Zynq UltraScale+ RFSoC RF Data Converter Evaluation Tool (ZCU111)

User Guide

UG1287 (v2019.1) May 29, 2019

XILINX

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
05/29/2019 Version 2019.1	
General updates.	Updated for Vivado Design Suite version 2019.1.
12/05/2018 Version 2018.3	
Programmable Logic Filter	Added this section.
Chapter 2, Package Details	Updated the evaluation tool package name for Vivado Design Suite version 2018.3.
11/06/2018 Version 2018.2	
Chapter 2, Package Details	Updated design package link.
10/01/2018 Version 2018.2	
DAC Data Flow	Added information about feeding data to the RF-DAC.
Streaming MUX	Added channel control selection information.
GPIO Selection	Replaced Table 3-2.
Application Flow	Added DDR and BRAM selection information and information to start DMA.
DAC Flow for PL DDR	Added this section.
Example Commands and Responses	Added commands to Table A-1.
08/14/2018 Version 2018.2	
Initial Xilinx release.	N/A

Table of Contents

Introduction

Overview

The objective of this reference design is to help you quickly and easily evaluate the new RF Data Converter (DC) Evaluation Tool functionality in the Zynq® UltraScale+™ family of RFSoCs. The RFSoC design demonstrates the capabilities and performance of the RF data converter (RFDC—RF-ADC and RF-DAC) available in the RFSoC devices. The evaluation tool serves as a platform for you to evaluate the Zynq UltraScale+ RFSoC features and helps accelerate the product design cycle.

The evaluation tool consists of a ZCU111 evaluation board and a custom-developed graphical user interface (GUI) installed on a Windows host machine. The evaluation tool allows you to configure the operation of the RF-ADCs and RF-DACs and perform some basic tests e.g., FFT analysis of the ADC output for various input test signals. The key differentiator of Zynq UltraScale+ RFSoC devices when compared to many other discrete solutions is that the device contains both RF-ADCs and RF-DACs. However, one significant benefit is the DACs can be used to provide test signals for the ADC (i.e., loopback) which facilitates a very compact and easy to use solution for early demonstration or evaluation.

All communications to the host PC (GUI) use the processing system (PS) Ethernet interface. This is necessary to facilitate the transfer of a large amount of test data as efficiently as possible. The ZCU111 evaluation board supports an external DDR4 memory interface on the programmable logic (PL) in addition to the PS DDR4 memory. Waveforms with a limited number of samples can leverage on-chip memory, but application testing and prototyping require the use of much larger external memories.

The Xilinx® Vivado® IP integrator flow is used to create the hardware design, which is partitioned between the PS, RFDC, and PL. The reference design uses the IP integrator core for the RF Data Converter subsystem. The implementation supports all data rates on PL to the Data Converter interface, all converter sample rates and digital up conversion (DUC)/digital down conversion (DDC) configurations with a single design. The Xilinx PetaLinux flow is used to create and integrate the software components, including the Linux kernel and drivers.

This user guide describes the architecture of the design and provides a functional description of its components. It is organized as follows:

Chapter 1, Introduction (this chapter) provides a high-level overview of the Zynq UltraScale+ RFSoC architecture, the design architecture, and a summary of key features.

Chapter 2, Package Details gives an overview of the design modules and design components that make up this design.

Chapter 3, Hardware Design describes the hardware platform of the design including key PS and PL peripherals.

Chapter 4, Clocking describes the details on clocking used for the design.

Chapter 5, Evaluation Tool System Configuration using the GUI describes the details of system configuration and features supported using the GUI.

Chapter 6, Software Architecture describes the application processor unit (APU) software platform including the Linux software stack and the Linux rftool application running on the APU.

Chapter 7, Protocol Specification describes the protocol used to communicate between the host and RFSoC.

Chapter 8, Zynq UltraScale+ RFSoC Data Converter Bare-metal/Linux Driver describes where to get more information about the driver.

Chapter 9, System Considerations describes system architecture considerations including boot flow and the system address map.

Appendix A, Reference Design Protocol Specification describes the commands used in the software design.

Appendix B, Additional Resources and Legal Notices lists additional resources and references.

Zynq UltraScale+ RFSoC Overview

The Zynq UltraScale+ RFSoC family integrates the key subsystems required to implement a complete software-defined radio including direct RF sampling data converters, enabling CPRI and Gigabit Ethernet-to-RF on a single, highly programmable SoC.

Each RFSoC offers multiple RF-sampling analog-to-digital (RF-ADC) and RF-sampling digital-to-analog (RF-DAC) data converters. The RF-ADC supports a maximum sample rate of 4 GSPS with dynamic range and has a signal bandwidth of up to 4 GHz. The RF-DAC can clock at up to 6.554 GSPS with an output signal bandwidth of greater than 4 GHz. The RF data converters also include power efficient digital down converters (DDCs) and digital up converters (DUCs) that include programmable interpolation, decimation rates, a numerically controlled oscillator (NCO), and a complex mixer. The DDCs and DUCs can also support multi-band operation. Figure 1-1 shows the block diagram of the Zynq UltraScale+ RFSoC RF Data Converter.

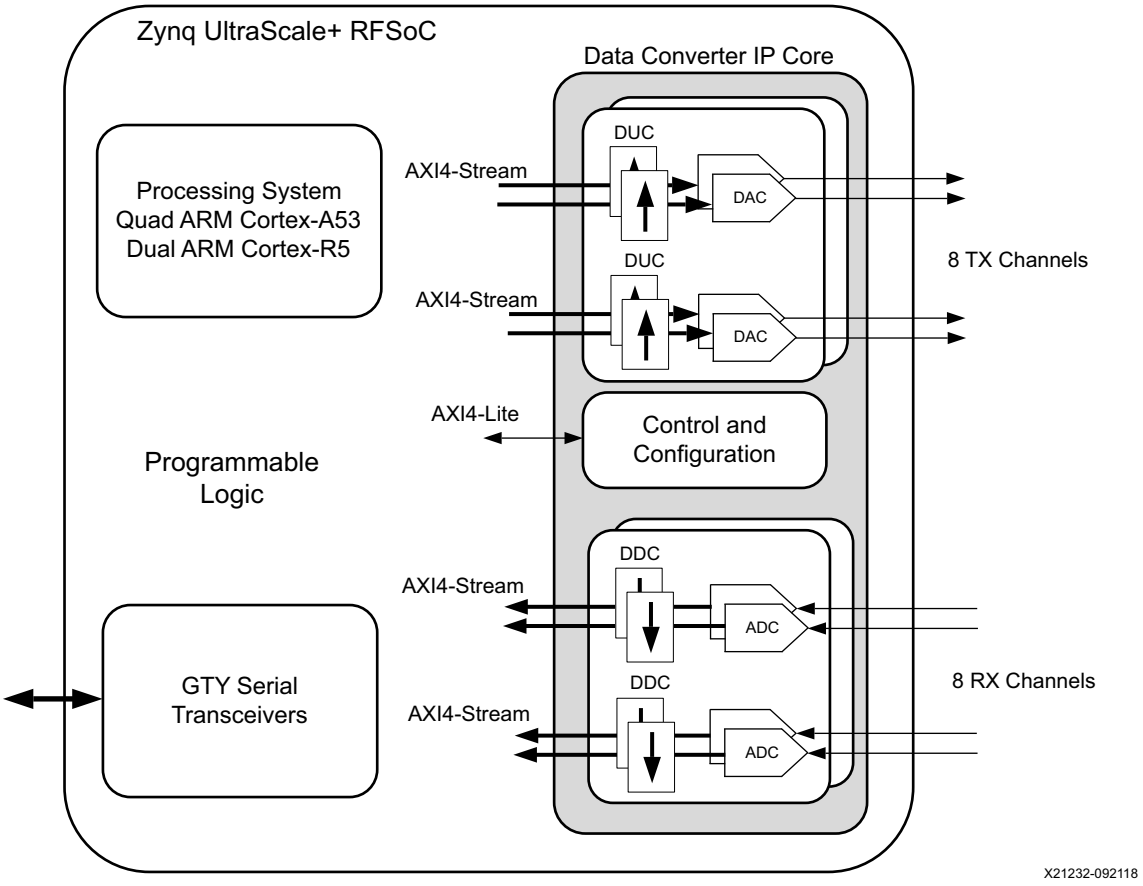


Figure 1-1: **Zynq UltraScale+ RFSoC RF Data Converter in RFSoC**

The RF-ADCs and RF-DACs are organized into tiles, each containing either two or four RF-ADCs or four RF-DACs (see Figure 1-2). Each tile also includes a block with a PLL and all the necessary clock handling logic and distribution routing for the analog and digital logic.

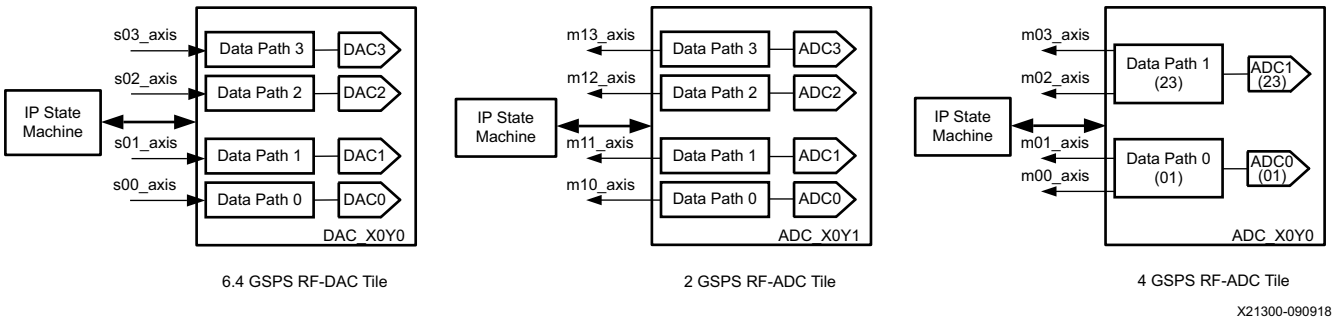


Figure 1-2: **Converter Tile Structure**

For device specifications and additional information, see:

- *Zynq UltraScale+ RFSoC Data Sheet: Overview* (DS889) [Ref 1]
- *Zynq UltraScale+ RFSoC Data Sheet: DC and AC Switching Characteristics* (DS926) [Ref 2]
- *Zynq UltraScale+ Device Technical Reference Manual* (UG1085) [Ref 3].

Reference Design Overview

The evaluation tool targets the Zynq UltraScale+ RFSoC ZU28DR-FFVG1517 running on the ZCU111 evaluation board and provides a platform to evaluate the RFSoC features. The system level block diagram of the evaluation tool design is shown in Figure 1-3.

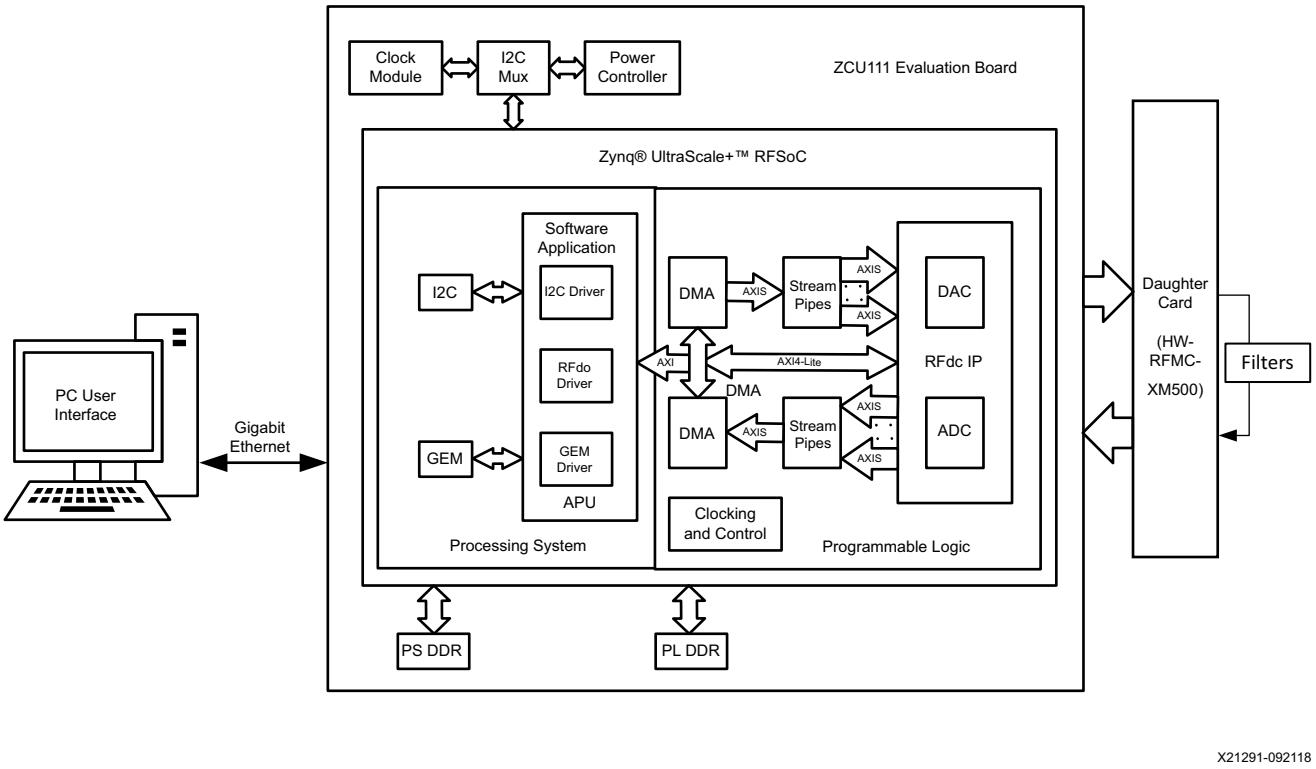


Figure 1-3: **RF Data Converter Evaluation Tool System Level Block Diagram**

The evaluation tool uses an integrated RF Data Converter in an 8x8 configuration along with AXI DMA and AXI4-Stream components for high performance data transfers between PL-DDR to RFDC and vice versa. Stream Pipe comprises various AXI4-Stream Infrastructure IPs. The AXI DMA is configured in Scatter Gather (SG) mode for high performance. The evaluation tool also makes use of multiple processing units available inside the PS, such as Gigabit Ethernet, I2C, and SD Interface. The APU inside the PS is configured to run in symmetric multiprocessing (SMP) Linux mode. The main task of the Linux application is to configure and control the RF-ADC and RF-DAC blocks and the flow of data through the streaming pipeline.

