2.8
	v3.5.0 and after	1.4	2.25	2.4 to 2.7

7.4 Sidekiq X4 - Methods of LO frequency tuning

There are three methods of tuning the LO frequency on the Sidekiq X4, with each method having advantages and disadvantages that need to be considered based on the user's system requirements. Each mode is described below with respect to configuration, re-tuning, and differences from other modes. At the end, some real-world tuning duration measurements are shown to help demonstrate what can typically be expected in a given Sidekiq X4 system. Please note that the timing tests were performed with a Sidekiq X4 installed on a HiTech Global HTG-K800 PCIe carrier card, and housed in a PCIe-to-Thunderbolt 3 chassis. An Intel NUC7 was connected to the PCIe-to-Thunderbolt 3 chassis over Thunderbolt 3. The test utilities were based upon the libsidekiq v4.13.0 release.

7.4.1 `hop_on_timestamp` (FPGA triggered)

The `skiq_freq_tune_mode_hop_on_timestamp` mode provides the fastest LO tuning times, using the FPGA to pulse a GPIO connected to the ADRV9009 RFIC at a user-defined timestamp in order to tune the LO frequency at a precise time. Once configured, the entire LO tuning operation is performed automatically by the FPGA without any additional software intervention.

Configuration for hopping on timestamp

1. Configure the frequency tune mode to `skiq_freq_tune_mode_hop_on_timestamp` using `skiq_write_rx_freq_tune_mode()`
2. Configure the frequency hop list and initial hop index using `skiq_write_rx_freq_hop_list([freqA, freqB, freqC], freqA_idx)`

Perform frequency retune at a specified RF timestamp

The RFICs used on Sidekiq X4 and libsidekiq impose a restriction on how to perform frequency hopping. A hop index can only be written to a "mailbox" slot on the RFIC. A hop operation executes a retune to the frequency in the "next" slot, then delivers the index from the "mailbox" to the "next" slot. It acts much like a 2 element deep FIFO that must have 1 or 2 indices enqueued and cannot go empty. Even though this approach is only required for Sidekiq X4, libsidekiq presents a consistent interface across all radio products.

In this example, note that `freqA` is the first tuned frequency since `freqA_idx` is configured as the initial index in Configuration step 2 from above.

1. Prepare the next hop index using `skiq_write_next_rx_freq_hop(<card>, <hdl>, freqB_idx)`
2. Perform the re-tune to `freqA` using `skiq_perform_rx_freq_hop(<card>, <hdl>, <rf_timestamp>)`
3. Once `<rf_timestamp>` has passed, tuning to `freqA` has completed
4. Prepare the next hop index using `skiq_write_next_rx_freq_hop(<card>, <hdl>, freqC_idx)`
5. Perform the re-tune to `freqB` using `skiq_perform_rx_freq_hop(<card>, <hdl>, <rf_timestamp>)`
6. Once `<rf_timestamp>` has passed, tuning to `freqB` has completed

Caveats

During standard LO tuning operations, software is in control of the entire process (including proper selection of the RF preselect filter). When frequency hopping on a timestamp, the FPGA is in control, and doesn't currently coordinate selection of the RF preselect filter as part of the process. Thus, if the user-defined frequency hopping list spans more than one of the preselect filter bands (shown below), then libsidekiq automatically chooses the "bypass" filter path, where no preselect filtering is applied. This way the tuning speed is optimized for speed. Here are the preselect bands on Sidekiq X4 for convenience:

- `skiq_filt_390_to_620MHz`
- `skiq_filt_540_to_850MHz`
- `skiq_filt_770_to_1210MHz`
- `skiq_filt_1130_to_1760MHz`
- `skiq_filt_1680_to_2580MHz`
- `skiq_filt_2500_to_3880MHz`
- `skiq_filt_3800_to_6000MHz`

7.4.2 `hop_immediate` (software triggered)

The `skiq_freq_tune_mode_hop_immediate` mode provides the second fastest LO tuning speed on Sidekiq X4. This mode uses software to tell the ADRV9009 RFIC to perform the LO frequency hop operation immediately upon reception of a SPI command initiated by libsidekiq. This method is not as deterministic as the hop-on-timestamp mode, since the host's CPU load and transport layer between the CPU and FPGA (typically PCIe) affects communication latency.

Configuration for hopping immediately

1. Configure the frequency tune mode to `skiq_freq_tune_mode_hop_immediate` using `skiq_write_rx_freq_tune_mode()`

2. Configure the frequency hop list and initial hop index using `skiq_write_rx_freq_hop_list([freqA, freqB, freqC], freqA_idx)`

Perform frequency retune immediately

The RFICs used on Sidekiq X4 and libsidekiq impose a restriction on how to perform frequency hopping. A hop index can only be written to a "mailbox" slot on the RFIC. A hop operation executes a retune to the frequency in the "next" slot, then delivers the index from the "mailbox" to the "next" slot. It acts much like a 2 element deep FIFO that must have 1 or 2 indices enqueued and cannot go empty. Even though this approach is only required for Sidekiq X4, libsidekiq presents a consistent interface across all radio products.

In this example, note that `freqA` is the first tuned frequency since `freqA_idx` is configured as the initial index in Configuration step 2 from above.

1. Prepare the next hop index using `skiq_write_next_rx_freq_hop(<card>, <hdl>, freqB_idx)`
2. Perform the re-tune to `freqA` using `skiq_perform_rx_freq_hop(<card>, <hdl>, 0)` – specifying `0` as the timestamp means "immediate"
3. Prepare the next hop index using `skiq_write_next_rx_freq_hop(<card>, <hdl>, freqC_idx)`
4. Perform the re-tune to `freqB` using `skiq_perform_rx_freq_hop(<card>, <hdl>, 0)` – specifying `0` as the timestamp means "immediate"

Differences from `hop_on_timestamp`

In the "hop immediate" mode, the RF frontend is controlled by software so the preselect filters are used _even if the hop list spans more than one frequency band. This is different from "hop on timestamp" where the preselect filters are configured only _once_ whenever the hop list is written. This also means that the "hop immediate" mode performance is limited by the speed of the SPI transactions to configure the RF frontend.

7.4.3 Standard tune

The standard tune mode is the default LO tuning mode for libsidekiq, and is implemented for all Sidekiq products. Compared to the previously described hopping modes, this is the slowest LO tuning mode on Sidekiq X4, but also provides the most flexibility in terms of RF performance since the user can control the type(s) of receive calibration and choose between automatic (re-tune initiated) or manual calibration (user-initiated). The standard tune mode requires no special configuration to start using it: the user simply calls `skiq_write_rx_LO_freq()` as needed. Similar to the "hop-immediate" mode, the standard tune mode engages the appropriate RF frontend preselect filter configuration during the retune automatically based on the requested LO frequency.

1. Configure the frequency tune mode to `skiq_freq_tune_mode_standard` using `skiq_write_rx_freq_tune_mode()` (Note: this is only needed if a different tune mode was previously set by the user)
2. Use `skiq_write_rx_LO_freq()` as needed

Differences from `hop_immediate`

The standard tune mode performs receive calibration steps on the ADRV9009 RFIC as recommended by Analog Devices which can take upwards of one second to complete. These calibration steps ensure optimal quadrature error correction (thus minimizing image/sideband artifacts), as well as minimizing any DC offset in the received signal. The calibration operation is performed whenever the LO frequency is tuned more than 100MHz away from the current LO frequency, which is also based on recommendations from Analog Devices. The operation is also performed when the LO frequency crosses an RF PLL resolution boundary (see Talise User Guide for details). There are

multiple calibration knobs (refer to `skiq_rx_cal_type_t` starting with libsidekiq v4.13.0) that can be independently enabled/disabled through the libsidekiq API, thus allowing a user to strike the appropriate balance of performance and tuning speed.

7.4.4 Comparisons between tuning modes

The API call duration varies depending on which tuning mode is being used. The next table provides ranges of performance based on the specific configuration described in the introduction.

Typical API call duration (Sidekiq X4 only)

Tuning Mode	`skiq_write_rx_LO_freq`
`skiq_freq_tune_mode_standard` (> 3GHz)	7 - 8 ms[2]
`skiq_freq_tune_mode_standard` ($\leq$ 3GHz)	9 - 17 ms[2]

In the `skiq_freq_tune_mode_hop_immediate` tuning mode, the two API calls `skiq_write_next_rx_freq_hop()` and `skiq_perform_freq_hop()` are typically performed back-to-back so the total duration is shown below.

Tuning Mode	`skiq_write_next_rx_freq_hop` and `skiq_perform_rx_freq_hop`
`skiq_freq_tune_mode_hop_immediate`	400 - 500 us

In the `skiq_freq_tune_mode_hop_on_timestamp` tuning mode, the two API calls `skiq_write_next_rx_freq_hop()` and `skiq_perform_freq_hop()` can be separated in time so each call's duration is shown below.

Tuning Mode	`skiq_write_next_rx_freq_hop`	`skiq_perform_rx_freq_hop`
`skiq_freq_tune_mode_hop_on_timestamp`	~250 us	~40 us

[2] Only when retuning less than 100MHz from current LO frequency, otherwise calibration is performed and takes time to complete.

8 Errata

8.1 Software Errata

For a complete and up-to-date list of Software Errata, please visit https://support.epiqsolutions.com/viewtopic.php?f=115&t=2536

8.1.1 Errata SW1

Issue Description

The HTG-K800 FMC carrier's flash contents may fail to program successfully (~7%). When the flash fails to program in this manner, the user bitstream will subsequently fail to configure the FPGA and the fallback FPGA bitstream is ignored. If a user encounters this issue, currently the only means of recovery is to overwrite the carrier card flash contents using the JTAG pod.

Affected Product(s):

- Sidekiq X2 PDK (Thunderbolt 3 Chassis)

Impact Version(s):

- libsidekiq v4.2.x
- libsidekiq v4.4.x
- libsidekiq v4.6.0

Resolution:

When using the `store_user_fpga` test application, use the `--verify` command line option to verify the bitstream is correct after programming it to flash. In the software API, use the `skiq_verify_fpga_config_from_flash()` function after using `skiq_save_fpga_config_to_flash()` and check that the return code of `skiq_verify_fpga_config_from_flash()` is 0.

Fixed in:

libsidekiq v4.7.0 contains a fix for this intermittent flash erase / programming behavior. However, as with any unattended update procedure, it is still recommended that the verification step be performed where critical operations are in use.

8.1.2 Errata SW2

Issue Description

The expected locations of the fallback (aka golden) and user bitstreams have been updated in the HTG-K800 FMC carrier card's flash starting in libsidekiq v4.7.0 and later.

Affected Product(s):

- Sidekiq X2 PDK (Thunderbolt 3 Chassis)

Affected API Function(s):

- `skiq_save_fpga_config_to_flash`

Impact Version(s):

- libsidekiq v4.2.x
- libsidekiq v4.4.x
- libsidekiq v4.6.x

Resolution:

In order to better support FPGA bitstream fallback and to support full-sized FPGA bitstreams, it was necessary to change the locations of the fallback and user bitstreams.

The `sidekiq_hardware_updater_for_v4.7.0.sh` script will take care of flashing the fallback and user bitstreams to their new locations (only if needed).