A custom-developed Windows-based GUI is provided along with this evaluation tool. It can interact with the RFSoC running on the ZCU111 evaluation board. The GUI connects to the Linux application running on the RFSoC via a TCP Ethernet interface. Based on commands

received from the GUI, the Linux application performs various operations that are described in Chapter 6, Software Architecture. Because a TCP socket is used to transfer the data over Ethernet, it is possible to run the GUI on any machine connected to the network (i.e., cloud).

The evaluation tool can be run in three separate modes:

Standalone DAC: In this mode, a pattern is generated using the GUI on the host machine. This pattern is constantly replayed on the selected DAC channel. The output of the DAC can be monitored on any standard external equipment, such as a spectrum analyzer or oscilloscope.

Standalone ADC: In this mode, you generate an analog signal from external equipment, and this signal is fed to ADC inputs. Digital output of the ADC can be analyzed on the host machine using the GUI.

DAC to ADC Loopback: In this mode, the output of the DAC is looped back to the input of ADC. In this way, you can generate a pattern on DAC from the GUI and can analyze the same pattern on ADC output in the GUI. It is recommended to manage RF signal conditioning well. Anti-aliasing filters are supplied in the ZCU111 development kit for the existing RF line up of RFMC-XM500.

Note: For system performance and limitations in various scenarios, see the *ZCU111 RFSoC RF Data Converter Evaluation Tool Getting Started Guide* [Ref 4].

Components

- Evaluation platform
  - ZCU111 evaluation board
  - Daughter card (HW-FMC-XM500)
  - Cables and filters (see the *Zynq UltraScale+ RFSoC ZCU111 Evaluation Kit Quick Start Guide* (XTP490) [Ref 5]
- Xilinx tools
  - Vivado® Design Suite 2019.1 [Ref 6]
  - Xilinx® Software Development Kit (XSDK) 2019.1 [Ref 7]
  - PetaLinux tools 2019.1 [Ref 8]
- Hardware interfaces and IP
  - RFDC
  - AXI DMA
  - DDR controller interface
  - AXI SmartConnect
  - AXI interconnect

  - AXI4-Stream IPs
- Auxiliary peripherals
  - SD
  - I2C
  - PS-GPIO
  - Gigabit Ethernet
  - UART
  - JTAG

Software Components

- Operating systems
  - APU: SMP Linux
- Linux frameworks
  - Ethernet
  - Clock
  - Contiguous memory allocator (CMA)
- User applications
  - APU: Ethernet-based server application
- RFDC driver-based features
  - Digital down conversion (DDC)/digital up conversion (DUC)
  - Nyquist Mix mode
  - Digital complex mixers
  - Quadrature modulation correction (QMC)
  - Internal PLL
  - ADC calibration
  - DAC high linearity and low noise
  - Inverse sinc filter
  - DAC Output Current mode (20 or 32 mA)
  - External clock driver
- AXI-DMA client driver
- Board-specific components
  - Voltage controller

Package Details

The evaluation tool ZIP file package **rdf0476-zcu111-rf-dc-eval-tool-2019-1.zip** contains the following components grouped by application processor unit (APU) or programmable logic (PL). The package is available at the Zynq UltraScale+ RFSoC ZCU111 Evaluation Kit documentation website.

APU

petalinux_bsp: PetaLinux board support package (BSP) is included to build a pre-configured SMP Linux image for the APU. The BSP includes the following components:

- First stage boot loader (FSBL)
- ARM trusted firmware (ATF)
- U-Boot
- Linux kernel
- Device tree
- Root file system (rootfs)

PL

Vivado: Vivado IP integrator design that integrates the RF Data Converter subsystem, AXI DMA, Stream Pipe, AXI Interconnect, and PL DDR controller.

Host System GUI

The user interface connecting to the ZCU111 platform via Ethernet cable.

Hardware Design

Hardware Overview

The Vivado IP integrator flow is used to create the hardware design which is partitioned between the processing system (PS), RF Data Converter (RFDC), and programmable logic (PL). Figure 3-1 shows the hardware block diagram.

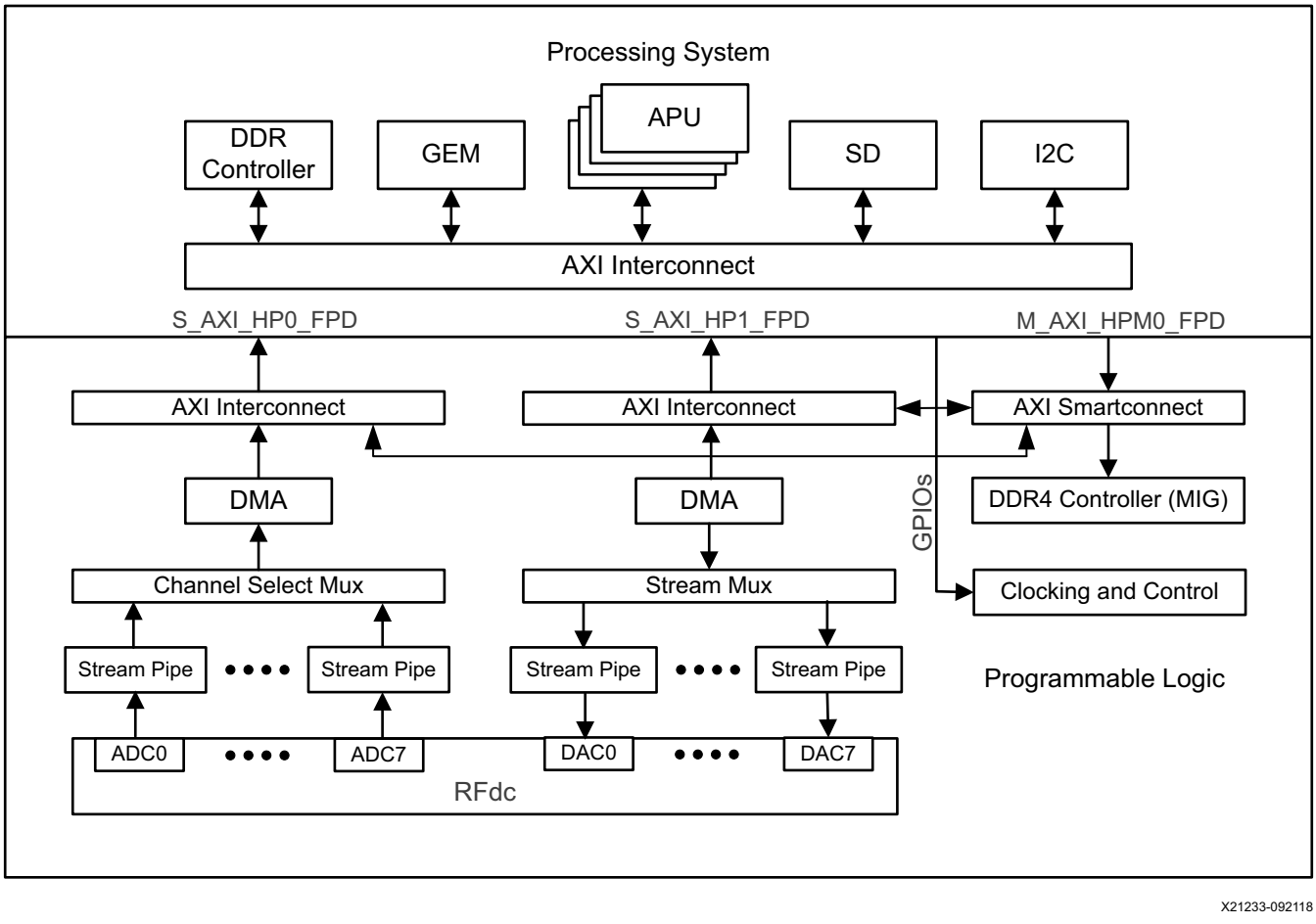


Figure 3-1: **Hardware Block Diagram**

The design is configured to operate in 8x8 mode (8-channel RF-DAC and 8-channel RF-ADC). The RFDC datapath consists of AXI DMA and Stream Pipe IPs for high performance data transfers between PS/PL DDR memories and RFDC IP. The RFDC datapath is based on AMBA AXI4-Stream protocol and the control path is based on the AXI4-Lite interface. Both datapath and control paths are implemented in the PL.

The PS is configured with a GEM Ethernet controller (GEM3) and I2C controllers (I2C0 and I2C1). The GEM Ethernet controller enables a Gigabit Ethernet interface between the host machine and the ZCU111 board. The I2C controller provides an interface between the PS and the on-board RF PLLs.

The *SD card* holds the image and file system, which loads the FPGA part when power is switched on.

The information passed from the host system GUI via Ethernet to the ZCU11 platform is stored in the PL DDR using a *DDR controller*. The application running on the processor then transfers this data to the programmable logic over the AXI ports.

The following sections provide detailed information for both RF-DAC and RF-ADC datapaths.

DAC Data Flow

Figure 3-2 shows the datapath implementation for the 8-channel RF-DAC (RF-DAC0 and RF-DAC7). There are eight stream pipes implemented. Each of these stream pipes feed data to the RF-DAC. Due to the extreme speed of the RF-DAC, it is not possible to live-feed the data using Ethernet, hence user-selected signals are continuously looped over or replayed to create a continuous and measurable signal at the output of the DAC.

There are two ways input samples are stored and replayed:

- Samples are stored and looped over in PL DDR (high storage but reduced speed).
- Samples are stored and looped over in block RAM (low storage but highest speed).

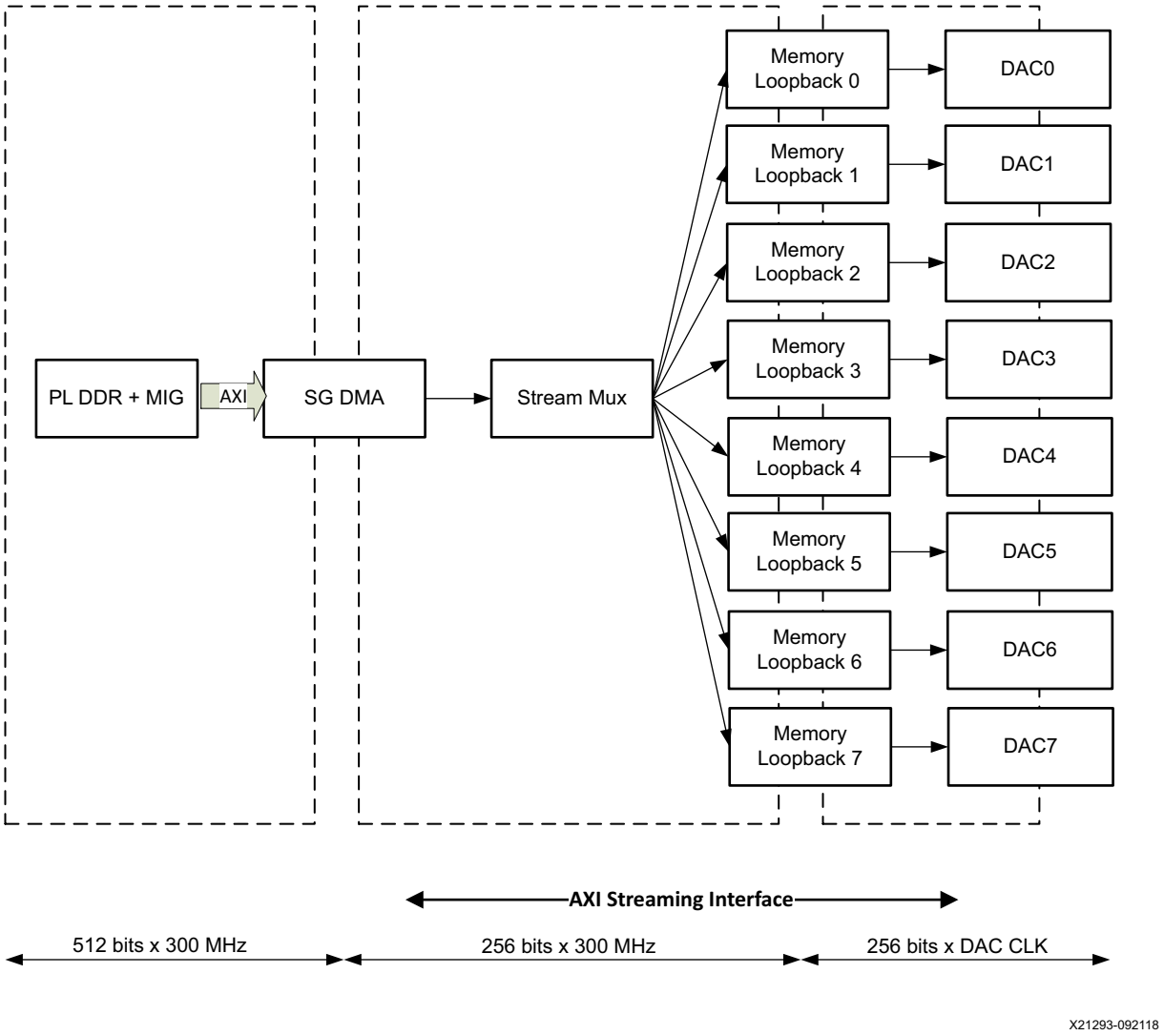


Figure 3-2: **Datapath Implementation for 8-Channel RF-DAC**

Figure 3-2 represents the architecture of the 8-channel RF-DAC (RF-DAC0 to RF-DAC7). The Scatter Gather (SG) DMA is used to source the data from the PL DDR memory controller to the DACs. The DMA sends this data to the stream MUX block, which is connected to each of the DAC channel's stream data path. Based on the channel select line input (PS-GPIOs routed through extended multiplexed I/Os (EMIOs)) of the stream pipe, the data gets routed to the corresponding RF-DAC channel. (See GPIO Selection.) In case of continuous replay from DDR, DMA constantly fetches the data and streaming mux switches the channel based on the user selection of enabled channels. For sample storage and replay from BRAM mode, the functionality of the memory loopback system is elaborated upon in Memory Loopback Details.

Streaming MUX

The streaming MUX connects the incoming pattern to the selected channel(s) based on a GUI command.

Figure 3-3 shows the stream data interface with AXIS FIFOs.

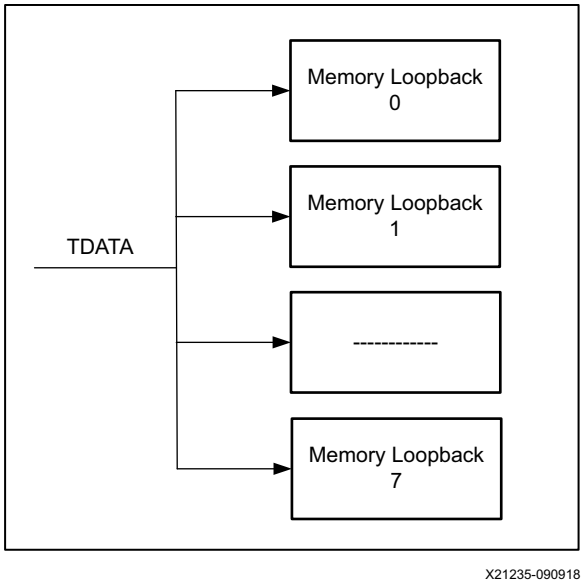


Figure 3-3: **Stream Data Interface with AXIS FIFOs**

The TDATA is broadcast as is without any additional component in the path. This helps to achieve timing closure because there is no multiplexer in the path.

The TVALID signal is ANDed with the Channel Control signal as shown in Figure 3-4. Based on user channel selection, the TVALID signal is enabled. The channel control select is asserted based on mode (BRAM or DDR). For BRAM mode, it is controlled by software using GPIOs. For DDR mode, it is controlled by a hardware logic block channel arbiter. The arbiter block allocates the DMA access among the eight streaming interfaces based on number of channels selected in the GUI. Software programs the Channel Select register based on the GUI command. After the register is programmed, the arbiter block first samples the predefined register to determine the active channels and arbitrates only among those active channels in a round robin fashion (based on the TLAST on the streaming interface), thereby effectively using the DDR bandwidth.

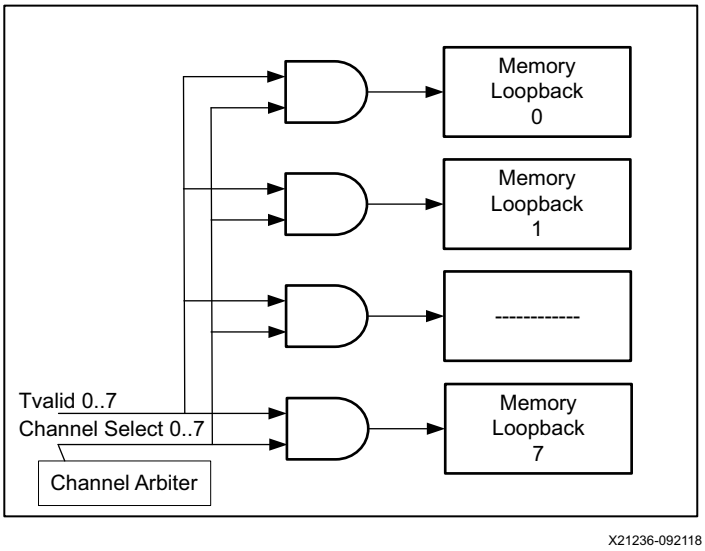


Figure 3-4: **Channel Selection Control Signals**

Figure 3-5 shows the TREADY signal generation. Only a single TREADY signal is selected based on the Channel Select signal; other TREADY signals are ignored.

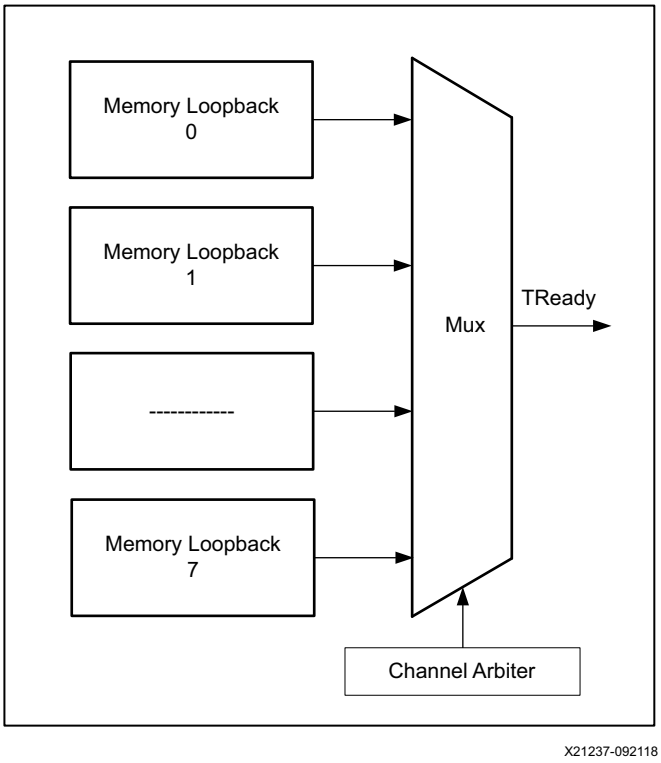


Figure 3-5: **TREADY Signal Generation**

Memory Loopback Details

Figure 3-6 illustrates the working of memory loopback (BRAM mode) in one DAC path.

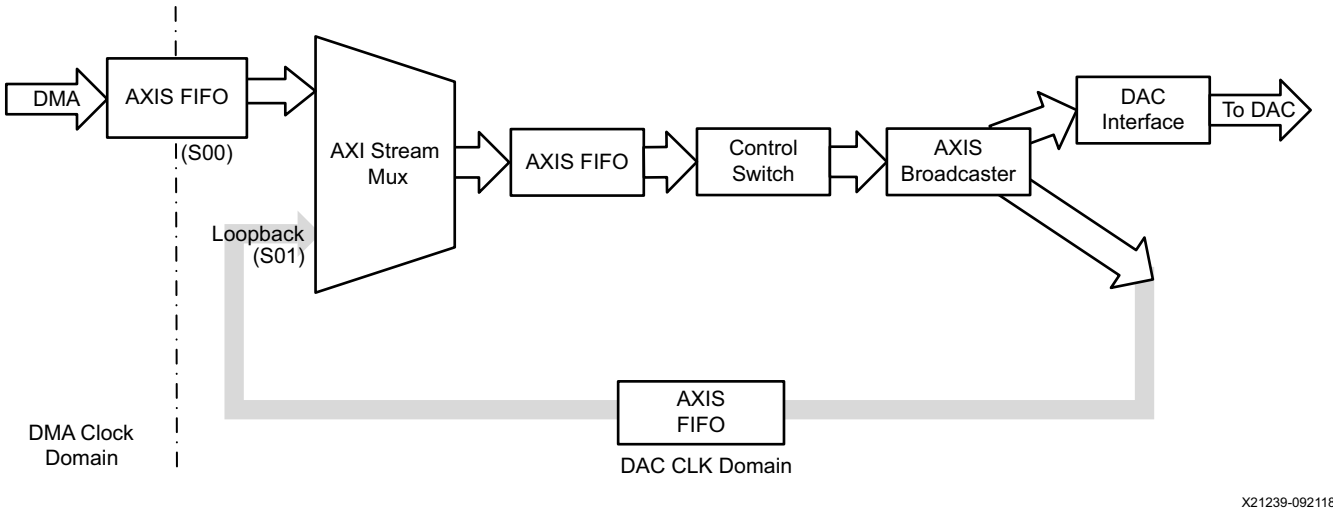


Figure 3-6: **Implementation of Memory Loopback Component**

The data coming from the DMA via the AXI4-Stream decoder (Stream MUX) is fed into an asynchronous AXI4-Stream FIFO. This takes care of the clock domain crossing between the DMA clock and DAC clock domain. The output of the FIFO is fed into an AXI4-Stream multiplexer component. This component switches between regular BRAM (Loopback) mode and DDR (Continuous Playback) mode. The control switch logic block takes input from PS-GPIOs through an EMIO interface, and when it is High (controlled by software), the output of the corresponding control logic goes High and enables the channel. The working of the control switch is further described in the DAC Control Switch section. Based on mode selection, either data is continuously replayed from AXIS FIFO to the AXIS broadcaster, or data is continuously fetched from DMA and subsequently transferred to DAC.

DAC Control Switch

The control switch is the final control logic before the streaming interface is connected to the DAC. This logic provides tight control over the streaming path and helps synchronize all the stream interfaces at any given time. Figure 3-7 shows the control switch.

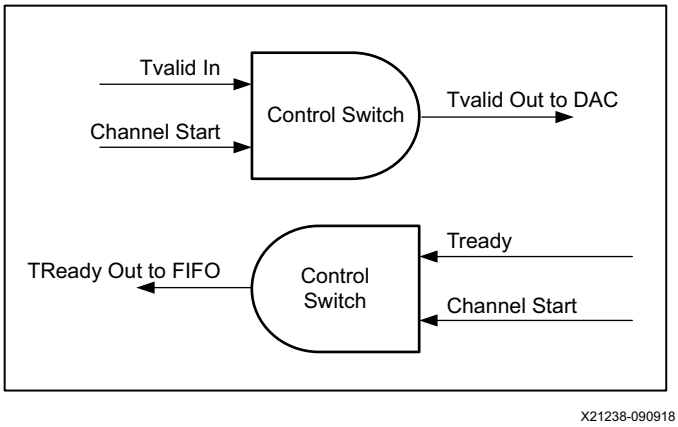


Figure 3-7: **Control Switch**

The channel control signal acts as a channel start/stop signal. This signal exists individually for all channels and can be used to control each channel independently. This is done using PS-GPIOs through the EMIO interface that are in turn controlled by software. The control switch controls TVALID input to DAC and TREADY input to FIFO, as shown in Figure 3-7. See GPIO Selection.

ADC Data Flow

Figure 3-8 shows the datapath implementation for 8-channel RF-ADC (RF-ADC0 and RF-ADC7). The default configuration of Real to IQ is enabled in the design so that all possible streaming interfaces from RF-ADC are accessible.

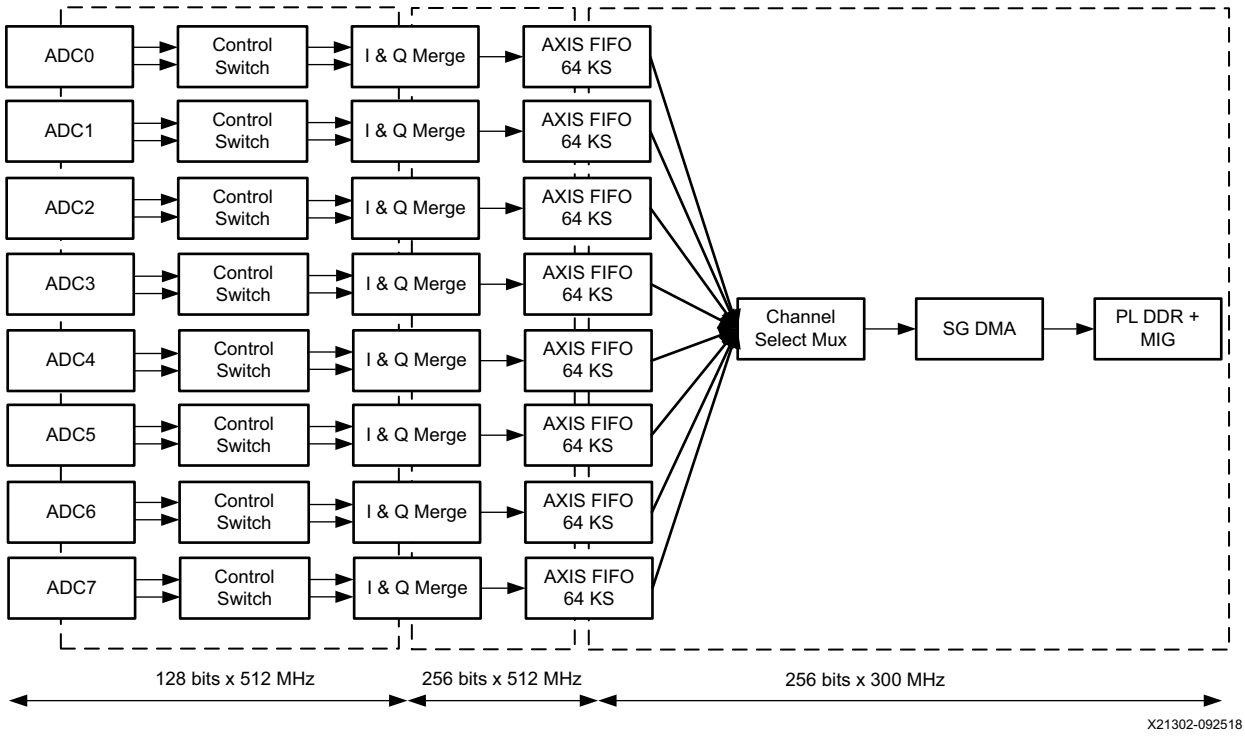


Figure 3-8: **Datapath Implementation for 8 Channel RF-ADC**

All the incoming ADC streams are gated through the control switch logic. After the gates are enabled, the I and Q streams are fed through IQ Merge logic. Based on user selection, either a real or a complex stream is passed on to the AXIS FIFO. The Channel Select MUX connects one of the AXIS FIFO to DMA, based on user selection. This data is further provided to SG DMA to be stored in PL DDR memory. The synchronization among all eight channels is achieved through control switch logic. A common global start signal is used for this purpose.

ADC Control Switch

The control switch is the final control logic before the streaming interface is connected to the ADC. This logic controls the streaming path and also helps to synchronize all the stream interfaces when required through software control. Figure 3-9 shows the control switch.

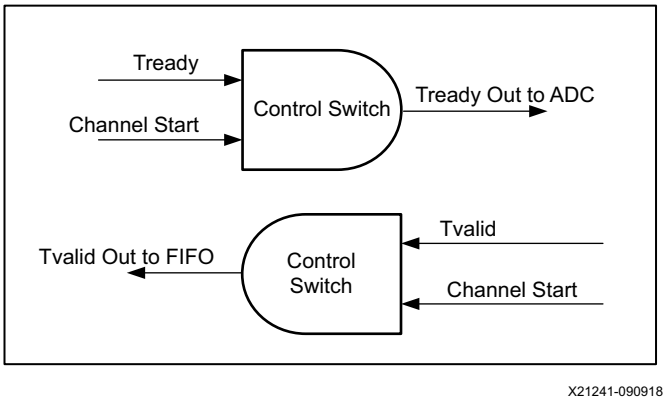


Figure 3-9: **Control Switch**

The channel select signal acts as a channel start/stop signal. This signal exists individually for all channels and can be used to control each channel independently. This is done using PS-GPIOs through the EMIO interface that are in turn controlled by software.