After the bitstreams are relocated, if an application compiled against an impacted version (see above) of libsidekiq uses `skiq_save_fpga_config_to_flash()`, an error will be emitted stating that "*golden FPGA not present, cannot update user FPGA image*". A user must use libsidekiq v4.7.0 or later in conjunction with relocated bitstreams in order to store a user bitstream to non-volatile memory.

If a user wishes to revert the relocation, they may run a `sidekiq_hardware_updater` from a release prior to v4.7.0.

8.1.3 Errata SW3

Issue Description:

A flash transaction on the HTG-K800 FMC carrier can fail intermittently and without detection. A user may see the warning message "*Quad SPI operation bit needed assertion*" emitted, however the software does not react correctly to address the warning.

Affected Product(s):

- Sidekiq X2 PDK (Thunderbolt 3 Chassis)
- Sidekiq X4 PDK (Thunderbolt 3 Chassis)

Affected API Function(s):

- skiq_save_fpga_config_to_flash
- skiq_verify_fpga_config_from_flash

Affected Test Application(s) / Utilities:

- sidekiq_hardware_updater.sh
- store_user_fpga

Impact Version(s):

- libsidekiq v4.2.x
- libsidekiq v4.4.x
- libsidekiq v4.6.x
- libsidekiq v4.7.x
- libsidekiq v4.8.x
- libsidekiq v4.9.0 through v4.9.3

Resolution:

Danger: **Do not use** the `sidekiq_hardware_updater.sh` utility prior to **v4.9.4** to upgrade or downgrade an X2 PDK or X4 PDK system. **Only** use the updaters and installers from the System Update `v4.9.4 20190503` and later

Danger: **Do not use** the `store_user_fpga` test application prior to **v4.9.4** on an X2 PDK or X4 PDK system. Use the `store_user_fpga` test application from the **v4.9.4** SDK or the run-time/complete installer from the System Update `v4.9.4 20190503` and later

If your application uses either of the affected API functions, you must update to libsidekiq SDK v4.9.4 or there is a risk of incorrect flash behavior (corruption).

8.1.4 Errata SW4

Issue Description:

Memory allocated for `skiq_tx_block_t` may not be aligned on a memory page boundary. When a misaligned `skiq_tx_block_t` is transferred to the FPGA, it is mishandled, resulting in corrupt samples inadvertently added to the sample block being transmitted.

Affected Product(s):

- Sidekiq X4 HTG platforms with 8 lane PCIe support (HTG-K800, HTG-K810)
- FPGA bitstreams >= v3.14.1

Affected API Function(s) / Macros:

- `skiq_tx_block_allocate`
- `skiq_tx_block_allocate_by_bytes`
- `SKIQ_TX_BLOCK_INITIALIZER`
- `SKIQ_TX_BLOCK_INITIALIZER_BY_BYTES`
- `SKIQ_TX_BLOCK_INITIALIZER_BY_WORDS`

Affected Test Application(s) / Utilities:

- `tx_samples`, `tx_benchmark`, `xcv_benchmark`

Impact Version(s):

- libsidekiq v4.7.0 through v4.16.2

Resolution:

Libsidekiq v4.17.0 will page align memory when allocating for Tx blocks. By default, a page size of 4K is used, but can be overridden by `#defining SKIQ_TX_BLOCK_MEMORY_ALIGN`. If an application uses any of the affected API functions, it *must* be updated to libsidekiq SDK v4.17.0 or there is a risk of incorrect data being transmitted by the FPGA.

8.1.5 Errata SW5

Issue Description:

When transmitting on timestamps, four samples are erroneously sent when the packet is received, regardless of the specified timestamp. This can be four samples from a previous packet, or samples from the current packet.

Affected Product(s):

- Sidekiq X2 PDK (Thunderbolt 3 Chassis)
- Sidekiq X4 PDK (Thunderbolt 3 Chassis) and Sidekiq X4 PCIe Blade

Affected Test Application(s) / Utilities:

- `tx_samples`, `tx_samples_async`, `fdd_rx_tx_samples`, `tdd_rx_tx_samples`

Impact Version(s):

- FPGA bitstreams version v3.14.1 through v3.15.1

Resolution:

If transmit on timestamp is utilized, please use a FPGA bitstream with a version prior to v3.14.1 when this defect was introduced or version v3.16.1 and later when the issue was resolved.

9 Troubleshooting

9.1 Troubleshooting a Sidekiq Installed in a New Host *

The following subsections address common issues when attempting to use a Sidekiq card in a new host system. If there are any questions on these steps or if additional problems are encountered, contact Epiq Solutions support, *[5]* (page 8).

9.1.1 Observing The LED State (Sidekiq mPCIe and Sidekiq m.2 only)

If the Sidekiq card is not appearing when running the `version_test` application, basic hardware functionality can be determined by monitoring the state of the LEDs on the Sidekiq card. This can be done independent of the drivers or kernel version.

With the Sidekiq installed in the host system, the host can be powered on and the 2 LEDs on the card described as User LED #1 (near the edge of the board) and User LED #2 (towards the middle of the board). These LED locations are highlighted in Figure 2 of the Hardware User's Manual.

When powering up the host, LED #2 turn on and quickly off, and then about a half second later, turn back on. This LED reflects the state of the FPGA configuration being loaded. At some point, either at the same time or later during the boot process, LED #1 should also turn on. This LED reflects the state of the PCIe link. If LED 1 is on, then the PCIe link has been established successfully. If both LEDs are on, then there is a good chance that the hardware will enumerate correctly on the host.

If the LEDs do not behave as described above, contact Epiq Solutions support, *[5]* (page 8), with information on what the behavior of the LEDs are and the host system being used.

9.1.2 Verifying the Hardware Interfaces Detected in Linux

Once the host completes booting into Linux, the USB and PCIe interfaces (if present) of the Sidekiq card can be detected (independent of kernel or driver versions). The USB interface can be checked by running `lsusb`. The output of `lsusb` from the Sidekiq PDK laptop is shown below. The line highlighted below is a Sidekiq mPCIe card.

```
Bus 001 Device 002: ID 8087:8008 Intel Corp.
Bus 002 Device 002: ID 8087:8000 Intel Corp.
Bus 003 Device 002: ID 04b4:1004 Cypress Semiconductor Corp.
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 003 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 004 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 003: ID 0c45:64d0 Microdia
Bus 002 Device 003: ID 8087:07da Intel Corp.
Bus 002 Device 004: ID 0a5c:5801 Broadcom Corp. BCM5880 Secure Applications Processor
```

Not all host systems connect the USB interface on the miniPCIe or M.2 connector to the processor. If *lsusb* does not show the Sidekiq card, it is possible that the host system being used does not have the USB routed properly. If this is observed, contact Epiq Solutions support, *[5]* (page 8), with information on the host system being used. Also note that Sidekiq mPCIe and Sidekiq m.2 have USB interfaces whereas other Sidekiq products do not.

Table 9.1: Sidekiq PCIe PID/VID Identifiers

Product	PID / VID
Sidekiq mPCIe	19AA:E004
Sidekiq m.2	19AA:7021
Sidekiq Stretch (m.2-2280)	19AA:2280
Sidekiq NV100	19AA:2280
Sidekiq X2	19AA:5832
Sidekiq X4	19AA:5832

The PCIe interface can be checked by running `lspci`. The output of `lspci` executed on the Sidekiq PDK laptop is shown below, where a Sidekiq mPCIe card is highlighted.

```
00:00.0 Host bridge: Intel Corporation Xeon E3-1200 v3/4th Gen Core Processor DRAM Controller
00:01.0 PCI bridge: Intel Corporation Xeon E3-1200 v3/4th Gen Core Processor PCI Express x16 Controller
00:02.0 VGA compatible controller: Intel Corporation 4th Gen Core Processor Integrated Graphics Controller
00:03.0 Audio device: Intel Corporation Xeon E3-1200 v3/4th Gen Core Processor HD Audio Controller
00:14.0 USB controller: Intel Corporation 8 Series/C220 Series Chipset Family USB xHCI
00:16.0 Communication controller: Intel Corporation 8 Series/C220 Series Chipset Family MEI Controller #1
00:16.3 Serial controller: Intel Corporation 8 Series/C220 Series Chipset Family KT Controller
00:19.0 Ethernet controller: Intel Corporation Ethernet Connection I217-LM
00:1a.0 USB controller: Intel Corporation 8 Series/C220 Series Chipset Family USB EHCI #2
00:1b.0 Audio device: Intel Corporation 8 Series/C220 Series Chipset High Definition Audio Controller
00:1c.0 PCI bridge: Intel Corporation 8 Series/C220 Series Chipset Family PCI Express Root Port #1
00:1c.2 PCI bridge: Intel Corporation 8 Series/C220 Series Chipset Family PCI Express Root Port #3
00:1c.4 PCI bridge: Intel Corporation 8 Series/C220 Series Chipset Family PCI Express Root Port #5
00:1c.5 PCI bridge: Intel Corporation 8 Series/C220 Series Chipset Family PCI Express Root Port #6
00:1c.6 PCI bridge: Intel Corporation 8 Series/C220 Series Chipset Family PCI Express Root Port #7
00:1c.7 PCI bridge: Intel Corporation 8 Series/C220 Series Chipset Family PCI Express Root Port #8
00:1d.0 USB controller: Intel Corporation 8 Series/C220 Series Chipset Family USB EHCI #1
00:1f.0 ISA bridge: Intel Corporation QM87 Express LPC Controller
00:1f.2 RAID bus controller: Intel Corporation 82801 Mobile SATA Controller [RAID mode]
00:1f.3 SMBus: Intel Corporation 8 Series/C220 Series Chipset Family SMBus Controller
01:00.0 Display controller: Advanced Micro Devices, Inc. [AMD/ATI]
02:00.0 Signal processing controller: Device 19aa:e004
03:00.0 Network controller: Intel Corporation Centrino Advanced-N 6235
0e:00.0 SD Host controller: O2 Micro, Inc. SD/MMC Card Reader Controller
```

If the PCIe interface is not present in the `lspci` output, contact Epiq Solutions support, *[5]* (page 8), with information on the host system being used.

If both the USB and PCIe interfaces are detected by Linux properly, then it is likely that the Sidekiq card will work properly with the host system.

Note: The Sidekiq X2, X4, and Stretch will not be present on the host's USB.

9.1.3 Checking Kernel and Drivers

The Sidekiq drivers are automatically loaded on bootup through either a systemd or init.d script depending on the host operating system. In a systemd configuration, the service script is located at `/lib/systemd/system/sidekiq`.

`service` where as the init.d script is located at `/etc/init.d/sidekiq`. In order for proper behavior, the kernel version used must match the kernel versions listed in *Developing for Alternative Host Platforms* (page 80). To request an alternative kernel or further details on driver validation, contact Epiq Solutions support, *[5]* (page 8).

* - this section does not apply to Sidekiq Z2 and Matchstiq Z3u

9.2 Frequently Asked Questions

Q: What is the maximum rate that I/Q samples can be received from the FPGA to the CPU without dropping any data?

A: Assuming no other application is executing on the CPU, and only simple operations are being performed on the received data (such as storing it in to a RAM buffer), the DMA-based FPGA to CPU interface using the PCIe transport can support a data transfer rate of ~50 Msamples/sec with mPCIe and can reach 61.44 Msamples/sec with m.2. A sample rate of 50 Msamples/sec equates to approximately 200 MB/sec, which approaches the limit of the single lane, Gen 1 PCIe interface. For USB, most systems can expect to run without issues at a data rate of ~10 Msamples/sec assuming that the bus is not used with any other USB devices at the time of sample capture. Note that the maximum throughput for both PCIe and USB is **highly dependent** on the host platform. For Sidekiq Z2, RX streaming on the card can achieve a sample rate of approximately 35 Msamples/sec (depending on CPU use by other applications).

Q: What is the maximum rate that I/Q samples can be transferred between the RF receiver and the FPGA?

A: The digital interface between the A/D converters in each RF receiver and the FPGA support operation up to 61.44 Msamples/sec on Sidekiq mPCIe, m.2, Z2, and NV100, 153.6 Msamples/sec (or 245.76 Msps on RxB1) on Sidekiq X2, and 250 Msamples/sec (or 500 Msps on RxC1 and RxD1) on Sidekiq X4. However, since this rate of data is not always sustainable between the FPGA and the CPU, the FPGA must perform some level of custom processing to reduce this data flow.

Q: Does the hardware support the ability to configure the FPGA registers and stream sample data across only the USB interface?

A: Yes, the Sidekiq mPCIe and m.2 hardware does support streaming sample data across the USB interface, but software currently only supports receiving data. Development is in progress to fully support transmit over USB. For further details please contact Epiq Solutions *[5]* (page 8).

Q: Can CPU architecture X or Linux kernel version Y be used with Sidekiq?

A: Please contact Epiq Solutions' support (*[5]* (page 8)) for any host platform inquiries.

Q: Either `Error: failed calibration for reg 0x247, bit_pos 1, exiting` or `Error: failed calibration for reg 0x05e, bit_pos 7, exiting` is occurring when running an application. What could be the cause of this?

A: These errors most commonly occur if the Sidekiq is configured to use an external reference clock and that reference clock is not present or inadequate. The specifications for the external reference clock requirements are outlined in the Sidekiq Hardware User's manual. The reference clock configuration can be checked with the `skiq_read_ref_clock_select()` API and modified with the `ref_clock` test application provided.

Q: Is GNU Radio supported?

A: Yes, please refer to *[7]* (page 8). There are two general options for `gr-sidekiq` that are available. The preferred method is to use libsidekiq to perform the interfacing to the radio. This version is the most full featured implementation and is available on the master branch. For embedded platforms (such as the Matchstiq S10) not running GNU

radio natively, a source-only block that interfaces with the SRFS application running on the embedded platform is available on the `srfs` branch. This relies on the socket interface presented by the SRFS application to both configure the radio and stream samples.

Q: Is MATLAB supported?

A: MATLAB is not currently supported with libsidekiq, though MATLAB may be used with the Z2 (using the IIO network interface and the Analog Devices BSP). For more information, please refer to Epiq Solutions support *[5]* (page 8).

Q: It appears as though signal frequencies close to DC are being attenuated. Is there any way to disable this?

A: The Sidekiq FPGA performs a DC offset correction using a simple leaky integrator. This is enabled by default but can be disabled through the API call `skiq_write_rx_dc_offset_corr()`. Additionally, there is DC offset correction provided by the RF IC. Disabling the RF IC DC offset correction and tracking is not currently supported.

Q: Why am I encountering various errors when configuring the FPGA to a different transport layer?

A: When programming the FPGA with a bitstream that implements the USB transport layer, errors will be generated if the Sidekiq was previously configured and initialized to operate with PCIe. Likewise errors will occur when transitioning from USB to PCIe. This is expected behavior due to libsidekiq being initialized to operate under constraints that are no longer valid. In order to avoid further errors, it is advised to reboot the host.

Q: Can I disable any prints/logging from the Sidekiq library?

A: Yes, `NULL` can be passed to the `skiq_register_logging()` API to completely disable any log messages. Refer to *Logging* (page 29) for more details.

Q: What should the bandwidth be configured to?

A: The bandwidth typically depends on the end application and the desired characteristics of the signal being received or transmitted. However, in general, it is recommended to limit the bandwidth to a maximum of approximately 80% of the sample rate.

Q: How is the gain configured?

A: The gain index, as an index into the configured gain table, is configured with `skiq_write_rx_gain()`. The approximate mapping of in dB is described in the API description of `skiq_write_rx_gain()`. Additionally, for a more precise mapping in dB based on a per unit calibration is available. For details on using the per unit calibrated data, refer to section *Using receive calibration offsets* (page 40).

Q: My timestamps are slipping, should this be happening?

A: Products that use the AD9361 RFIC will have timestamp slips when using API functions that need to deactivate the sample clock in order to make updates to the radio configuration. This occurs when: updating the LO frequency, updating the sample rate, and running the transmit quadrature calibration. It is recommended to use the system clock - which is not subject to interruptions - if a consistent time source is needed. For a list of functions that affect or are affected by the timestamp please refer to Timestamp slips within AD9361 Products, in the libsidekiq API documentation.

Q: The server card is not detected when using the network transport, what could be the cause?

A: In order to use the Network Transport, the `SKIQ_DAEMON_IP` environment variable must be configured to the server's IP address. Additionally, the `SKIQ_DAEMON_PORT` environment variable must be configured to the port number of the `skiq_daemon`. If the environment variables are configured and the server card still is not detected, ensure that the `skiq_daemon` application is running on the server.

Q: The `version_test` application indicates an FPGA bitstream version of `v0.0.0`, what could be the cause?

A: This can happen on Sidekiq mPCIe or Sidekiq m.2 cards if libsidekiq detects a card's USB interface, but is having trouble interfacing with the card over PCIe. There could be a few underlying reasons that cause trouble interfacing over PCIe. If all of these do not help, please contact Epiq Solutions support, *[5]* (page 8), with information on the host system being used.

- PCIe presence:
  - It may simply be that the host system is not detecting the card over PCIe. Check by using `lspci -d 19aa:` to see if there's the same number of entries for the expected number of Sidekiq cards.
- Device driver:
  - Check to see if the `dmadriver.ko` kernel module is loaded:
    - Running "`lsmod`"
    - Running "`grep dmadriver /proc/modules`"
    - Look through "`dmesg`" looking for entries beginning with "`DMAD`"
- Card manager cache:
  - The card manager stores information it discovered about Sidekiq cards in a shared memory cache and prior to libsidekiq v4.14.0, the card manager would only update its cache during the very first application execution. If applications that use libsidekiq prior to v4.14.0 are the only ones in use, try rebooting the system to clear and refresh the card manager cache. If there's a mix of applications on the system, try using the applications built against libsidekiq v4.14.0 or later.

10 Release Information

10.1 Known Issues / Limitations

There are some features that are either partially supported or unsupported. Below is a list of unsupported or limited features.

10.1.1 PCIe only functions

In general, any function involved with transmitting data is currently supported only through PCIe or a custom transport. Below is the specific list of functions that work only with the PCIe (or custom) interface.

```
int32_t skiq_start_tx_streaming (uint8_t card, skiq_tx_hdl_t hdl)
int32_t skiq_start_tx_streaming_on_1pps (uint8_t card, skiq_tx_hdl_t hdl, uint64_t sys_timestamp)
int32_t skiq_stop_tx_streaming (uint8_t card, skiq_tx_hdl_t hdl)
int32_t skiq_stop_tx_streaming_on_1pps (uint8_t card, skiq_tx_hdl_t hdl, uint64_t sys_timestamp)
int32_t skiq_write_tx_sample_rate_and_bandwidth (uint8_t card, skiq_tx_hdl_t hdl, uint32_t rate, uint32_t␣
↪bandwidth)
int32_t skiq_transmit (uint8_t card, skiq_tx_hdl_t hdl, skiq_tx_block_t *p_block, void *p_user)
```

10.1.2 USB only functions

In general, any function involved in reconfiguring the FPGA on-the-fly is currently only supported through USB (or Sidekiq Z2's custom transport). Below is the specific list of functions that work only when those interfaces are available.

```
int32_t skiq_prog_fpga_from_file (uint8_t card)
```

10.1.3 Limited Capabilities

```
int32_t skiq_read_rx_LO_freq (uint8_t card, skiq_rx_hdl_t hdl, uint64_t *p_freq, double *p_actual_freq)
int32_t skiq_read_tx_LO_freq (uint8_t card, skiq_tx_hdl_t hdl, uint64_t *p_freq, double *p_tuned_freq)
```

- Only returns cached frequency value, not actual tuned frequency

```
skiq_rx_status_t skiq_receive (uint8_t card, skiq_rx_hdl_t * p_hdl, skiq_rx_block_t** pp_data, uint32_t* p_data_
↪len)
```

- For systems using a USB transport FPGA bitstream, all received data packets will have word four of the header – containing *SYSTEM META*, *RFIC CTRL OUT*, *OVRLD*, *HDL* – set to zero.

10.2 Release History

Releases

10.2.1 v4.17.2 - 28-Feb-2022

```
New Features for libsidekiq/FPGA
--------------------------------
o New Features for v4.17.2

  === No new features for v4.17.2 ===

Bug Fixes for libsidekiq/FPGA
-----------------------------
o Bug Fixes for v4.17.2

  === All ===
    - Disable libsidekiq's internal custom transport when a user registers
      their own transport so as not to interfere with device detection
    - Downgrade logging message severity from WARN->INFO on a valid
      operation mode of reduced number of Tx/Rx channels

  === Test Apps ===
    - read_gpsdo.c
        - Set "host" as the default 1PPS source
        - Allow users to configure external 1PPS source
    - tdd_rx_tx_samples.c
        - Add additional command line parameters and utility functions
    - tx_samples_async.c
        - Restructure example application, improve the capability / command line options

  === Sidekiq mPCIe, M.2 ===
    - Report FPGA bitstream version as unavailable instead of v0.0.0 if the
      expected transport is absent

  === Sidekiq NV100 ===
    - Correct the reporting of the Tx channel bandwidth on NV100
        - Related API functions(s):
          o skiq_read_tx_sample_rate_and_bandwidth()

  === Sidekiq X2 and Sidekiq X4 ===
  - FPGA bitstream v3.16.1
    - Fix Errata SW5: Transmit on timestamp will no longer erroneously send
      four samples upon reception of the first packet.
    - X4 transmit at maximum block size (65532) no longer drops packets

  === Sidekiq Z2 and Matchstiq Z3u ===
    - Network Transport:
        - Significantly improve initialization time
```