The control switch controls TVALID input to FIFO and TREADY input to ADC.

I & Q Merge Logic

The IQ datapath for RF-ADC is shown in Figure 3-10. This pipe can capture IQ data while maintaining synchronization and coherency. The data is captured in an interleaved fashion i.e., eight samples of I data and eight samples of Q data. The AXIS combiner IP is used to combine I and Q streams.

Alternatively, you can select a Real channel only (in which case, the width converter IP is used to convert the incoming 128-bit Real data to 256-bit Real data). Finally, the data is stored into an AXIS FIFO.

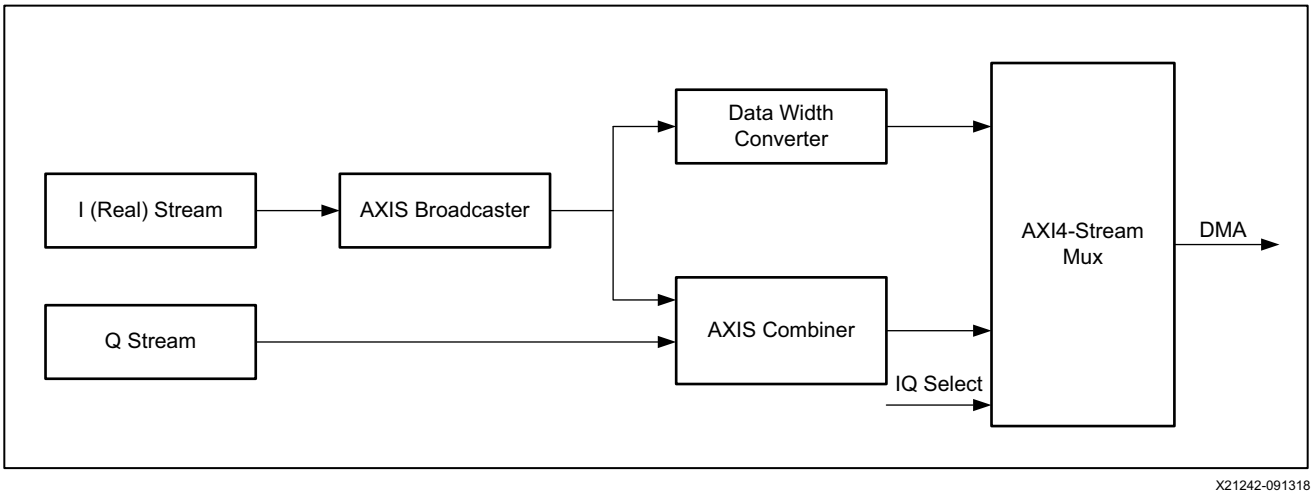


Figure 3-10: **IQ Datapath**

Channel Select MUX

The Channel MUX implementation is an 8:1 streaming selection logic as shown in Figure 3-11. Among eight incoming streaming inputs, only one streaming data is passed to SG DMA interface based on channel-select input.

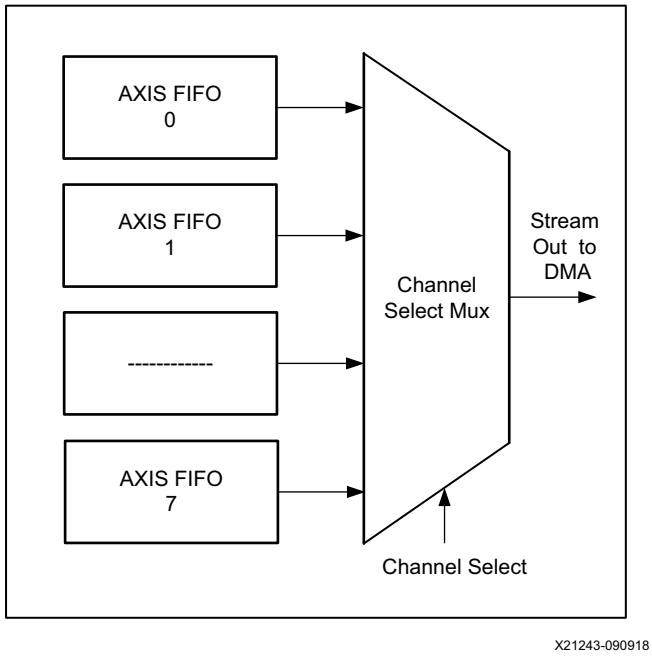


Figure 3-11: **Channel MUX**

Programmable Logic Filter

This design element illustrates the ease with which custom high-performance signal processing elements can be added to the signal chain in the Vivado IP integrator. These blocks are generated using the Xilinx System Generator flow. System Generator is a plug-in to the Simulink environment, adding a block set to the Simulink® software library browser. The block set has about 90 blocks. These blocks leverage the Xilinx IP core generators to deliver optimized results for RFSoCs. Using SysGen flow in Vivado Design Suite 2019.1, you can generate vector or multi-sample designs using super sample rate (SSR) filters. These filters help process data at a speed that matches RFDC ADC/DACs.

In the current evaluation design, SSR filters are implemented on tile 0/block0 for both ADC and DAC. The configuration of these filters is fixed to x8, with the user having an option either to bypass or enable x8 interpolation and decimation. If used along with RFDC IPs Interpolation/decimation filters, you can achieve an interpolation/decimation rate of 16x, 32x, and 64x.

Details of the implementation of the SSR filter are shown in Figure 3-12 and Figure 3-13.

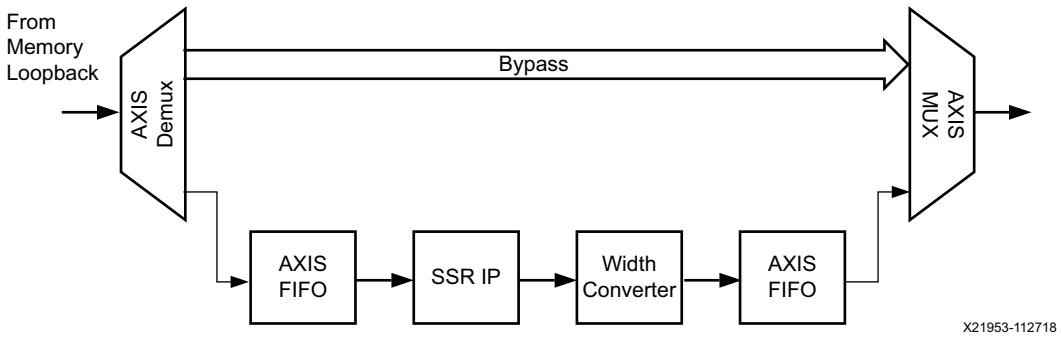


Figure 3-12: **DAC SSR IP Block Diagram**

The AXIS demux and AXIS multiplexer are is controlled through a common GPIO pin. You have the option to either bypass (disable) SSR IP or enable the SSR IP through the GPIO pin. Additional AXIS FIFO are added to the SSR IP input and output interface to provide a cushion for maintaining rate control.

Similarly, SSR IP for ADC is also enabled. Figure 3-13 shows the implementation of SSR ADC Filter IP.

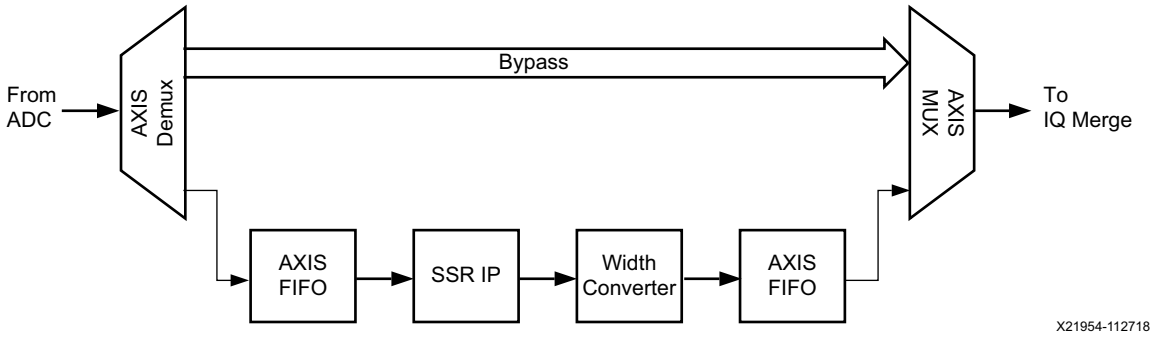


Figure 3-13: **ADC SSR IP Block Diagram**

Stream Pipes Control and Status Registers

The register sets in the design are available under the user_axilite_control block. The block has an AXI4-Lite interface through which the registers can be accessed. In the current implementation, the base address of the user_axilite_control is 0xB005_0000. Table 3-1 lists the registers that are available currently.

Table 3-1: **Control and Status Register**

Name	Offset	Description	Default Value
DAC_path_0_fifo_data_count	0x00	Indicates the count inside the FIFO	32'd0
DAC_path_1_fifo_data_count	0x04	Indicates the count inside the FIFO	32'd0
DAC_path_2_fifo_data_count	0x08	Indicates the count inside the FIFO	32'd0
DAC_path_3_fifo_data_count	0x0C	Indicates the count inside the FIFO	32'd0
DAC_path_4_fifo_data_count	0x10	Indicates the count inside the FIFO	32'd0
DAC_path_5_fifo_data_count	0x14	Indicates the count inside the FIFO	32'd0
DAC_path_6_fifo_data_count	0x18	Indicates the count inside the FIFO	32'd0
DAC_path_7_fifo_data_count	0x1C	Indicates the count inside the FIFO	32'd0
ADC_path_0_fifo_data_count	0x20	Indicates the count inside the FIFO	32'd0
ADC_path_1_fifo_data_count	0x24	Indicates the count inside the FIFO	32'd0
ADC_path_2_fifo_data_count	0x28	Indicates the count inside the FIFO	32'd0
ADC_path_3_fifo_data_count	0x2C	Indicates the count inside the FIFO	32'd0
ADC_path_4_fifo_data_count	0x30	Indicates the count inside the FIFO	32'd0
ADC_path_5_fifo_data_count	0x34	Indicates the count inside the FIFO	32'd0
ADC_path_6_fifo_data_count	0x38	Indicates the count inside the FIFO	32'd0
ADC_path_7_fifo_data_count	0x3C	Indicates the count inside the FIFO	32'd0

GPIO Selection

Table 3-2 provides the list of GPIOs for channel control, memory loopback reset, and IQ selection.

Table 3-2: **Control Signals**

DAC		ADC		Common	
Function	**GPIO#**	**Function**	**GPIO#**	**Function**	**GPIO#**
DAC0 Memory Loopback Reset	0	ADC0001 FIFO Reset	32	DAC Arbiter Reset	64
Reserved	1	ADC0001_IQ_Merge_sel	33	ADC Arbiter Reset	65
DAC0 Channel Control	2	ADC0001 Channel Control	34	Reserved	66
DAC0 Loopback select	3	Reserved	35	Reserved	67
DAC1 Memory Loopback Reset	4	ADC0203 FIFO Reset	36	DAC Multi Tile Select	68
DAC1 Loopback select	5	ADC0203_IQ_Merge_sel	37	Reserved	69
DAC1 Channel Control	6	ADC0203 Channel Control	38	Reserved	70
DAC2 Loopback select	7	Reserved	39	Reserved	79:71
DAC2 Memory Loopback Reset	8	ADC1011 FIFO Reset	40	ADC Channel Mux select	82:80
DAC3 Loopback select	9	ADC1011_IQ_Merge_sel	41	Reserved	83
DAC2 Channel Control	10	ADC1011 Channel Control	42	ADC Multi Tile Select	84
DAC4 Loopback select	11	Reserved	43	ADC Fabric Filter Select	90
DAC3 Memory Loopback Reset	12	ADC1213 FIFO Reset	44	DAC Fabric Filter Select	91
DAC5 Loopback select	13	ADC1213_IQ_Merge_sel	45		
DAC3 Channel Control	14	ADC1213 Channel Control	46		
DAC6 Loopback select	15	Reserved	47		
DAC4 Memory Loopback Reset	16	ADC2021 FIFO Reset	48		
DAC7 Loopback select	17	ADC2021_IQ_Merge_sel	49		
DAC4 Channel Control	18	ADC2021 Channel Control	50		
Reserved	19	Reserved	51		
DAC5 Memory Loopback Reset	20	ADC2223 FIFO Reset	52		
Reserved	21	ADC2223_IQ_Merge_sel	53		
DAC5 Channel Control	22	ADC2223 Channel Control	54		
Reserved	23	Reserved	55		
DAC6 Memory Loopback Reset	24	ADC3031 FIFO Reset	56		
Reserved	25	ADC3031_IQ_Merge_sel	57		

Table 3-2: **Control Signals *(Cont'd)***

DAC		ADC		Common	
DAC6 Channel Control	26	ADC3031 Channel Control	58		
Reserved	27	Reserved	59		
DAC7 Memory Loopback Reset	28	ADC3132 FIFO Reset	60		
Reserved	29	ADC3132_IQ_Merge_sel	61		
DAC7 Channel Control	30	ADC3132 Channel Control	62		
Reserved	31	Reserved	63		

Clocking

The evaluation tool design has 256 MHz, 409.625 MHz, 300 MHz, and 512 MHz clock domains. The RF-DAC output is a maximum of 409.625 MHz and RF-ADC 256 MHz. The 256 MHz output from ADC is supplied to the clocking wizard for generating 512 MHz for driving the PL ADC path whereas 409.625 MHz drives the PL DAC path directly. The 300 MHz clock is generated from the PL DDR block (ddr4_0).

Figure 4-1 shows the analog and mixed signal (AMS) clocking structure in the ZCU111 evaluation board.