10.2.2 v4.17.1 - 11-Feb-2022

```
New Features for libsidekiq/FPGA
--------------------------------
o New Features for v4.17.1

  === No new features for v4.17.1 ===

Bug Fixes for libsidekiq/FPGA
-----------------------------
o Bug Fixes for v4.17.1

  === Test Apps ===
    - rx_samples_on_trigger.c
        - Fix output file naming
    - pps_tester.c
        - Fix card detection logic

  === Sidekiq NV100 ===
    - Fix receive path by setting RF input port after every re-tune of the LO
    - Fix last calibrated LO frequency tracking for transmit paths
    - Update RFIC profiles to accurately report RF transmit bandwidth

  === Sidekiq X4 ===
    - Fix internal handle assignment following a re-initialization of a card.
      This can happen in several cases of interacting with the libsidekiq API:
        - If a user calls:
          - skiq_disable_cards() followed by a call to skiq_enable_cards()
          - skiq_exit() followed by a call to skiq_init()
          - skiq_write_ref_clock_select() on its own
          - skiq_write_ext_ref_clock_freq() on its own
          - skiq_prog_fpga_from_flash() on its own

  === Matchstiq Z3u ===
    - Fix setting RF isolation during TRX

  === Sidekiq mPCIe, M.2, Stretch, Z2, and Matchstiq Z3u ===
    - Fix configuring RFIC back to defaults after a re-initialization.  This can
      happen in several cases of interacting with the libsidekiq API.
        - If a user calls:
          - skiq_disable_cards() followed by a call to skiq_enable_cards()
          - skiq_exit() followed by a call to skiq_init()
          - skiq_write_ref_clock_select() on its own
          - skiq_write_ext_ref_clock_freq() on its own
          - skiq_prog_fpga_from_flash() on its own
        - Resolves "Rx filter caching issue after FPGA reprogramming" issue
```

10.2.3 v4.17.0 - 15-Oct-2021

```
New Features for libsidekiq/FPGA
--------------------------------
o New Features for v4.17.0

  === All ===
    - Add support for Sidekiq NV100 rev C
```

(continues on next page)

(continued from previous page)

```
    - Extend on-the-fly reference clock source switching to include frequency
      - Related API function(s):
          o skiq_write_ext_ref_clock_freq()
    - Expose GPSDO lock state through new public API function
      - Related API functions(s):
          o skiq_gpsdo_is_locked()
      - Example Test Application: (new)
          o read_gpsdo
    - Expose Sidekiq card's warp capabilities in skiq_param_t
    - Improve rx_samples_on_trigger.c example application

  === Sidekiq mPCIe, M.2, Stretch, Z2, and Matchstiq Z3u ===
    - Enable user override of AD9361 analog filter settings
      - Related API function(s):
          o skiq_read_rx_analog_filter_bandwidth()
          o skiq_read_tx_analog_filter_bandwidth()
          o skiq_write_rx_analog_filter_bandwidth()
          o skiq_write_tx_analog_filter_bandwidth()

  === Sidekiq M.2 ===
    - Add pull-ups to BOARD_ID pins on FPGA design

  === Matchstiq Z3u ===
    - Significantly increased the sustained receive performance

  === Sidekiq X4 ===
    - Update fdd_rx_tx_samples.c example application to add support for Sidekiq X4

Bug Fixes for libsidekiq/FPGA
-----------------------------
o Bug Fixes for v4.17.0

  === All ===
    - Update skiq_write_rx_dc_offset_corr() to return -ENOTSUP for Sidekiq X2 and X4
    - Fix defect where skiq_gpsdo_read_freq_accuracy() reports success when there's no fix or 1PPS
    - Disable all receive / transmit channels when disabling a specified card
    - Fix defect when reading TCVCXO warp voltage prior to full initialization
    - Fix defect where a card failed initialization if skiq_rx_cal_mode_auto is set
    - Fix defect related to hotplugging when utilizing a custom transport

  === Sidekiq Z2 and Matchstiq Z3u ===
    - Fix defect causing timestamp gaps when using network transport
```

10.2.4 v4.16.2 - 9-Sept-2021

```
New Features for libsidekiq/FPGA
--------------------------------
o New Features for v4.16.2

  === All ===
    - No new features added in v4.16.2

Bug Fixes for libsidekiq/FPGA
-----------------------------
o Bug Fixes for v4.16.2
```

(continues on next page)

(continued from previous page)

```
  === Matchstiq Z3u ===
    - Fix occasional device initialization issue after reprogramming the FPGA
```

10.2.5 v4.16.1 - 9-Jun-2021

```
New Features for libsidekiq/FPGA
--------------------------------
o New Features for v4.16.1

  === Sidekiq M.2 ===
    - Add support for M.2 rev D hardware

Bug Fixes for libsidekiq/FPGA
-----------------------------
o Bug Fixes for v4.16.1

  === Sidekiq Z2 and Matchstiq Z3u ===
    - Fix card manager caching of network transport entries

  === Sidekiq mPCIe, M.2, Stretch, Z2, and Matchstiq Z3u ===
    - Fix defect when specifying non-zero initial_index in calls to
      skiq_write_rx_hop_list() or skiq_write_tx_hop_list()
```

10.2.6 v4.16.0 - 1-Jun-2021

```
New Features for libsidekiq/FPGA
--------------------------------
o New Features for v4.16.0

  === All ===
    - Prevent out-of-specification RF LO tune requests by returning an error
      code.  Check skiq_read_parameters() or the output from `version_test
      --full` to determine the permitted tuning range.
    - Perform a soft reset of all FPGA registers at card initialization and
      exit
    - In FPGA bitstreams v3.15.1 and later, add support for configuring the
      transmit timestamp base for RF Sample Clock or System Clock.  This
      feature only applies when transmitting on timestamp.
       - Related API functions(s):
          o skiq_read_tx_timestamp_base()
          o skiq_write_tx_timestamp_base()

  === Sidekiq Z2 ===
    - Add support for network-based transport (see SDK manual for more
      details)

  === Sidekiq X4 ===
    - Add support on Windows 10 for FPGA designs v3.14.0 and later

  === Sidekiq Stretch ===
    - Add support for GPSDO 1PPS source configuration (matches the system 1PPS
```

(continues on next page)

(continued from previous page)

```
      source configuration) on products that support the FPGA-based GPSDO
      algorithm
       - Related API function(s):
          o skiq_read_1pps_source()
          o skiq_write_1pps_source()

  === Matchstiq Z3u ===
    - Add support for network-based transport (see SDK manual for more
      details)
    - Add GPSDO support
       - Related API function(s):
          o skiq_is_gpsdo_supported()
          o skiq_gpsdo_enable()
          o skiq_gpsdo_disable()
          o skiq_gpsdo_is_enabled()
          o skiq_gpsdo_read_freq_accuracy()
    - Add support for GPSDO 1PPS source configuration (matches the system 1PPS
      source configuration) on products that support the FPGA-based GPSDO
      algorithm
       - Related API function(s):
          o skiq_read_1pps_source()
          o skiq_write_1pps_source()

Bug Fixes for libsidekiq/FPGA
-----------------------------
o Bug Fixes for v4.16.0

  === All ===
    - Fix skiq_read_tx_tone_freq() when only frequency hopping to report
      correct value
    - FPGA bitstream v3.15.1
       - Fix missing timestamp reset related to clock crossing synchronization
         on the register interface

  === Sidekiq mPCIe, M.2, Stretch, Z2, and Matchstiq Z3u ===
    - Fix tracking of DC offset calibration when switching internal RFIC RX
      ports

  === Sidekiq M.2 and Stretch ===
    - FPGA bitstream v3.15.1
       - Fix Windows 10 enumeration related issues for Sidekiq M.2 and Sidekiq
         Stretch
       - Fix errantly flushing subsequent transmit burst packets after
         encountering a late timestamp by limiting flush to full and partial
         (i.e. arriving) packets in the FPGA FIFO

  === dmadriver ===
    - Fix Physical Address Space verification against PCIe address space
       - Addresses issues observed with 3rd-party NVIDIA Jetson carriers
```

10.2.7 v4.15.2 - 31-Mar-2021

```
New Features for libsidekiq/FPGA
--------------------------------
o New Features for v4.15.2
```

(continues on next page)

(continued from previous page)

```
  === sample applications ===
    - Added 'pps_tester.c' sample application

Bug Fixes for libsidekiq/FPGA
-----------------------------
o Bug Fixes for v4.15.2

  === Sidekiq X4 ===
    - Resolved issue where spurs appeared intermittently after continuous
      transmission for a relatively long duration (e.g. multiple seconds)
    - Updated clock termination settings to match recommended values
    - Improved JESD sync procedure

  === Sidekiq X2 ===
    - Updated clock termination settings to match recommended values

  === Sidekiq mPCIe, M.2, Stretch, Z2, Z3u ===
    - Fix off-by-one transmit frequency defect when frequency hopping
    - Fix off-by-one receive filter selection defect when frequency hopping
```

10.2.8 v4.15.1 - 3-Mar-2021

```
New Features for libsidekiq/FPGA
--------------------------------
o New Features for v4.15.1

  === Matchstiq Z3u ===
    - Add support for Matchstiq Z3u rev C hardware

Bug Fixes for libsidekiq/FPGA
-----------------------------
o Bug Fixes for v4.15.1

  === Sidekiq mPCIe, M.2, Stretch ===
    - Fix unaligned packed receive samples after an LO re-tune

  === Matchstiq Z3u ===
    - Prohibit unsupported packed mode setting for Matchstiq Z3u
```

10.2.9 v4.15.0 - 3-Feb-2021

```
New Features for libsidekiq/FPGA
--------------------------------
o New Features for v4.15.0

  === All ===
    - Add preliminary support for Matchstiq Z3u
    - Identify out-of-specification RF LO tune requests by issuing a
      message to user.  Starting with libsidekiq v4.16.0, tune requests
      outside of valid tuning range will result in a return error code.
```

(continues on next page)

(continued from previous page)

```
  === Sidekiq mPCIe ===
    - Add identification of hardware revision E

  === Sidekiq X4 ===
    - Add API function to configure multiple handles in a single call
       - Related API function(s):
          o skiq_write_rx_sample_rate_and_bandwidth_multi()
    - FPGA bitstream v3.14.1 on HTG-K800 and HTG-K810 carriers adds support
      for PCIe Gen3 x8 lane as long as the host supports it

  === Sidekiq Stretch ===
    - Add support for GPSDO
       - Related API function(s):
          o skiq_is_gpsdo_supported()
          o skiq_gpsdo_enable()
          o skiq_gpsdo_disable()
          o skiq_gpsdo_is_enabled()
          o skiq_gpsdo_read_freq_accuracy()

  === dmadriver ===
    - Add DKMS support (if source code is licensed)
    - Add support for Linux kernels v5.6 and v5.8

  === pci_manager, sidekiq_uart, sidekiq_gps ===
    - Add DKMS support
    - Add support for Linux kernels v5.6 and v5.8

Bug Fixes for libsidekiq/FPGA
-----------------------------
o Bug Fixes for v4.15.0

  === All ===
    - Lower 1PPS polling interval to help reduce overruns in the receive
      streams
    - FPGA bitstream v3.14.1
      - Fixes reporting FIFO underruns in dual channel transmit
        applications
      - Fixes streaming synchronization issue on 1PPS where samples started
        flowing up to 1 second prior to function returning control to the
        caller.

  === Sidekiq Z2 ===
    - FPGA bitstream v3.14.1
      - Fixes empty FIFO issue in FPGA that resulted in intermittent
        receive block corruption
      - Fixes errant packed mode in FPGA on platforms that do not support
        packed mode that resulted in timestamp gaps and receive block
        corruption
```

10.2.10 v4.14.2 - 12/09/2020

```
New Features for libsidekiq/FPGA
--------------------------------
o New Features for v4.14.2
```

(continues on next page)

(continued from previous page)

```
  === No new features for v4.14.2 ===

Bug Fixes for libsidekiq/FPGA
-----------------------------
o Bug Fixes for v4.14.2

  === Sidekiq Z2 ===
    - Fix receive streaming after call to skiq_prog_fpga_from_file() on BSP
      v3.1.0

  === Example Applications ===
    - Add missing rf-port-config parameter in tdd_rx_tx_samples.c
    - Add call to skiq_set_rx_transfer_timeout() in rx_samples_on_trigger.c
      when --blocking is specified
    - Bundle prog_fpga in pre-built applications for Sidekiq Z2
```

10.2.11 v4.14.1 - 10/30/2020

```
New Features for libsidekiq/FPGA
--------------------------------
o New Features for v4.14.1

  === No new features for v4.14.1 ===

Bug Fixes for libsidekiq/FPGA
-----------------------------
o Bug Fixes for v4.14.1

  === All ===
    - Internally restore the configured channel mode when calling
      skiq_prog_rfic_from_file() instead of defaulting to
      skiq_chan_mode_single
    - Realign receive streams if an overrun is detected in skiq_receive()

  === Sidekiq Z2 ===
    - Update libsidekiq product capabilities to reflect that Sidekiq Z2 does
      not support packed mode of I/Q sample blocks
```

10.2.12 v4.14.0 - 10/16/2020

```
New Features for libsidekiq/FPGA
--------------------------------
o New Features for v4.14.0

  === All ===
    - Add support for hotplug of Sidekiq devices by identifying addition,
      removal, or replacing a card.  See the Software Development Manual for
      more details.
    - Add support for automatically calling skiq_exit() as a precaution at
      Linux application exit (unintentional or otherwise).
       - New Related API function(s):
          o skiq_set_exit_handler_state()
```

(continues on next page)

(continued from previous page)

```
=== Sidekiq mPCIe, M.2 ===
  - Add support for configuring the RX calibration mode and type mask.
     - Related API function(s):
        o skiq_read_rx_cal_mode()
        o skiq_write_rx_cal_mode()
        o skiq_run_rx_cal()
        o skiq_read_rx_cal_type_mask()
        o skiq_write_rx_cal_type_mask()
        o skiq_read_rx_cal_types_avail()

=== Sidekiq Z2 ===
  - Add support for run-time reference clock source management and control
     - Related API function(s):
        o skiq_write_ref_clock_select()
  - Add support for configuring the RX calibration mode and type mask.
     - Related API function(s):
        o skiq_read_rx_cal_mode()
        o skiq_write_rx_cal_mode()
        o skiq_run_rx_cal()
        o skiq_read_rx_cal_type_mask()
        o skiq_write_rx_cal_type_mask()
        o skiq_read_rx_cal_types_avail()

=== Sidekiq X2 ===
  - Add support for run-time reference clock source management and control
     - Related API function(s):
        o skiq_write_ref_clock_select()

=== Sidekiq X4 ===
  - Add FMC carrier support for HTG-K810 (Xilinx Kintex UltraScale COM
    EXPRESS CPU Card (Type 6))
  - Add support for run-time reference clock source management and control
     - Related API function(s):
        o skiq_write_ref_clock_select()
  - Add support for RFIC pin control mode to allow enabling and disabling
    receivers handles and transmitters handles from the FPGA.
     - Related API function(s):
        o skiq_read_rx_rfic_pin_ctrl_mode()
        o skiq_read_tx_rfic_pin_ctrl_mode()
        o skiq_write_rx_rfic_pin_ctrl_mode()
        o skiq_write_tx_rfic_pin_ctrl_mode()
  - Add support for FPGA bitstreams that have the decimator built for
    RFIC B.  Future FPGA PDKs for Sidekiq X4 on the HTG-K800 will offer a
    build option for decimating samples from RFIC B.

=== Sidekiq Stretch ===
  - Add support for run-time reference clock source management and control
     - Related API function(s):
        o skiq_write_ref_clock_select()
  - Add support for configuring the RX calibration mode and type mask.
     - Related API function(s):
        o skiq_read_rx_cal_mode()
        o skiq_write_rx_cal_mode()
        o skiq_run_rx_cal()
        o skiq_read_rx_cal_type_mask()
```

(continues on next page)

(continued from previous page)

```
          o skiq_write_rx_cal_type_mask()
          o skiq_read_rx_cal_types_avail()

  === Example Applications ===
    - Convert fdd_rx_tx_samples.c and tdd_rx_tx_samples.c to use arg_parser.
    - Add optional Sidekiq card temperature logging to rx_benchmark.c,
      tx_benchmark.c, and xcv_benchmark.c

Bug Fixes for libsidekiq/FPGA
-----------------------------
o Bug Fixes for v4.14.0

  === Sidekiq M.2, Stretch ===
      - FPGA bitstream v3.14.0 fixes reporting of transmit block underruns
        with the dual channel transmit mode.  In previous bitstreams,
        underruns were not always reported.

  === Sidekiq X4 ===
      - Fix frequency hopping mode when using RxA2 with skiq_chan_mode_dual
      - Fix frequency hopping on timestamp on RFIC B (any of RxB1, RxB2, TxB1,
        TxB2)

  === dmadriver ===
      - Fix registration of TTY and GPIO devices if the Sidekiq card is using
        FPGA bitstream v3.14.0 or later.  At this time, this only affects the
        Sidekiq Stretch.

  === Example Applications ===
      - Switch calling order between LO retuning API functions and sample rate
        configuration API functions.  Refer to the latest Software Development
        Manual for more details on why calling order can matter.
```

10.2.13 v4.13.1 - 09/10/2020

```
New Features for libsidekiq/FPGA
--------------------------------
o New Features for v4.13.1

  === Sidekiq mPCIe ===
      - Add support for Sidekiq mPCIe rev D hardware

Bug Fixes for libsidekiq/FPGA
-----------------------------
o Bug Fixes for v4.13.1

  === Sidekiq ===
      - Fix known issue where frequency hopping would fail on a Sidekiq's RxA2
        handle when channel mode is set to skiq_chan_mode_dual

  === Sidekiq mPCIe / M.2 / Z2 / Stretch ===
      - Correct hop list returned by skiq_read_rx_freq_hop_list() and
        skiq_read_tx_freq_hop_list()

  === Sidekiq X4 ===
      - If a transmit test tone is enabled, restore its state after a sample
```

(continues on next page)

(continued from previous page)

```
       rate reconfiguration
     - Revert some poll intervals values (see Features v4.13.0) since it can
       adversely affect radio tuning and cause critical errors
     - Disable receive LNAs during receive calibration to provide maximal
       isolation
     - Fix misconfiguration of pre-select filters on Sidekiq X4 rev C
```

10.2.14 v4.13.0 - 06/30/2020

```
New Features for libsidekiq/FPGA
--------------------------------
o New Features for v4.13.0

  === Sidekiq ===
    - Add API call to initialize libsidekiq without specifying any cards --
      skiq_init_without_cards()
    - Logging from AD9361 driver routed through libsidekiq logging facilities

  === Sidekiq X2 / X4 ===
    - Add API functions to read/write/run RX calibration procedures manually
      or automatically
      - New functions include: skiq_read_rx_cal_mode, skiq_write_rx_cal_mode,
        skiq_run_rx_cal, skiq_read_rx_cal_type_mask,
        skiq_write_rx_cal_type_mask, skiq_read_rx_cal_types_avail

  === Sidekiq X4 ===
    - Improve standard and frequency hop tune times by lowering poll intervals
    - Add support for dual independently tunable RF transmit (A1 and B1)
    - Add support for AMS 3U VPX WB3XBM platform
    - Add support for importing RFIC profile from ADI's Profile Wizard

  === dmadriver ===
    - Add support for larger DMA ring buffer for Rx operation when using PCIe
      transport
    - Add support for Linux v5.4 kernels


Bug Fixes for libsidekiq/FPGA
-----------------------------
o Bug Fixes for v4.13.0

  === Sidekiq ===
    - Fix known issue: Process forking and skiq_receive()
      - With FPGA bitstream v3.13.1 and later along with libsidekiq v4.13.0 and
        later, stale DMA data is no longer an issue.
    - Fix skiq_save_fpga_config_to_flash() to reset metadata for config slot #0

  === Sidekiq Z2 ===
    - Fix filter setting after calling skiq_disable_cards / skiq_enable_cards
    - Fix known issue: FPGA reprogramming preserves transmit capability
      (requires updated Z2 BSP)

  === Sidekiq X4 ===
    - Address ADI-ERROR message(s) regarding "gain index exceeded expected
      maximum or minimum value..." when restoring RFIC configuration changing
```

(continues on next page)

(continued from previous page)

```
      sample rate
    - Fix ref_clock application so that verification no longer fails to execute
      when an external reference clock is configured
    - Extend calibration timeout from 8 seconds to 25 seconds as is recommended
      by ADI

  === sidekiq_gps ===
    - Fix known issue: GPIO values are retained and reapplied when reprogramming
      the FPGA
      - With dmadriver v5.4.0 and sidekiq_gps v1.0.1, the settings are retained
        through an FPGA reprogramming.
```

10.2.15 v4.12.2 - 04/06/2020

```
New Features for libsidekiq/FPGA
--------------------------------
o New Features for v4.12.2

  === No new features for v4.12.2 ===


Bug Fixes for libsidekiq/FPGA
-----------------------------
o Bug Fixes for v4.12.2

  === Sidekiq X4 ===
    - Fix issue where RX calibration was inadvertently skipped for LO
      frequencies around 4.2GHz

  === Sidekiq X2 ===
    - Fix issue where system timestamp frequency could be incorrect after
      loading a custom RFIC profile using skiq_prog_rfic_from_file

  === Sidekiq ===
    - Eliminate race condition in skiq_read_last_1pps_timestamp() where
      timestamps from different PPS periods could be returned
    - Fix issue where custom transport was inadvertantly skipped on
      aarch64.gcc6.3
    - Resolved incorrect behavior with skiq_rx_stream_mode_low_latency that
      could occur after FPGA re-programming
```

10.2.16 v4.12.1 - 02/21/2020

```
New Features for libsidekiq/FPGA
--------------------------------
o New Features for v4.12.1

  === No new features for v4.12.1 ===

Bug Fixes for libsidekiq/FPGA
-----------------------------
o Bug Fixes for v4.12.1
```

(continues on next page)

(continued from previous page)

```
  === Sidekiq Stretch ===
    - Restore reference clock selection and internal flash configuration after
      FPGA reprogramming

  === Sidekiq ===
    - Full reinitialization of RF front end configuration after FPGA
      reprogramming
```