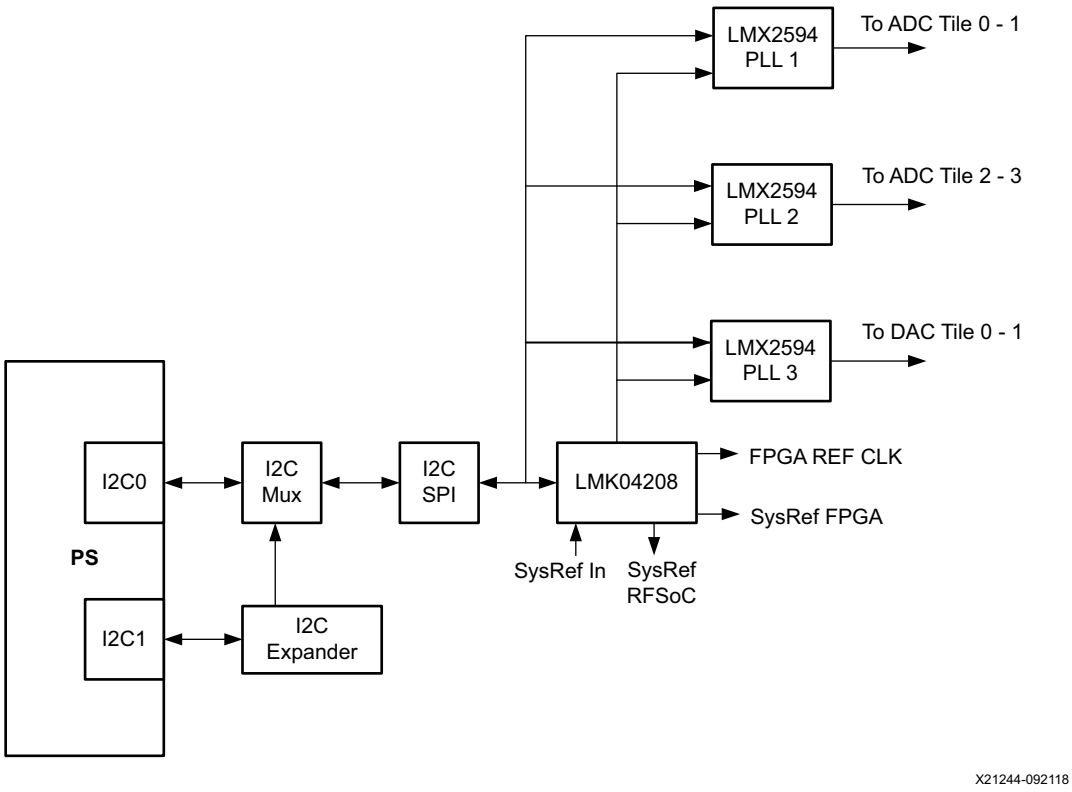


Figure 4-1: **ZCU111 AMS Clocking Structure**

The RF data converter clocking includes a primary external onboard reference PLL (LMK04208) and onboard RF PLLs (LMX2594) to generate RF-ADC and RF-DAC sample clocks. The primary management interface for the PLL devices is the Serial Protocol Interface (SPI). The SPI is enabled on the ZCU111 via an I2C to SPI bridge. The bridge components are enabled using the I2C MUX/Expander component present on ZCU111. This

scheme allows the application software to program the PLL chip sets without any external tools.

The LMK component shown in Figure 4-1 provides a low jitter and low noise clock to RFDC IP and to RF PLLs which act as a source for further clock generation and clock synchronization. LMK modules generate six clock outputs that are connected to these various blocks and components:

- PL: Sys Ref
- DAC: Sys Ref
- PL: FPGA Ref Clk
- LMX: Ref Clock
- LMX: Sync Clock
- SMA: 12.8 MHz Clock

Refer to Appendix A, Reference Design Protocol Specification for more details about clock commands and arguments.

In each Zynq UltraScale+ RFSoC, each RF-ADC or RF-DAC tile has its own clock input. Additionally, there is a dedicated input PL SYSREF pin pair per package. The PL SYSREF clock is used for multi-tile and multi-chip synchronization. For multi-tile designs, the PL SYSREF connects into a master tile and the signal is distributed within the master tile and all the other tiles in the design. The generation from PL SysRef is done using the output of the LMK module. The RFDC IP requires PL SysRef synchronously captured in the PL. For this purpose, PL SysRef provided to the IP must be synchronous to the clock domain of the group, i.e., PL clock. This is achieved using the logic diagram shown in Figure 4-2.

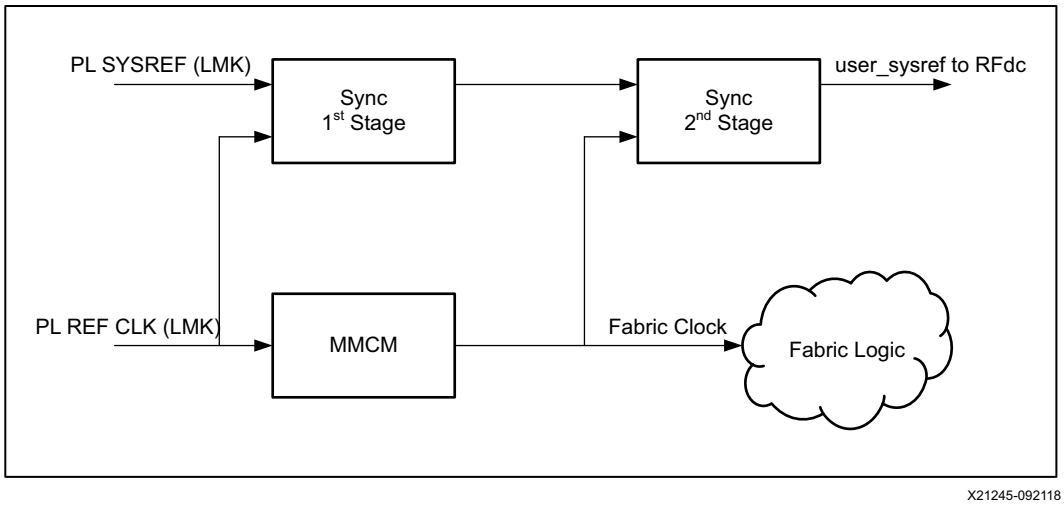


Figure 4-2: **SysRef Synchronization**

First, the PL SysRef is synchronized to the incoming PL clock for both DAC and ADC as shown in Figure 4-2. This output of the second stage synchronizer is connected to user_sysref ports of RFDC IP. Refer to *Zynq UltraScale+ RFSoC RF Data Converter LogiCORE IP Product Guide* (PG269) for more details on multi-tile synchronization [Ref 9].

Clock Switching

The design supports multi-tile synchronization (MTS) mode and non-MTS mode. The requirements for both modes are different. In MTS mode, all the DAC and ADC streams should be triggered and captured at the same time. Hence all the DAC fabric paths associated with different tiles are sourced by a common PL clock. This is the same case with all the ADC paths. In non-MTS mode, the streaming interface clocks can be sourced by their respective DAC and ADC tile clocks. The switching of the clock is controlled via a clock MUX primitive BUFGMUX (see Figure 4-3 and Figure 4-4). Clock selection depends on whether MTS mode is selected or not.

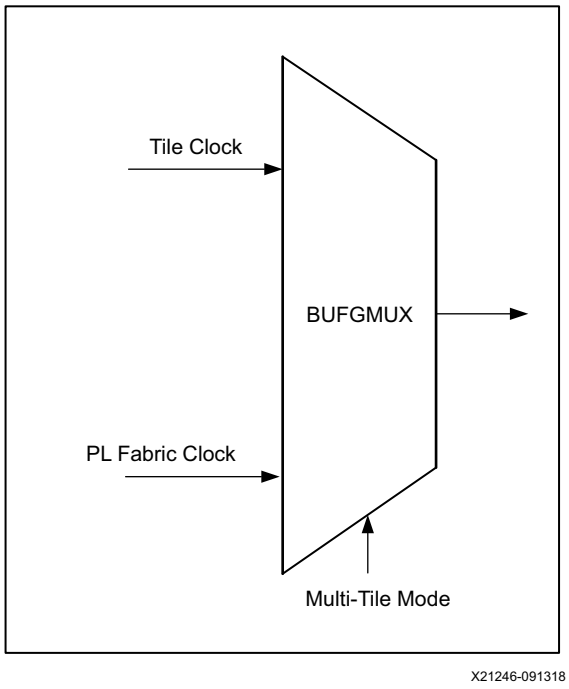
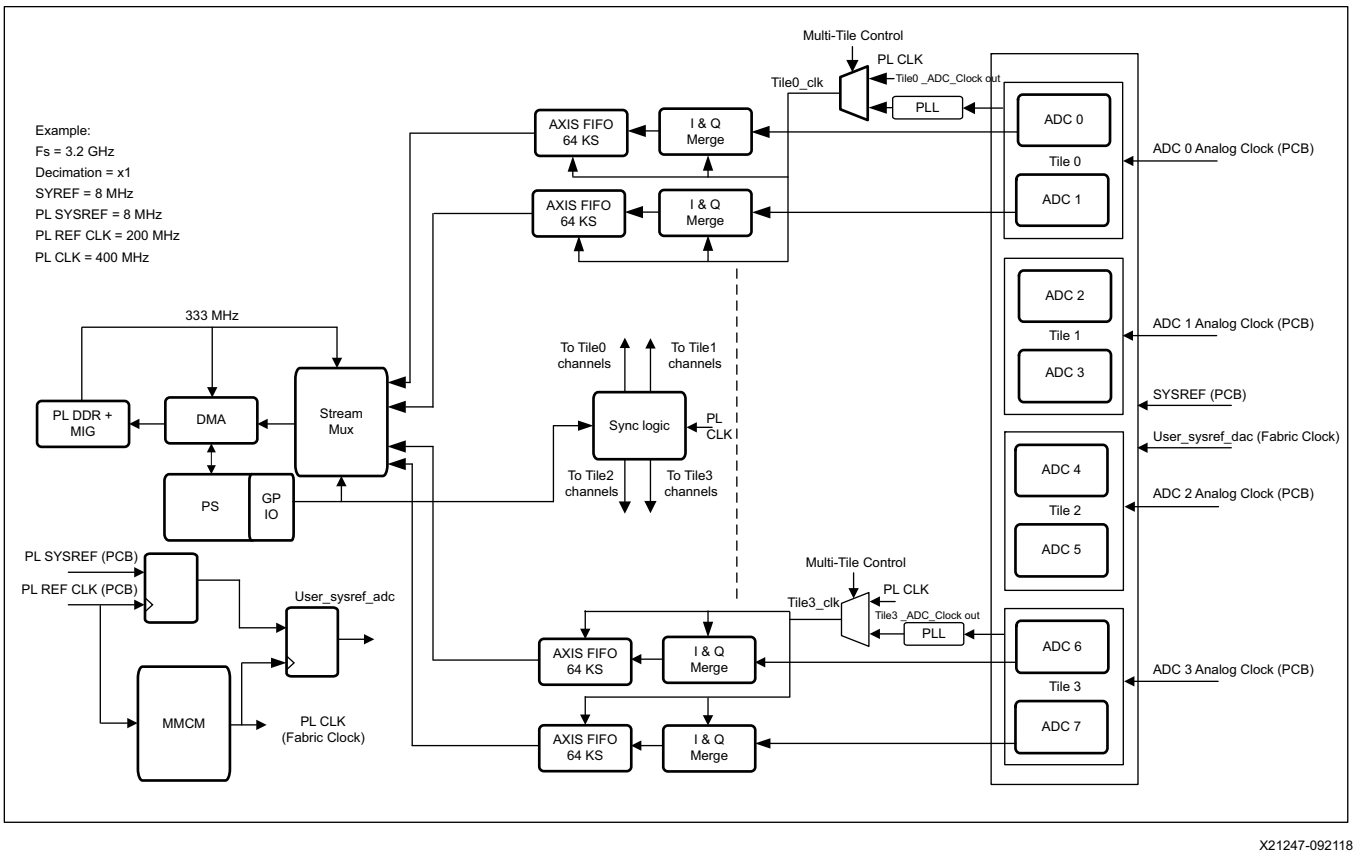


Figure 4-3: **Clock MUX Selection**

The other requirement of MTS mode is that all the incoming and outgoing streams of RFDC IP should be triggered at the same instant. This is achieved using a combination of control switch and channel control logic (Figure 4-5 and Figure 4-6). Figure 4-6 shows the channel control scheme implemented in the design. The individual channel control signal is generated using PS-GPIO pins. Later on, these signals are synchronized with respect to the selected clock from BUFGMUX. This signal acts as an individual channel start/stop signal which is fed to the control switch block of the ADC/DAC datapath. However, in MTS mode,

this signal is overridden with a global start/stop signal which is generated using Channel Select of the Master DAC block, i.e., tile 0 block 0. This signal selection is controlled using multi-tile mode select.