10.2.17 v4.12.0 - 02/10/2020

```
New Features for libsidekiq/FPGA
--------------------------------
o New Features for v4.12.0

  === Sidekiq ===
    - Add support for storing and accessing multiple FPGA bitstreams in flash on
      supported devices.  Refer to the latest SDK manual under the "FPGA
      Configuration Flash Slots" section for more information.

  === Sidekiq mPCIe and M.2 ===
    - FPGA bitstream v3.13.0 for the USB transport doubles the FPGA's RX FIFO
      depth to help absorb delays by an overloaded host.

  === Sidekiq X2 ===
    - Although the Sidekiq X2 does not support frequency hopping, the libsidekiq
      API now allows a developer to use those functions with the X2.  The
      under-the-hood implementation uses the traditional tuning mode.  This
      allows developers to use a common tuning interface among products.

  === Sidekiq X4 ===
    - Support for 4 channel transmit at sample rates of 491.52Msps and 500Msps
      (400MHz or 450MHz RF BW) and transmit channels remain 4 channel phase
      coherent.  At this time, however, users may only transmit 4 channels using
      the FPGA's BRAM.  PCIe streaming is only supported for A1/A2.  BRAM
      playback is supported for A1/A2/B1/B2.  If users are interested in PCIe
      streaming to A1/B1, please contact Epiq Solutions through its support
      forums.


Bug Fixes for libsidekiq/FPGA
-----------------------------
o Bug Fixes for v4.12.0

  === Sidekiq Z2 ===
    - Restore FPGA reprogramming functionality that was inadvertently removed in
      libsidekiq v4.11.x

  === Sidekiq mPCIe + Dropkiq ===
    - Significant performance improvement in frequency tuning speed, requires
      FPGA bitstream v3.13.0 or later

  === Sidekiq Stretch ===
    - FPGA bitstream v3.13.0 defaults the PPS source to the
      `skiq_1pps_source_host`, the 1PPS signal from the on-board GPS module.
```

(continues on next page)

(continued from previous page)

```
  === Sidekiq X4 ===
    - TALISE_setRfPllFrequency() : Invalid rfpllLoFreq -- Fixed tuning failures
      at LO frequencies that should be permitted for a configured RxA1/A2/B1/B2
      RF bandwidth.
    - Frequency hop list is retained when sample rate configuration is changed
    - Restore 4 channel phase coherency functionality that was inadvertently
      removed in libsidekiq v4.11.1

  === dmadriver ===
      - Addressed incorrect usage of holding a spinlock across kernel calls
        that can sleep

  === sidekiq_uart ===
      - Addressed incorrect usage of mutex in timer callback
```

10.2.18 v4.11.1 - 11/22/2019

```
New Features for libsidekiq/FPGA
--------------------------------
o New Features for v4.11.1

  === Sidekiq X2 ===
    - Add support for the HTG-K800 FMC carrier with Xilinx Kintex Ultrascale
      KU115

  === Sidekiq X4 ===
    - Expand sample rate and bandwidth support on ORx handles (C1 and D1) to
      include 491.52Msps and 500Msps.
    - Add more sample rate and bandwidth RFIC profiles to match existing X2
      RFIC profiles (see Limited Sidekiq X4 Capabilities below for a full
      list)


Bug Fixes for libsidekiq/FPGA
-----------------------------
o Bug Fixes for v4.11.1

  === Sidekiq Stretch (M.2-2280) ===
    - Fix external 10MHz reference clock selection in ref_clock

  === Sidekiq X4 ===
    - Restore Tx test tone after transmit calibration
    - Fix frequency hop on timestamp
```

10.2.19 v4.11.0 - 10/17/2019

```
New Features for libsidekiq/FPGA
--------------------------------
o New Features for v4.11.0

  === Sidekiq Stretch (M.2-2280) ===
```

(continues on next page)

(continued from previous page)

```
    - Add product support
      - Receive, Transmit, and Transceive features fully functional
      - See "Limited Capabilities of Sidekiq Stretch" for what's missing
    - dmadriver v5.3.0.0 offers support for Stretch and its GPS/UART interface
      - skiq_platform_device is required to be loaded prior to dmadriver
      - sidekiq_uart and sidekiq_gps kernel modules are needed to receive NMEA
        sentences

  === Sidekiq X4 ===
    - Add support for the HTG-K800 FMC carrier with Xilinx Kintex Ultrascale
      KU115


Bug Fixes for libsidekiq/FPGA
-----------------------------
o Bug Fixes for v4.11.0

  === All ===
    - Perform range check for entries in passed frequency hop list to
      skiq_write_rx_freq_hop_list() and skiq_write_tx_freq_hop_list()

  === Sidekiq mPCIe / M.2 ===
    - Fix reading / writing gain mode selection -- skiq_read_rx_gain_mode()
      and skiq_write_rx_gain_mode()

  === Sidekiq Z2 ===
    - Fix reading / writing gain mode selection -- skiq_read_rx_gain_mode()
      and skiq_write_rx_gain_mode()
    - Fix transport defect where starting streaming, stopping streaming, or
      receiving sample blocks would result in an indefinite hang

  === Sidekiq X4 ===
    - Fix invalid Rx filter selection when frequency hopping
```

10.2.20 v4.10.1 - 08/16/2019

```
New Features for Libsidekiq/FPGA
--------------------------------
o New Features for v4.10.1

  === Sidekiq mPCIe / M.2 / Z2 ===
    - Add support for skiq_freq_tune_mode_hop_on_timestamp mode.
      NOTE: All frequencies in hopping list must fall within a
      single filter band.

  === Sidekiq X4 ===
    - Add support for skiq_freq_tune_mode_hop_on_timestamp mode.
      NOTE: If all frequencies do not fall within a single filter
      band, the bypass setting is used for all frequencies.  Users
      can override the filter setting at any point with the
      skiq_write_rx_preselect_filter_path() API.
    - Add RFIC profile for 50 Msps / 41 MHz channel bandwidth

Bug Fixes for Libsidekiq/FPGA
-----------------------------
```

(continues on next page)

(continued from previous page)

```
o Bug Fixes for v4.10.1

  === All ===
    - Add rx_samples_freq_hopping.c sample application to SDK
    - Fix missing Jetson Xavier driver directory for JetPack 4.1.1
    - Fix card index issue in rx_samples_on_trigger.c sample
      application

  === Sidekiq X4 ===
    - Improve JESD sync behavior to reduce number of necessary
      retries
```

10.2.21 v4.10.0 - 07/30/2019

```
New Features for Libsidekiq/FPGA
--------------------------------
o New Features for v4.10.0
  Includes all new features from v4.9.5 and earlier

  === All ===
    - Update tx_samples_async test application to use arg_parser
    - Add support for swapping order of I/Q components in both
      receive and transmit block formats
      - skiq_write_iq_order_mode() and skiq_read_iq_order_mode()
    - Extend access to more FPGA user app registers (banks 10 - 15)
    - Deprecate SRFS across all platforms (except on Matchstiq S10 /
      S20 series)
    - Add support for JetPack 4.2 for Jetson TX2 and Xavier

  === Sidekiq mPCIe / M.2 / Z2 ===
    - Add support for fast-lock profiles (receive)
      - skiq_write_rx_freq_tune_mode / skiq_read_rx_freq_tune_mode
      - skiq_write_rx_freq_hop_list / skiq_read_rx_freq_hop_list
      - skiq_write_next_rx_freq_hop / skiq_perform_rx_freq_hop
      - skiq_read_curr_rx_freq_hop / skiq_read_next_rx_freq_hop
      - NOTE: The `skiq_freq_tune_mode_hop_on_timestamp`
        skiq_freq_tune_mode_t functionality is not implemented as
        part of this release
    - Add support for fast-lock profiles (transmit)
      - skiq_write_tx_freq_tune_mode / skiq_read_tx_freq_tune_mode
      - skiq_write_tx_freq_hop_list / skiq_read_tx_freq_hop_list
      - skiq_write_next_tx_freq_hop  / skiq_perform_tx_freq_hop
      - skiq_read_curr_tx_freq_hop / skiq_read_next_tx_freq_hop
      - NOTE: The `skiq_freq_tune_mode_hop_on_timestamp`
        skiq_freq_tune_mode_t functionality is not implemented as
        part of this release
    - Improve RFIC register transaction performance

  === Sidekiq X2 ===
    - Add support for sample decimation in FPGA
      - Sample decimation is enabled automatically dependent on
        requested sample rate and channel bandwidth
      - See "Sidekiq X2 Capabilities" for more details

  === Sidekiq X4 ===
```

(continues on next page)

(continued from previous page)

```
    - Add support for sample decimation in FPGA
      - Sample decimation is enabled automatically dependent on
        requested sample rate and channel bandwidth
      - See "Sidekiq X4 Capabilities" for more details
    - Add support for fast frequency hopping (receive)
      - skiq_write_rx_freq_tune_mode / skiq_read_rx_freq_tune_mode
      - skiq_write_rx_freq_hop_list / skiq_read_rx_freq_hop_list
      - skiq_write_next_rx_freq_hop / skiq_perform_rx_freq_hop
      - skiq_read_curr_rx_freq_hop / skiq_read_next_rx_freq_hop
      - NOTE: The `skiq_freq_tune_mode_hop_on_timestamp`
        skiq_freq_tune_mode_t functionality is not implemented as
        part of this release
    - Add support for fast frequency hopping (transmit)
      - skiq_write_tx_freq_tune_mode / skiq_read_tx_freq_tune_mode
      - skiq_write_tx_freq_hop_list / skiq_read_tx_freq_hop_list
      - skiq_write_next_tx_freq_hop / skiq_perform_tx_freq_hop
      - skiq_read_curr_tx_freq_hop / skiq_read_next_tx_freq_hop
      - NOTE: The `skiq_freq_tune_mode_hop_on_timestamp`
        skiq_freq_tune_mode_t functionality is not implemented as
        part of this release
    - Configure single channel on single JESD lane when appropriate
    - Add support for 2-channel phase coherent transmit streaming
      over PCIe (handles A1 and A2)
    - Add test application to allow signal playback (transmit) from
      FPGA BRAM independent of transport (requires support in FPGA
      user app)
      - This new feature can also be used to showcase 4-channel
        phase coherent transmit

  === Windows ===
    - Add digitally signed Windows USB INF driver for Windows 10
      compatibility

Bug Fixes for Libsidekiq/FPGA
-----------------------------
o Bug Fixes for v4.10.0
  - Includes all bug fixes from v4.9.5 and earlier

  === Sidekiq mPCIe / M.2 ===
    - Fix FPGA transmit FIFO flushing behavior in timestamp mode and
      large block sizes
    - Fix missing clock configuration after
      skiq_prog_rfic_from_file()
    - Fix issue where AGC gets "stuck" until receiving a large
      (~20-30dB) delta in signal strength

  === Sidekiq Z2 ===
    - Fix missing clock configuration after
      skiq_prog_rfic_from_file()
    - Fix issue where AGC gets "stuck" until receiving a large
      (~20-30dB) delta in signal strength

  === Sidekiq X2 / X4 ===
    - Increase PCIe FIFO depth to deal with timestamp gaps at higher
      sample rates and multiple receive handles
```

10.2.22 v4.9.5 - 06/26/2019

```
New Features for Libsidekiq/FPGA
--------------------------------
o New Features for v4.9.5
  o Add support for arm_cortex-a9.gcc7.2.1_gnueabihf build
    configuration to support next Z2 BSP

  o Add support for FPGA programming on Sidekiq Z2 with the
    skiq_prog_fpga_from_file() API function
    o WARNING: Transmit functionality after using
      skiq_prog_fpga_from_file() on the Sidekiq Z2 is broken.  This
      remains an unresolved defect and will be addressed in future
      software releases.

Bug Fixes for Libsidekiq/FPGA
-----------------------------
o Bug Fixes for v4.9.5
  o Fix incorrectly reported transmit tone offset frequency when
    using Dropkiq at lower frequencies
  o Improve performance of the Z2 LO frequency retune operation
  o Resolved intermittent "MCS failed" errors on Sidekiq X2 / X4 by
    validating SYSREF pulse completion to the AD9528
  o Fix hardware probe false positives when attempting to detect
    Dropkiq
```