X21247-092118

Figure 4-4: **RF-ADC Multi-Tile Sync Clocking Structure**

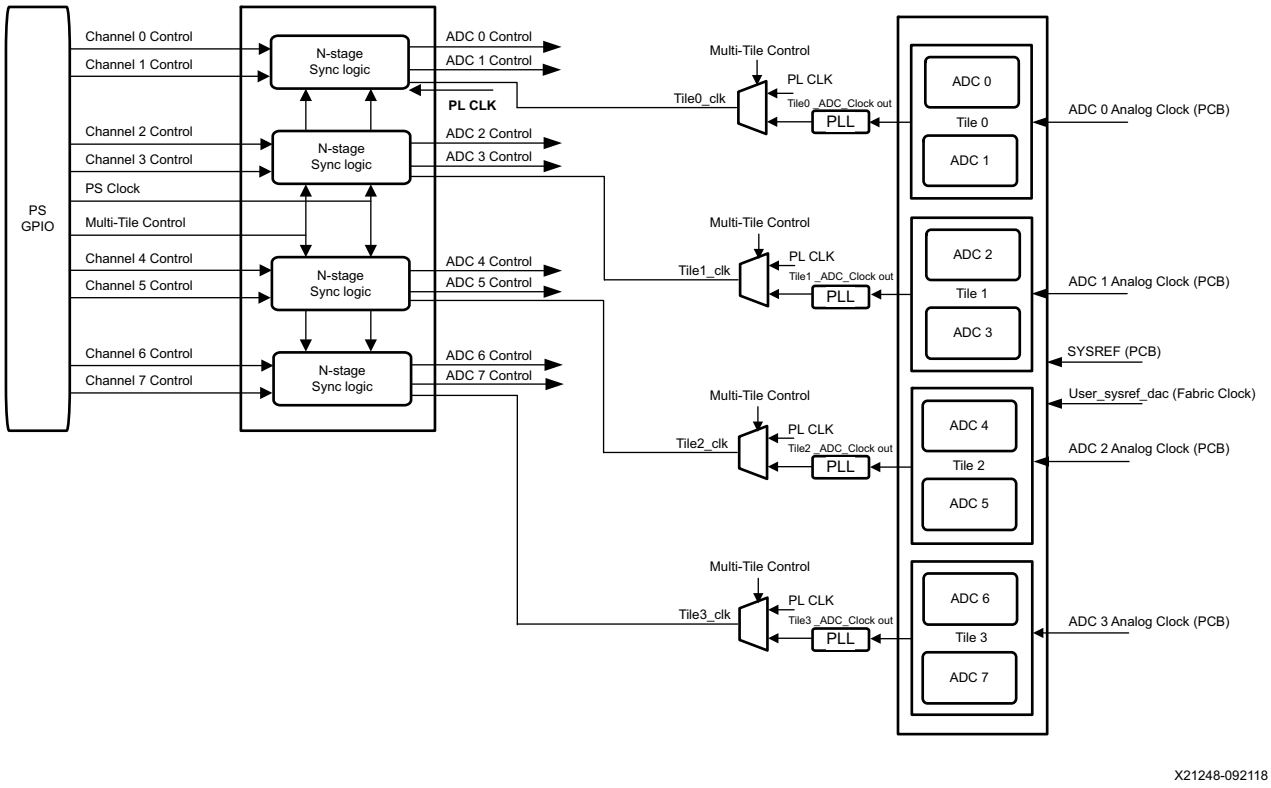


Figure 4-5: **RF-ADC Control Signals for Multi-Tile Sync (Sync Block)**

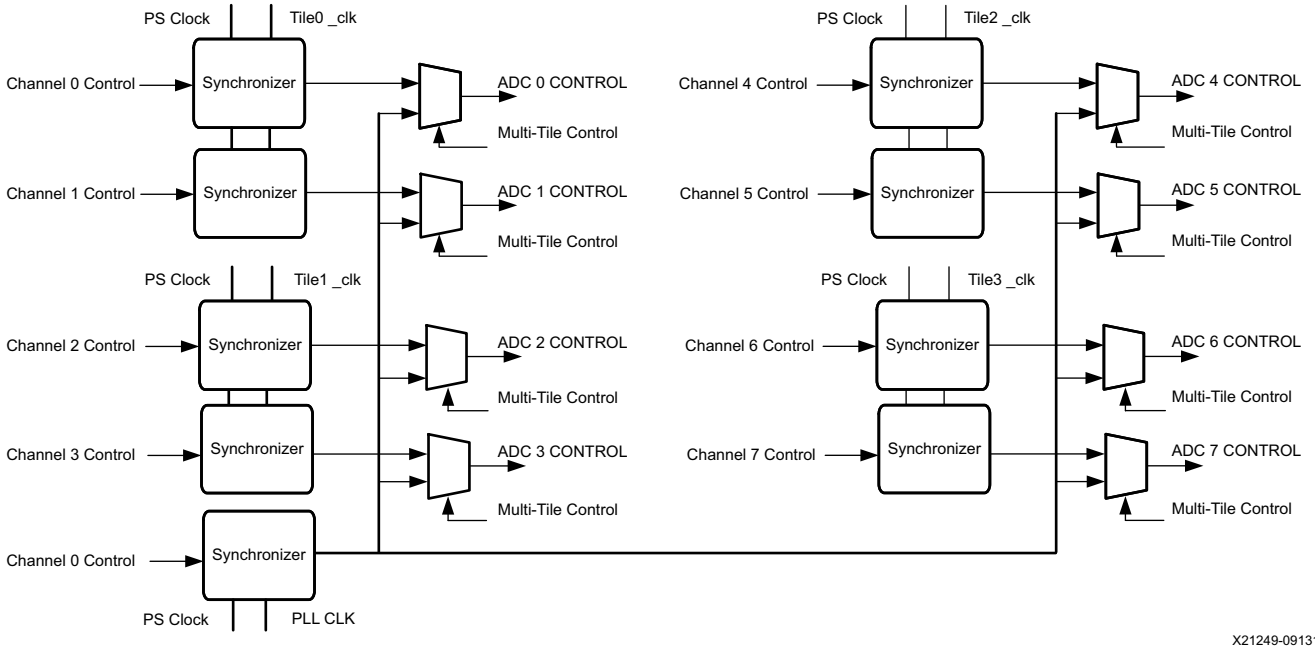


Figure 4-6: **N-Stage Sync Logic for RF-ADC**

Table 4-1 lists the clocks used for the RF-ADC control path and datapath.

Table 4-1: **Clock Domains in RF-ADC Control and Datapath**

Logic Block	ADC Stream Clock Domain (512 MHz)	Clock Domain (300 MHz)	ADC Clock Domain (256 MHz)
ADC (usp_rf_data_converter)			X
Channel select multiplexer		X	
FIFO (axis_data_fifo_0) (ADC 0-ADC7) Asynchronous FIFO	X (WRITE Clock)	X (READ Clock)	
DMA (axi_dma)		X	
AXI interconnect (AXI smart connect)		X	
PL DDR controller interface		X	
ADC_0 I & Q Merge	X		
ADC_1 I & Q Merge	X		
ADC_2 I & Q Merge	X		
ADC_3 I & Q Merge	X		
ADC_4 I & Q Merge	X		
ADC_5 I & Q Merge	X		
ADC_6 I & Q Merge	X		
ADC_7 I & Q Merge	X		

The RF-DAC has 300 MHz and DAC_CLK_OUT (409.625 MHz) clock domains. The DAC_CLK_OUT is an output of RF data converter block (usp_rf_data_converter_0) and is currently being configured to 409.625 MHz. Figure 4-7 and Figure 4-8 show the RF-DAC clock domains.

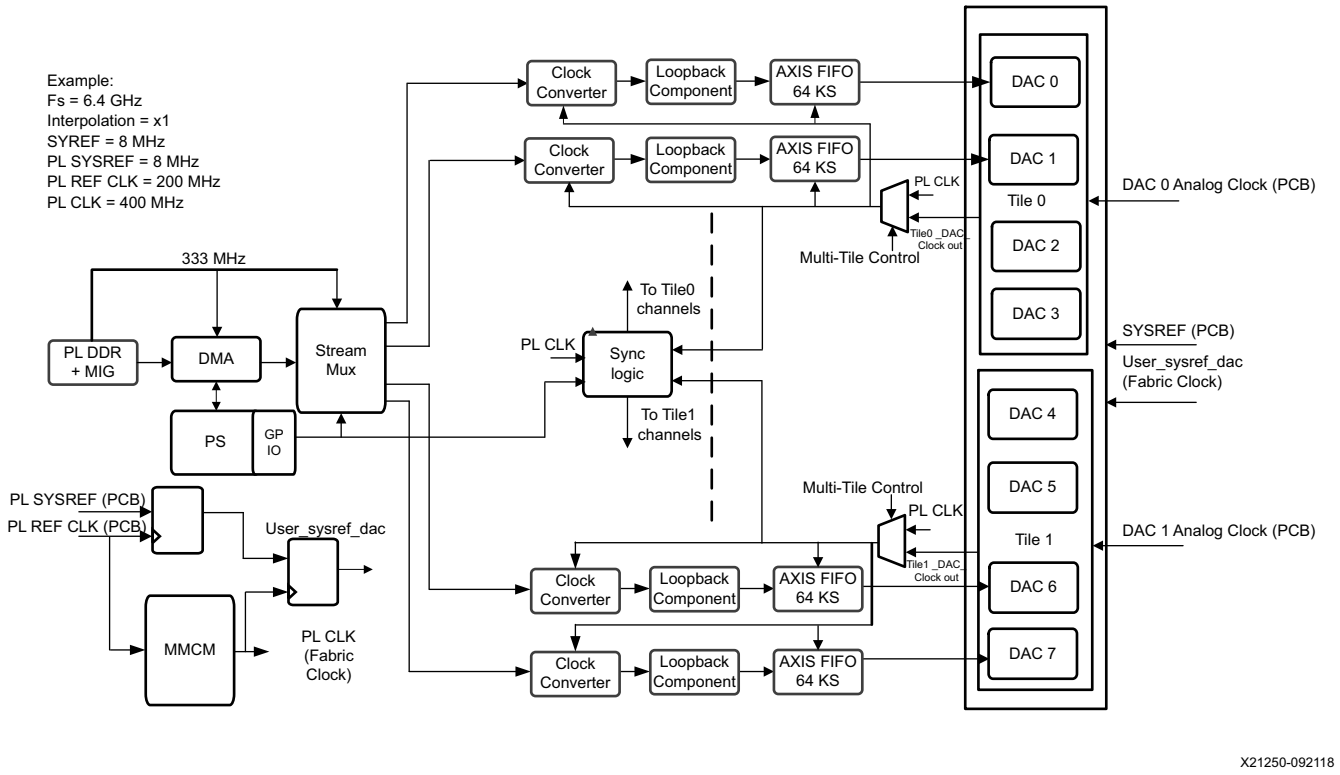


Figure 4-7: **RF-DAC Multi-Tile Sync Clocking Structure**

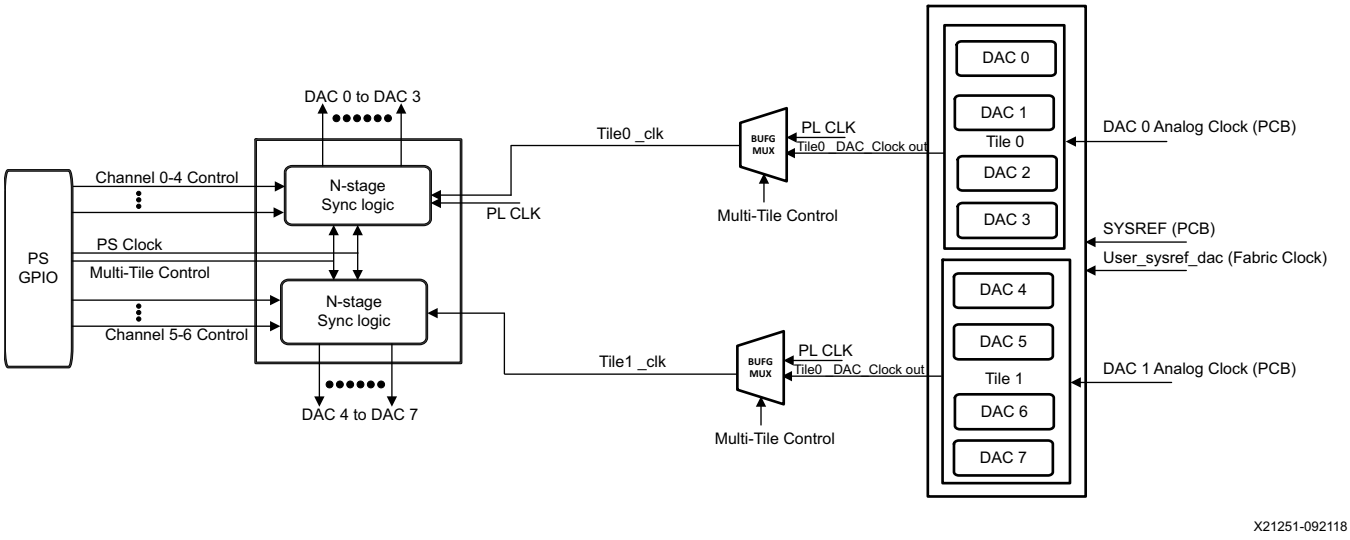


Figure 4-8: **RF-DAC Control Signals for Multi-Tile Sync (Sync Block)**

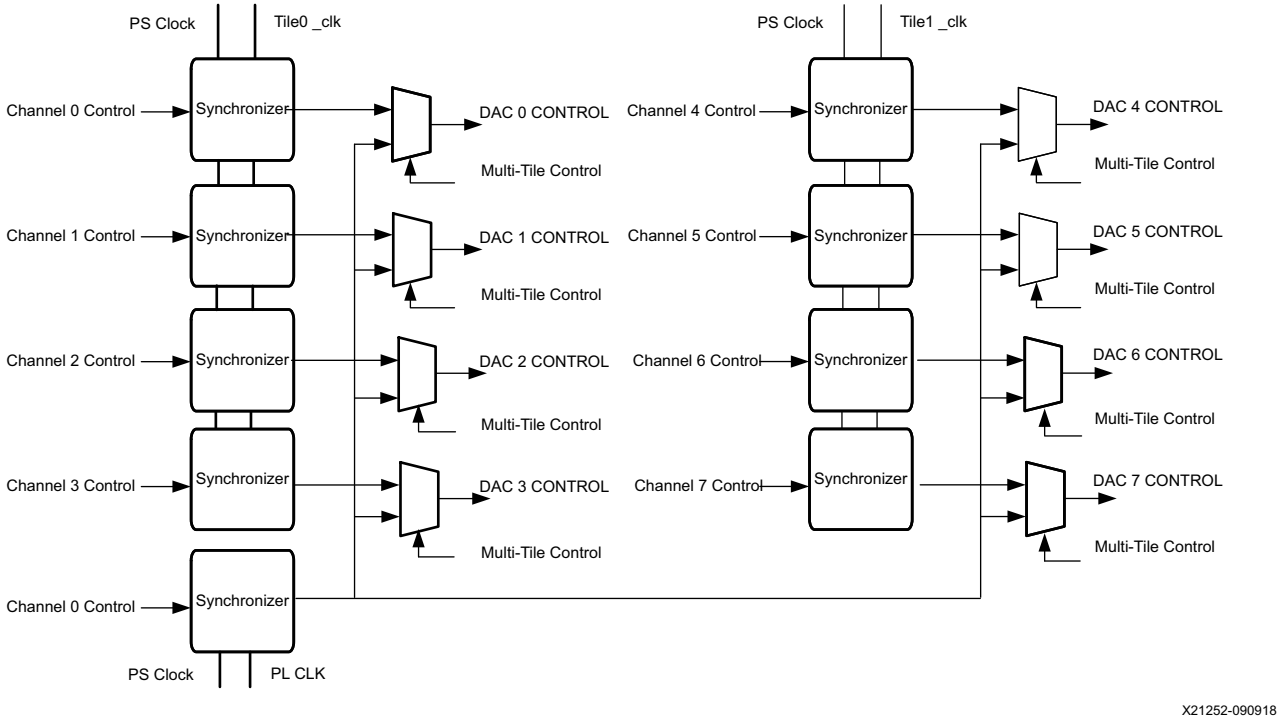


Figure 4-9: **N-Stage Sync Logic for RF-DAC**

Table 4-2 lists the clocks used for RF-DAC control path and datapath.