10.2.23 v4.9.4 - 05/03/2019

```
New Features for Libsidekiq/FPGA
--------------------------------
o New Features for v4.9.4
  o No new features

Bug Fixes for Libsidekiq/FPGA
-----------------------------
o Bug Fixes for v4.9.4
  o Fix "Quad SPI operation bit needed assertion" errors on the
    Sidekiq X2 PDK or Sidekiq X4 PDK
    o Affects skiq_save_fpga_config_to_flash and
      skiq_verify_fpga_config_from_flash
  o Resolved initialization issue for PCIe-only Sidekiq M.2
    configured with FPGA bitstreams prior to v3.5
  o Prohibit unsupported packed mode setting for Sidekiq X2 and
    Sidekiq X4
    o Affects skiq_write_iq_pack_mode
  o Fix 1PPS timeout approach to consider user specified system
    timestamp
    o Affects the following API functions:
      o skiq_start_tx_streaming_on_1pps /
        skiq_stop_tx_streaming_on_1pps
      o skiq_start_rx_streaming_on_1pps /
        skiq_stop_rx_streaming_on_1pps
      o skiq_start_rx_streaming_multi_on_trigger /
        skiq_stop_rx_streaming_multi_on_trigger
  o Fix MCS failure on second initialization of Sidekiq X2
```

10.2.24 v4.9.3 - 03/19/2019

```
New Features for Libsidekiq/FPGA
--------------------------------
o New Features for v4.9.3
  o No new features

Bug Fixes for Libsidekiq/FPGA
-----------------------------
o Bug Fixes for v4.9.3
  o Fix race condition in skiq_prog_from_file() / skiq_prog_from_flash() that
    could cause a deadlock across multiple applications controlling different
    Sidekiq cards
  o Correct DAC resolution provided by skiq_read_tx_iq_resolution() for
    Sidekiq X4, was 16 (incorrect), now 14 (correct)
```

10.2.25 v4.9.2 - 03/08/2019

```
New Features for Libsidekiq/FPGA
--------------------------------
o New Features for v4.9.2
  o No new features

Bug Fixes for Libsidekiq/FPGA
-----------------------------
o Bug Fixes for v4.9.2
  o Prevent Sidekiq X4 from tuning the LO frequency to problematic values
  o Fix M.2 dual channel transmit FIFO issue in FPGA design for M.2
  o Fix "last TX timestamp" (affects API function skiq_read_last_tx_timestamp)
    issue in FPGA design for mPCIe and M.2
  o Update tx_configure, tx_samples, and tx_samples_async test applications to
    set some of the "other" handle parameters when configured for dual channel
    mode
```

10.2.26 v4.9.1 - 02/26/2019

```
New Features for Libsidekiq/FPGA
--------------------------------
o New Features for v4.9.1
  o Added 250Msps Sample Rate / 200 MHz Channel Bandwidth profile for Sidekiq X4
  o Added a timeout when starting/stopping streaming upon 1PPS if 1PPS never occurs

Bug Fixes for Libsidekiq/FPGA
-----------------------------
o Bug Fixes for v4.9.1
  o Resolved issue with segfault observed with certain custom profiles for Sidekiq X2
  o Resolved issue where register verification could fail at lower sample rates
  o Implement retrying if configuring FPGA from flash fails (for Sidekiq M.2 / mPCIe)
  o Revolved data streaming issues for Sidekiq X4 when using sample rates of
    245.76Msps
  o Reduced LO tuning time for Sidekiq X4
  o Improved RX calibration settings for Sidekiq X4
```

10.2.27 v4.9.0 - 02/06/2019

```
New Features for Libsidekiq/FPGA
--------------------------------
o New Features for v4.9.0
  o Added ability to query any actively streaming RX handles
    o skiq_read_rx_streaming_handles()
  o Added ability to query conflicting RX stream handles
    o skiq_read_rx_stream_handle_conflict()
  o Added ability to start streaming synchronously
    o skiq_start_rx_streaming_multi_synced()
    o skiq_stop_rx_streaming_multi_synced()
  o Added ability to configure TX test tone frequency (not available with all
    products)
    o skiq_read_tx_tone_freq_offset()
    o skiq_write_tx_tone_freq_offset()
  o Added ability to perform verification of user FPGA register after writing
    o skiq_write_and_verify_user_fpga_reg()
  o Added RF IC control output support for X2/X4
    o skiq_read_rfic_control_output_rx_gain_config()
    o skiq_enable_rfic_control_output_rx_gain()
  o Added support for X4 revision B
  o Added support for X4 RX C1 / D1 handles
  o Added support for X4 transmit
  o Refer to X4 limited capabilities for details on TX limitations
  o Added support for 245.76Msps for Sidekiq X2 RX B1
o Includes features from v4.8.0
  o Ability to enable/disable cards independently without reinitializing the library
    o skiq_enable_cards()
    o skiq_disable_cards()
o Added ability for users to create and load their own RF profiles for
  Sidekiq X2 generated by Analog Devices' Filter Wizard (refer to the Sidekiq
  Software Developer's Manual for additional details)
    o skiq_prog_rfic_from_file()
o Added support for blocking receive when using a USB transport

Bug Fixes for Libsidekiq/FPGA
-----------------------------
o Resolved issue with asynchronous transmit occasionally not completing
o Reduced tune times for Sidekiq X2
o Improved Windows performance for all Sidekiq products
o Improved RX streaming performance for Sidekiq Z2
o Added new return code to skiq_receive() in the case where data is requested
  from a card not currently streaming
  o skiq_rx_status_error_card_not_active
o Improved error reporting and handling
```

10.2.28 v4.7.1 - 10/15/2018

```
New Features for Libsidekiq/FPGA
--------------------------------
o None

Bug Fixes for Libsidekiq/FPGA
-----------------------------
```

(continues on next page)

(continued from previous page)

```
o Decreased complete / run-time installer sizes
o Add -std=gnu11 to CFLAGS in SDK Makefiles
o Resolved known JESD sync transmit issues on Sidekiq X2
o Resolved intermittent transmit timestamp hang on Sidekiq X2
o Resolved storage of Dropkiq factory calibration on Matchstiq S12
```

10.2.29 v4.7.0 - 09/24/2018

```
New Features for Libsidekiq/FPGA
--------------------------------
o Sidekiq X2 rev C hardware support added
o Sidekiq Z2 rev C hardware support added
o Preliminary X4 rev A support added
o Balanced RX stream mode for Z2
o Transmit support for Z2
o Support for host reference clock for X2 (for use with Epiq's GPS module)
o Support for host 1PPS source for X2 (for use with Epiq's GPS module)
o X2 calibration support added
o Storage/query of calibration date added
o Added support for compiling for a specific target platform (Sidekiq Z2) for
  a reduced application and library size
o Added support for warpable TCVCXO for Sidekiq X2 rev C

Bug Fixes for Libsidekiq/FPGA
-----------------------------
o Improved reliability of flash reprogramming of X2 FPGA bitstream
o Resolved X2 FPGA reprogramming for larger bitstreams
o Resolved system timestamp frequency incorrectly reported after sample rate
  configuration of the X2
o Resolved TX test tone frequency incorrectly reported with X2
o Resolved FPGA reprogramming inconsistencies when performed across multiple processes
o Resolved TX attenuation configuration reset after retune for X2
o Added full RF IC reset functionality for m.2 / mPCIe / z2 upon initialization
o Resolved issue with incorrect number of taps reported and used for custom FIR
  coefficients for Sidekiq mPCIe / m.2 / Z2
o Significant performance improvement with Sidekiq Z2 RX stop streaming
o Significant reduction in Sidekiq Z2 FPGA resource utilization
o Fixed incorrect RF port mapping for Sidekiq X2 Rx A1 / Rx B1
```

10.2.30 v4.6.0 - 06/15/2018

```
New Features for Libsidekiq/FPGA
------------------------
o Increased maximum available Sidekiq cards supported to 32
o Sidekiq Z2 rev B hardware supported
o Added RF port configuration and querying ability
o Added Sidekiq m.2 USB streaming capability
o Added support for a list of receive handles to be provided to start receive
  streaming
o Added new transmit mode to transmit a packet in timestamp mode if the
  timestamp is late.
o Added Windows 10 support
```

(continues on next page)

(continued from previous page)

```
o Added ability for user to enable or disable TX quadrature calibration.
o Added new RX streaming mode to allow for shorter packets to be transferred,
  thus reducing the transport latency.

Bug Fixes for Libsidekiq/FPGA
-----------------------
o Unaligned memory access resolved in ARM platforms
o Resolved issues with custom FIR coefficients overwritten during TX retune
o Updated numerous test applications to allow for selection of Sidekiq based on serial
  number in addition to card number.
o Improved detection of the Dropkiq low frequency extender card for use with
  Sidekiq mPCIe.
o Resolved invalid reporting of a late timestamp in the case where transmit
  FIFO was empty.
o Improved TX quadrature error calibration for Sidekiq X2.
```

10.2.31 v4.4.0 - 11/02/2017

```
New Features for Libsidekiq/FPGA
-----------------------
o Windows 7 support now available, including signed device driver
o Additional sample rates supported for Sidekiq X2
o Preliminary Sidekiq Z2 support
o Support for RX attenuation mode configuration for Sidekiq X2. New APIs added
  to support this are: skiq_write_rx_attenuation(), skiq_read_rx_attenuation(),
  skiq_read_rx_attenuation_mode(), skiq_write_rx_attenuation_mode()
o The ability to query various parameters of the radio is now available via
  skiq_read_parameters()
o Management of FPGA versions now support semantic versioning and can be queried
  via skiq_read_fpga_semantic_version()
o RX calibration for Sidekiq mPCIe and m.2 are maintained within the flash part
  of the card. To determine if stored calibration data is present, the
  skiq_read_rx_cal_data_present() API is available, otherwise default
  calibration data is used
o Automatic detection of transport layer now supported. The skiq_init() API can
  take the skiq_xport_type_auto and the transport is automatically detected and
  selected

Bug Fixes for Libsidekiq/FPGA
-----------------------
o Erratic system timestamps fixed in FPGA v3.8.0
o Fixed transmit PPS synchronization error
o Improved oscillator drift of Sidekiq X2
o Improved initialization time of Sidekiq X2
o Resolved issues with transmitter filter configuration for Sidekiq mPCIe and m.2
o Resolved incorrect setting of TX attenuation for Sidekiq X2
```

10.2.32 v4.2.1 - 11/02/2017

```
New Features for Libsidekiq/FPGA
-----------------------
No new features from v4.2.0 have been added.
```

(continues on next page)

(continued from previous page)

```
Bug Fixes for Libsidekiq/FPGA
-------------------------
o RX FIR filter coefficients below sample rates of 13.3 Msps have been
  updated. In v4.2.0, the RX FIR coefficients resulted in a 6dB increase when
  operating at sample rates lower than 13.3 Msps vs higher than 13.3 Msps.
o The minimum and maximum variable names of skiq_read_rx_gain_index_range()
  have been modified to match actual implementation.
```