Table 4-2: **Clock Domains in RF-DAC Control and Datapath**

Logic Block	Clock Domain (300 MHz)	DAC_CLK_OUT (409.625 MHz)
PL DDR MIG	X	
SG DMA	X	
Streaming MUX	X	
Memory Loopback 0	x (Write)	x (Read)
Memory Loopback 1	x (Write)	x (Read)
Memory Loopback 2	x (Write)	x (Read)
Memory Loopback 3	x (Write)	x (Read)
Memory Loopback 4	x (Write)	x (Read)
Memory Loopback 5	x (Write)	x (Read)
Memory Loopback 6	x (Write)	x (Read)
Memory Loopback 7	x (Write)	x (Read)

Resets

The following reset sources are in the design:

- **pl_resetn0** (from PS to PL)

  This active-Low reset is asserted by the PS during initialization.

- **ddr4_sync_rst** (or c0_ddr4_ui_clk_sync_rst from the PL DDR memory interface group (MIG))

  This active-High reset is asserted by the PL DDR MIG. It is synchronized to the UI clock coming out the PL DDR MIG.

- **User-controlled reset block**

  This reset block is used to control the reset of FIFOs in both the RF-ADC datapaths (adc_0, adc_1) and RF-DAC datapath (dac_0). It has an AXI4-Lite interface through which the reset register can be accessed. Each bit of the register can be used to drive reset to the block appropriately.

Table 4-3: **Reset Distribution in the Evaluation Tool Design**

Logic Block	pl_resetn0	ddr4_sync_rst	User-controlled Reset Block
PL DDR MIG		X	
DAC DMA	X		
ADC DMA	X		
RF data converter AXIS slave and master interfaces (so, s1, and m0)	X		
ADC_0 block AXIS data FIFO			x (reset_0_n)
ADC_1 block AXIS data FIFO			x (reset_1_n)
ADC_2 block AXIS data FIFO			x (reset_2_n)
ADC_3 block AXIS data FIFO			x (reset_3_n)
ADC_4 block AXIS data FIFO			x (reset_4_n)
ADC_5 block AXIS data FIFO			x (reset_5_n)
ADC_6 block AXIS data FIFO			x (reset_6_n)
ADC_7 block AXIS data FIFO			x (reset_7_n)
DAC 0 block- output AXIS data FIFOs			x (dac reset_0_n)
DAC 1 block- output AXIS data FIFOs			x (dac reset_1_n)
DAC 2 block- output AXIS data FIFOs			x (dac reset_2_n)
DAC 3 block- output AXIS data FIFOs			x (dac reset_3_n)

Table 4-3: **Reset Distribution in the Evaluation Tool Design *(Cont'd)***

Logic Block	pl_resetn0	ddr4_sync_rst	User-controlled Reset Block
DAC 4 block- output AXIS data FIFOs			x (dac reset_4_n)
DAC 5 block- output AXIS data FIFOs			x (dac reset_5_n)
DAC 6 block- output AXIS data FIFOs			x (dac reset_6_n)
DAC 7 block- output AXIS data FIFOs			x (dac reset_7_n)

XILINX

Chapter 5

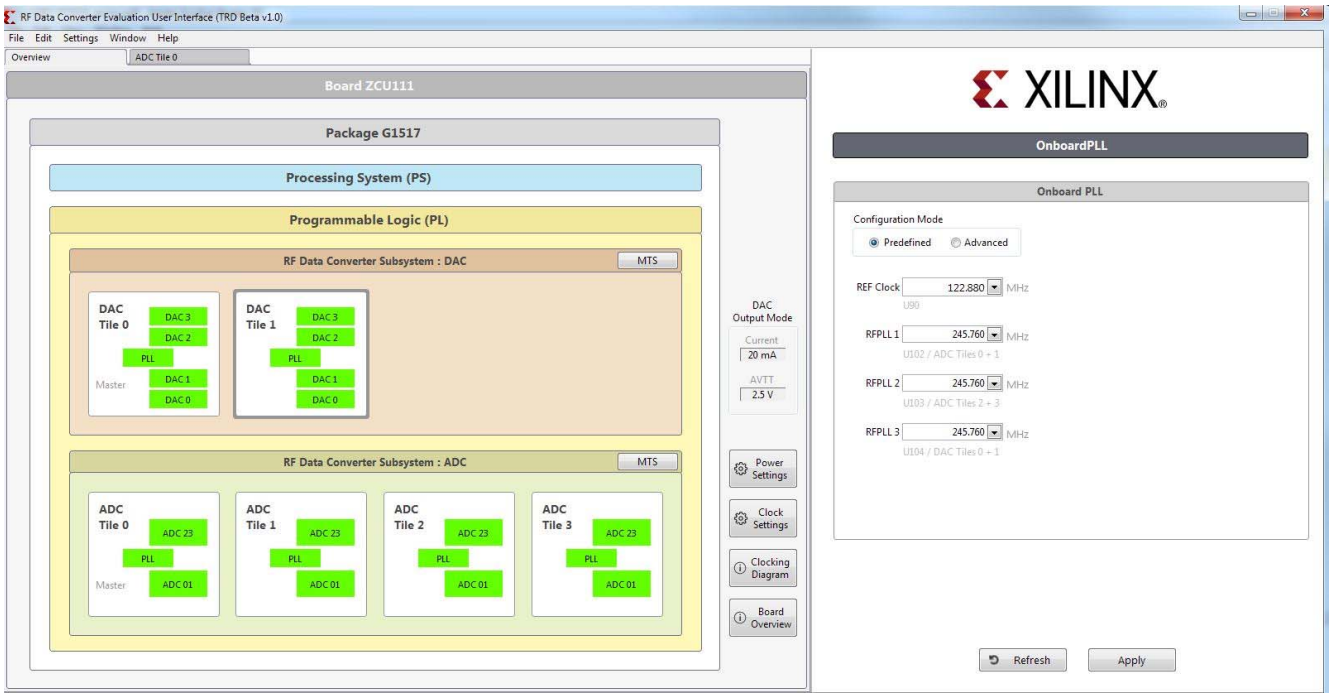
Evaluation Tool System Configuration using the GUI

The evaluation tool GUI is PC-based software that allows you to configure the operating modes of the ADCs and DACs. The GUI also generates test patterns which can be downloaded for DAC testing and manages the upload of data from the ADCs for analysis in the GUI. The GUI shipped with the Xilinx evaluation board supports most of the API functions and configuration options for the ADC and DACs. The GUI also supports the testing of all ADCs and DACs, both individually and simultaneously. Details of the supported configuration are outlined in the following sections.

The GUI is further documented in the *RF Data Converter Interface User Guide* (UG1309) [Ref 10].

External Component Configuration

In the overview tab, when clicking on **Clock Settings**, the external PLL can be configured with a set of predefined frequencies as shown in Figure 5-1.



X21283-090918

Figure 5-1: **Overview of External PLLs**

When clicking on **Power Settings**, the DAC power mode can be changed between 20 mA and 32 mA.

When selecting a tile, the tile status is displayed on the left side as shown in Figure 5-2, including the power up state machine's current state. Controls for **reset** and **shutdown/startup** are provided. A reset reconfigures the tile to its original bitstream state.

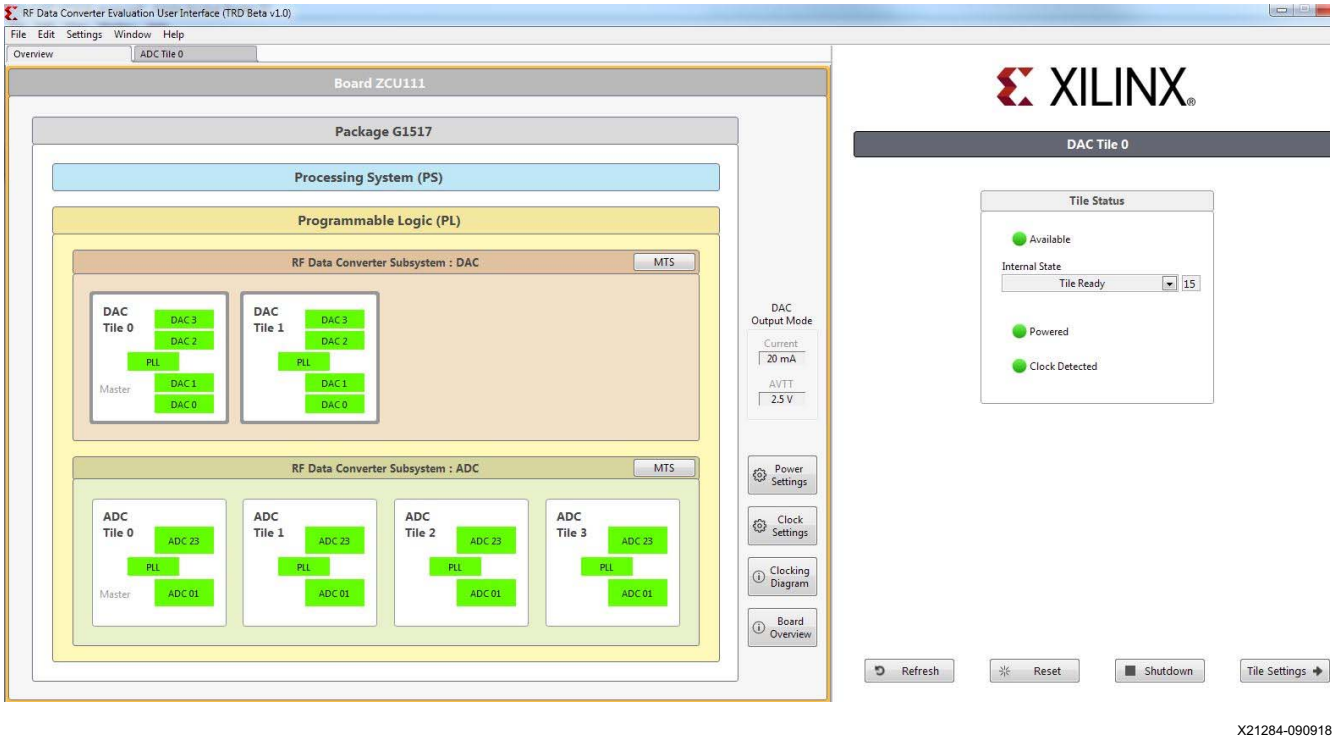


Figure 5-2: **Overview of Tile Status**

Each individual tile configuration can be accessed by double-clicking the desired tile.

ADC Configuration

The ADC tiles contain the ADCs and supporting signal processing blocks or digital down converters (DDCs). There are also clock generators or PLLs in each ADC tile. The GUI supports the configuration of all these blocks.

The ADC settings need to be optimized for certain modes of operation. The modes relate to where the signal being sampled by the ADC (Fin) lies in relation to the sampling frequency (Fs) of the ADCs. As shown in Figure 5-3, the ADC configuration is similar to the RF Data

Converter IP GUI. A notable difference is the multi-band and complex/real settings available in the **crossbar** setting shown in Figure 5-4.