10.2.33 v4.2.0 - 09/29/2017

```
New Features for Libsidekiq/FPGA
-------------------------
The following functions were added for v4.2.0. For full details of the
new functions, refer to sidekiq_api.h.

o skiq_read_tx_tone_freq()
  - allows for the frequency of the TX test tone to be determined
o skiq_is_accel_supported()
  - determines if the accelerometer is supported based on the Sidekiq type detected
o skiq_write_accel_reg()
  - provides generic write register access to access to the accelerometer
o skiq_read_accel_reg()
  - provides generic read register access to access to the accelerometer
o skiq_read_part_info()
  - provides part information regarding Sidekiq detected
o skiq_read_max_sample_rate()
  - allows for the maximum supported sample rate for the card to be determined
o skiq_read_min_sample_rate()
  - allows for the minimum supported sample rate for the card to be determined
o skiq_read_rx_iq_resolution()
  - allows for the RX IQ resolution for the card to be determined
o skiq_read_tx_iq_resolution()
  - allows for the TX IQ resolution for the card to be determined
o skiq_read_rx_filters_avail()
  - provides an array of RX filters available on the Sidekiq card
o skiq_read_tx_filters_avail()
  - provides an array of TX filters available on the Sidekiq card
o skiq_read_filter_range()
  - provides the minimum and maximum frequency range of the filter specified
o skiq_read_usb_enumeration_delay()
  - returns the delay of the Sidekiq USB enumeration on the USB bus
o skiq_read_sys_timestamp_freq()
  - returns the frequency at which the system timestamp increments for the
    Sidekiq card
o skiq_read_ext_ref_clock_freq()
  - returns the frequency at which the external reference clock operates

Bug Fixes for Libsidekiq/FPGA
-------------------------
o Race condition causing rare lockup in transmit shutdown resolved

o Sidekiq M.2 FPGA flash fallback image now supported (refer to section 14.3)
o Sidekiq X2 hardware revision B support added
o Sidekiq mPCIe / M.2 transmit quadrature / DC offset calibration improved
```

(continues on next page)

(continued from previous page)

```
o Sidekiq mPCIe / M.2 FIR filter coefficients updated for sample rates below 13.3MHz
o Sidekiq mPCIe / M.2 phase inversion of A2 resolved
```

10.2.34 v4.0.1 - 07/18/2017

```
New Features for Libsidekiq/FPGA
--------------------------------
No functions were added for v4.0.1

Bug Fixes for Libsidekiq/FPGA
-----------------------------
o Fix USB probing issuing that ended up limiting USB to 3 instances instead of 4
```

10.2.35 v4.0.0 - 05/15/2017

```
New Features for Libsidekiq/FPGA
------------------------
The following functions were added for v4.0.0. For full details of the
new functions, refer to sidekiq_api.h.

o skiq_get_cards()
  - provides a list of cards based on the transport type specified
o skiq_init_by_serial_str()
  - identical to skiq_init() except serial numbers can be provided instead
    of card numbers
o skiq_is_xport_avail()
  - determines if a transport type is available for the specified card
o skiq_is_card_avail()
  - determines if the card is available for use
o skiq_verify_fpga_config_from_flash()
  - verifies the contents of the FPGA bitstream specified matches the
    FPGA bitstream stored in flash on the Sidekiq
o skiq_read_rx_cal_offset()
  - provides calibration offset
o skiq_read_rx_cal_offset_by_LO_freq()
  - provides calibration offset based on frequency specified
o skiq_read_rx_cal_offset_by_gain_index()
  - provides calibration offset based on gain specified
o skiq_read_rx_cal_offset_by_LO_freq_and_gain_index()
  - provides calibration offset based on gain and frequency specified
o skiq_read_last_tx_timestamp()
  - provides last timestamp seen by the FPGA when transmitting
o skiq_set_tx_block_timestamp(), skiq_tx_get_block_timestamp()
  - convenience functions to access timestamps in TX block
o skiq_tx_block_allocate_by_bytes(), skiq_tx_block_allocate(), skiq_tx_block_free()
  - convenience functions to allocate / free skiq_tx_block_t
o SKIQ_TX_BLOCK_INITIALIZER_BY_BYTES(), SKIQ_TX_BLOCK_INITIALIZER_BY_WORDS(),
  SKIQ_TX_BLOCK_INITIALIZER()
  - convenience MACROs to statically allocate skiq_tx_block_t
o SKIQ_RX_BLOCK_INITIALIZER_BY_BYTES(), SKIQ_RX_BLOCK_INITIALIZER_BY_WORDS(),
  SKIQ_RX_BLOCK_INITIALIZER()
  - convenience MACROs to statically allocate skiq_rx_block_t
```

(continues on next page)

(continued from previous page)

```
Bug Fixes for Libsidekiq/FPGA
-----------------------
o Critical race condition causing EEPROM corruption in applications that use
  multiple cards has been resolved
o Critical race condition in reconfiguring the FPGA on certain host systems resolved.
o Flash access speeds greatly improved
o Parallel reconfiguring of FPGA resolved
o Card access by multiple applications protection added
o System update fully supports systems with multiple cards
o USB RX streaming for m.2 now supported

Libsidekiq APIs Modified
-----------------------
o skiq_init()
  Details:
    The function has been updated to support the transport type and
    initialization level selected as separate parameters of the
    initialization. Additionally, if a USB interface is available, USB only
    features are automatically available regardless of the transport type
    specified (ex. programming the FPGA from file).

  Migration Path:
    Examples of how to transition from the skiq_init() function of v3.X.Y to
    v4.0.0 are shown below.

    o skiq_init(skiq_pcie_init_level_1, skiq_usb_init_level_0, p_cards, num_cards ) =>
      skiq_init(skiq_xport_type_pcie, skiq_xport_init_level_basic, p_cards, num_cards )

    o skiq_init(skiq_pcie_init_level_2, skiq_usb_init_level_0, p_cards, num_cards ) =>
      skiq_init(skiq_xport_type_pcie, skiq_xport_init_level_full, p_cards, num_cards )

    o skiq_init(skiq_pcie_init_level_2, skiq_usb_init_level_1, p_cards, num_cards ) =>
      skiq_init(skiq_xport_type_pcie, skiq_xport_init_level_full, p_cards, num_cards )

    o skiq_init(skiq_pcie_init_level_0, skiq_usb_init_level_1, p_cards, num_cards ) =>
      skiq_init(skiq_xport_type_usb, skiq_xport_init_level_basic, p_cards, num_cards )

    o skiq_init(skiq_pcie_init_level_0, skiq_usb_init_level_2, p_cards, num_cards ) =>
      skiq_init(skiq_xport_type_usb, skiq_xport_init_level_full, p_cards, num_cards )

o skiq_register_tx_complete_callback()
  Details:
    The function has been updated to use the new skiq_tx_callback_t type. This
    allows the transmit complete callback function to be provided both the
    transmitted packet (skiq_tx_block_t) as well
    as user data specified during the transmit call.

  Migration Path:
    Examples of how to transition from the
    skiq_register_tx_complete_callback() function of v3.X.Y to v4.0.0 is shown
    below.

      o skiq_register_tx_complete_callback( card, &tx_complete )
        o void tx_complete( int32_t status, uint32_t *p_data ) =>
          void tx_complete( int32_t status, skiq_tx_block_t *p_data , void *p_user )
```

(continues on next page)

(continued from previous page)

```
 o skiq_read_libsidekiq_version()
   Details:
     A new parameter, p_label has been added.

   Migration Path:
     Examples of how to transition from the skiq_read_libsidekiq_version()
     function of v3.X.Y to v4.0.0 is shown below.

     o skiq_read_libsidekiq_version( &major, &minor, &patch ) =>
       skiq_read_libsidekiq_version( &major, &minor, &patch, & label );

o skiq_register_critical_error_callback()
  Details:
    The critical error callback function that can be registered has been
    updated to also allow custom user data to be referenced.

  Migration Path:
    Examples of how to transition from the
    skiq_register_critical_error_callback() function of v3.X.Y to v4.0.0 is
    shown below.

    o void skiq_register_critical_error_callback( void (*critical_handler)(int32_t status) ) =>
      void skiq_register_critical_error_callback( void (*critical_handler)(int32_t status, void* p_user_data ),␣
↪void* p_user_data );

o skiq_transmit()
  Details:
    The transmit function has been updated to use the newly added
    skiq_tx_block_t type. Additionally, custom private data can be provided on
    a per packet basis. This private data is then provided to the registered
    TX complete callback function when running in asynchronous mode.

  Migration Path:
    Examples of how to transition from the skiq_transmit() function of v3.X.Y
    to v4.0.0 is shown below.

      o int32_t skiq_transmit(uint8_t card, skiq_tx_hdl_t hdl, int32_t *p_samples); =>
        int32_t skiq_transmit(uint8_t card, skiq_tx_hdl_t hdl, skiq_tx_block_t *p_block , void *p_user );

o skiq_receive()
  Details:
    The receive function has been updated to use the newly added skiq_rx_block_t type.

  Migration Path:
    Examples of how to transition from the skiq_receive() function of v3.X.Y
    to v4.0.0 is shown below.

      o skiq_rx_status_t skiq_receive(uint8_t card, skiq_rx_hdl_t* p_hdl, uint8_t** pp_data, uint32_t* p_data_
↪len); =>
        skiq_rx_status_t skiq_receive(uint8_t card, skiq_rx_hdl_t* p_hdl, skiq_rx_block_t** pp_block, uint32_t*␣
↪p_data_len);

Libsidekiq Functions Deprecated (v3.X.Y => v4.0.0)
-------------------------
o skiq_probe()
```

(continues on next page)

(continued from previous page)

```
  Details:
    This function is no longer needed as all necessary hardware probing handles
    automatically. Hotplugging hardware is not currently supported. As a result
    As a result, all Sidekiq hardware must be present prior to running any Sidekiq
    application.
  Migration Path:
    This function is no longer needed and does not have a replacement.

o skiq_probe_xport()
  Details:
    This function is no longer needed as all necessary hardware probing handles
    automatically. Hotplugging hardware is not currently supported. As a result
    all Sidekiq hardware must be present prior to running any Sidekiq application.
  Migration Path:
   This function is no longer needed and does not have a replacement.

o skiq_get_avail_cards()
  Details:
    This function has been replaced by skiq_get_cards(), which returns a list of
    all card numbers detected on the specified transport interface
  Migration Path:
    Replace skiq_get_avail_cards() with skiq_get_cards().

o skiq_get_serial_num()
  Details:
    This function was only supported with miniPCIe serial number formats.
  Migration Path:
    Replace skiq_get_serial_num() with skiq_read_serial_string().

o skiq_get_card_num()
  Details:
    This function was only supported with miniPCIe serial number formats.
  Migration Path:
    Replace skiq_get_card_num() with skiq_get_card_from_serial_string().

o skiq_read_serial_num()
  Details:
    This function was only supported with miniPCIe serial number formats.
  Migration Path:
    Replace skiq_read_serial_num() with skiq_read_serial_string().

o skiq_init_xport()
  Details:
    This function is no longer needed. Initialization of a custom transport is
    supported by specifying skiq_xport_type_custom in skiq_init().
  Migration Path:
    Replace skiq_init_xport() with skiq_init() and specify skiq_xport_type_custom.

o skiq_read_rfic_temp()
  Details:
    The RF IC temperature correlates to the Sidekiq board temperature
    (available with skiq_read_temp()). Thus, the function was removed
    as it did not provide any additional temperature information.
  Migration Path:
    This function is no longer needed and does not have a replacement.
```