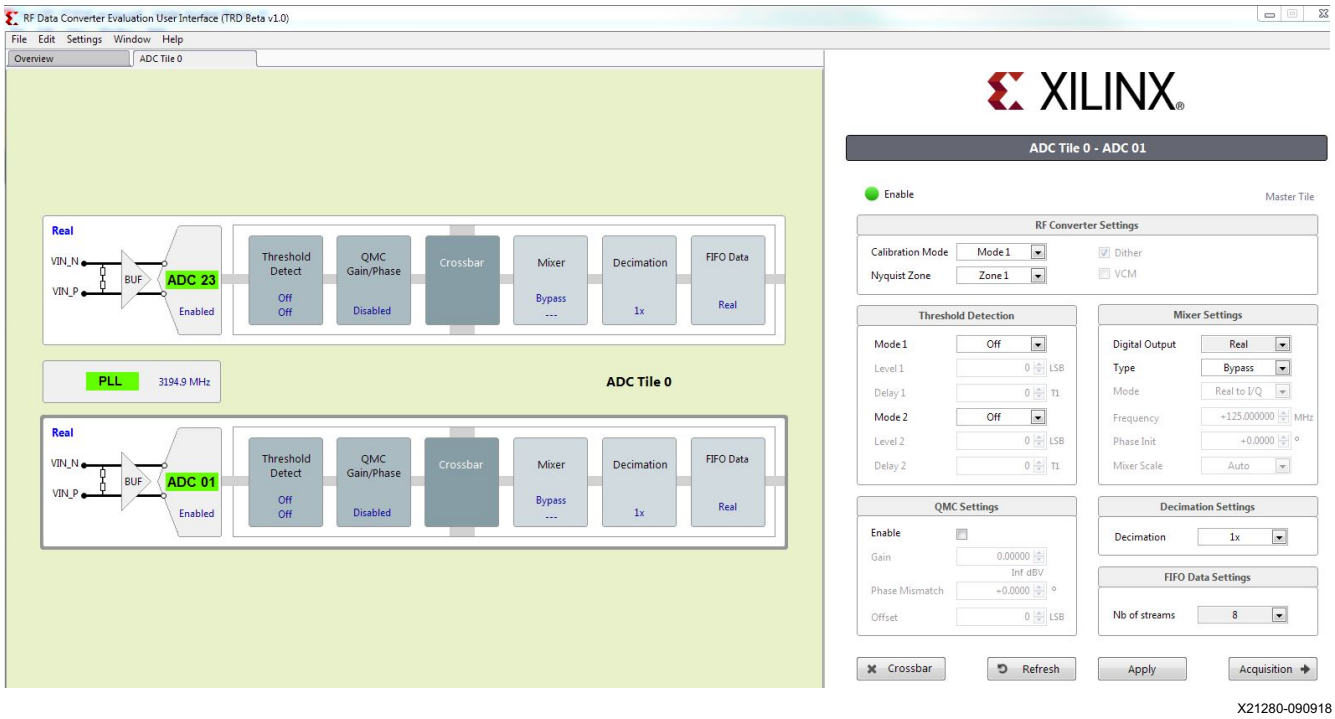


Figure 5-3: **ADC Configuration**

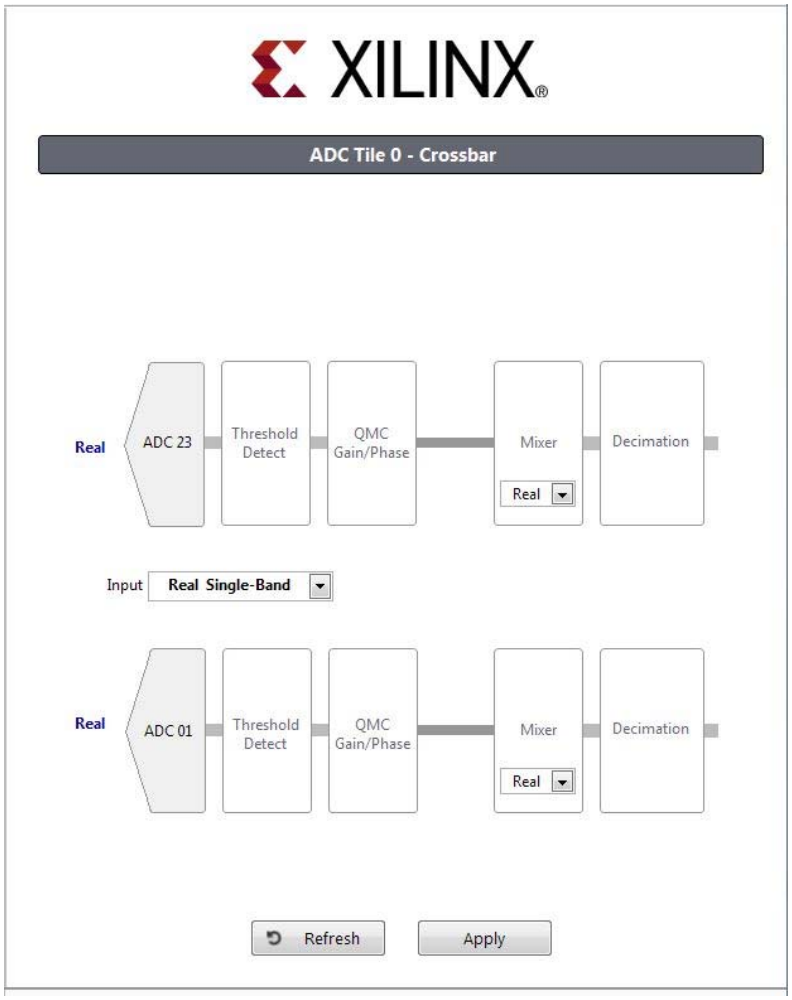


Figure 5-4: **ADC Crossbars**

ADC Clock Configuration

The GUI supports:

- Selection of external or internal (PLL) sample clock options
- On-chip PLL configuration for internal sample clock generation (see Figure 5-5).
- The configuration of the RF PLLs on the evaluation board for external clocking

Note: RFPLL (LMK) is also used to set up the reference clock for the on-chip PLLs. The nominal settings are 245.76 MHz or 491.52 MHz, for example.

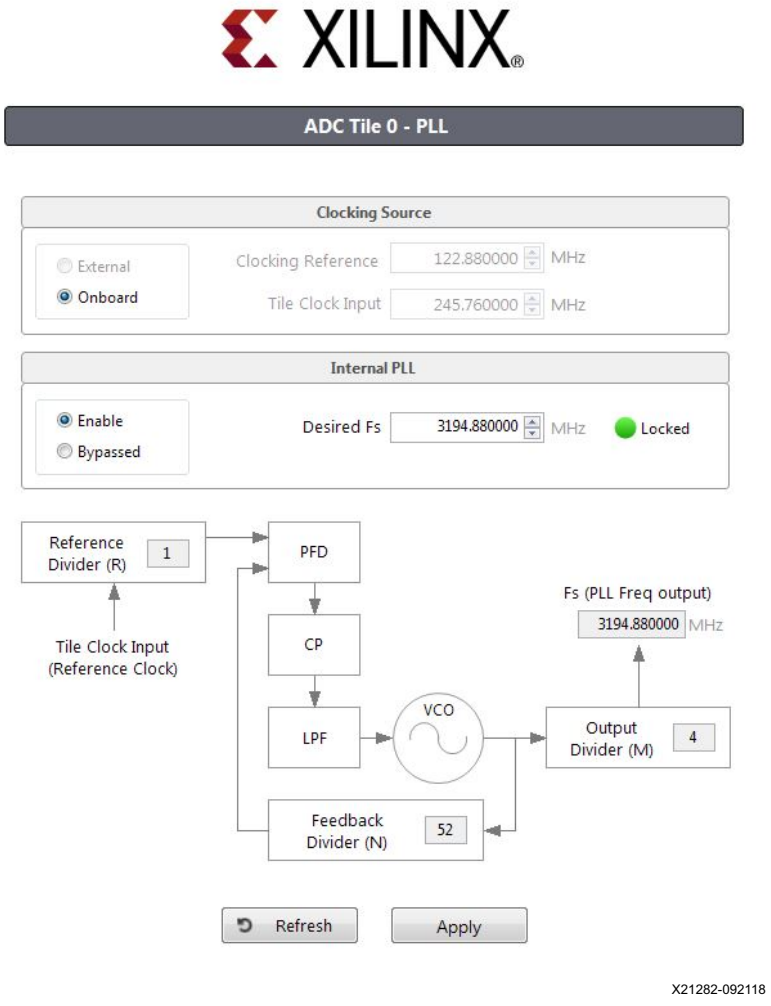


Figure 5-5: **ADC Internal PLL**

Digital Down Converter Configurations

The GUI supports:

- Setting up complex mixer functionality and NCO frequency
- Setting the decimation rate on the decimation filters
- Selecting and configuring the decimation filter in the PL, if desired
- Configuring the quadrature modulator correction block
- Enabling and configuring dual band support

DAC Configuration

Like the ADC tiles, the DAC tiles contain four DACs. Unlike the ADC tiles, there is only one configuration. The DAC tile contains the same clock generation (PLL) functionality as the ADC tiles and a digital up converter (DUC) signal processing block.

The following features are supported in the GUI:

- DAC output current range (20 mA or 32 mA)
- DAC low noise mode
- DAC enhanced linearity mode
- DAC mix mode operation for second Nyquist zone operation

DAC Clock Configurations

The GUI supports:

- Selection of external or internal (PLL) sample clock options
- On-chip PLL configuration for internal sample clock generation
- Full access to configuring the on-chip PLLs using the software API

Digital Up Converter Configurations

The GUI supports:

- Setting the complex mixer functionality and NCO frequency
- Setting the interpolation rate on the filters
- Selecting and configuring the interpolation filters in the PL, if desired
- Configuring the quadrature modulator correction block
- Enabling and configuring dual band support

ADC Tone Testing

The GUI supports uploading ADC output data for all ADC channels one by one, but the capture can happen simultaneously. The GUI supports the simultaneous capture and generation of samples in Real and Complex mode (in-phase and quadrature I/Q data).

For FFT analysis, the setup supports coherent sampling. The two basic requirements for coherent sampling are:

- The sample clocks for the DAC (or external signal generator) and ADC are frequency-locked. This is achieved if the ADC and DAC clocks are derived from the same reference clock in the case of the on-chip PLLs or external PLLs that use the same clock reference.
- The test signal (e.g., test tone) should generate an integer number of complete cycles when sampled by the ADC at the specified rate because the FFT expects a periodic signal. This is usually managed by carefully choosing the frequency of the input test tone(s) to the ADC or output tone(s) from the DAC for a fixed sample clock setting. For loopback testing, it means careful generation of the DAC output tone(s) frequency (i.e., SRAM vectors), which is sent to the DAC. This is automatically handled by the GUI for DAC for CW generation. For the ADC, the recommended coherent tones are listed to allow you to set external test equipment.

Software Architecture

This chapter describes the software platform running on the application processor unit (APU), which is logically further subdivided into user and kernel space components (see Figure 6-1). The Linux application rftool in the user space receives the command over Ethernet and performs the appropriate action. This application is the main interface to the GUI and uses a string-based communication protocol described in Chapter 7, Protocol Specification. Device drivers expose a systematic interface to control hardware. These interfaces are used by the user space application to control hardware.

All configurations of the ADCs and DACs are done by the rftool Linux application running on the PS. The application supports all the configuration options supported by the GUI. The GUI sends configuration details, via the Ethernet interface, to the Linux application, which is implemented by their respective SW API.

Software Architecture

Figure 6-1 shows the APU Linux software platform which has two logical software flows, namely control path and datapath. Datapath and control path are implemented using two different TCP sockets. The components involved in the software flows are implemented in the user space and kernel space.

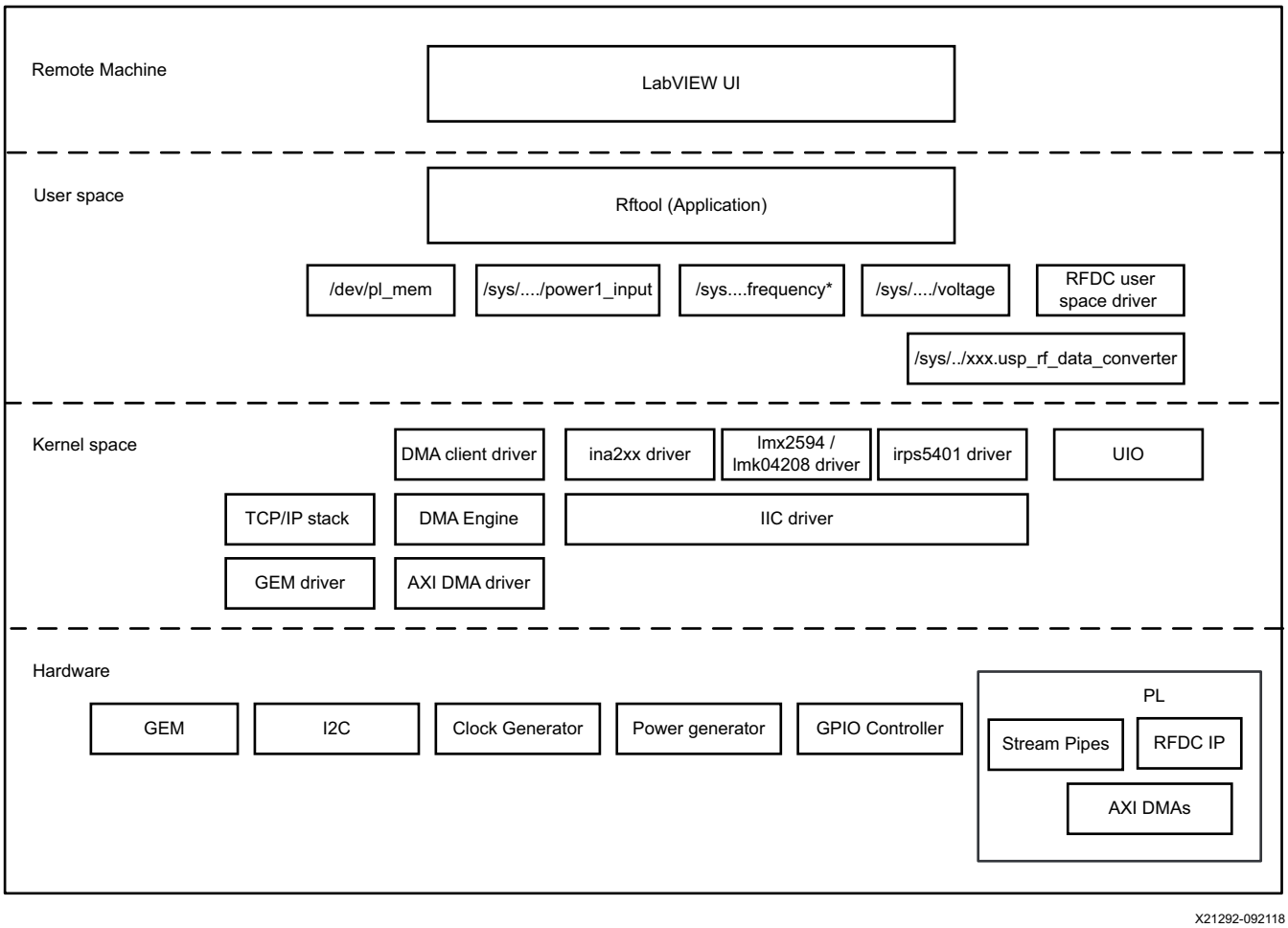


Figure 6-1: **APU Linux Software Platform**

User Space Components

- *Application* is the Linux application that receives commands over Ethernet from the PC GUI and performs appropriate actions.
- *RFDC User Space Drivers* provide APIs for communication with the RFDC hardware.
- DMA client driver interface */dev/pl_mem* is used to allocate buffer from PL DDR. It is also used to trigger a DMA transaction from the user space.

Kernel Space Components

- *GEM driver* provides the interface to send and receive packets over the GEM Ethernet controller.
- *TCP/IP* stack provides the interface to create a transmission control protocol (TCP) socket. It also provides the interface to send and receive data over the TCP socket. The stack uses the GEM driver to send and receive packets from a network interface card (NIC).
- *DMA client driver* provides the interface for a user application to trigger DMA transactions.
- The *AXI DMA driver* and *DMA Engine* are used to provide a driver interface to AXI DMA. The DMA client driver uses APIs for this DMA engine. The DMA Engine uses the AXI DMA driver to control hardware.
- *ina2xx_driver* is a Linux driver for ina2xx chips. It also provides a sysfs interface for the application to control hardware. This chip is connected over an inter IC (I2C) bus, hence the ina2xx driver uses the I2C subsystem to configure the ina2xx chip set. It is a power monitor chip from Texas Instruments (TI) on the ZCU111 evaluation board to monitor key RFSoC power rails.
- *irps5401_driver* is a Linux driver for power management ICs (PMICs). It also provides a sysfs interface for the application to control hardware. This chip is connected over an I2C bus, hence this driver uses the I2C subsystem to configure PMICs.
- *UIO* is a Linux framework used to export a hardware space to a user space. It is used by RFDC user space driver libmetal to configure RFDC hardware.

Figure 6-2 shows the application execution flow.

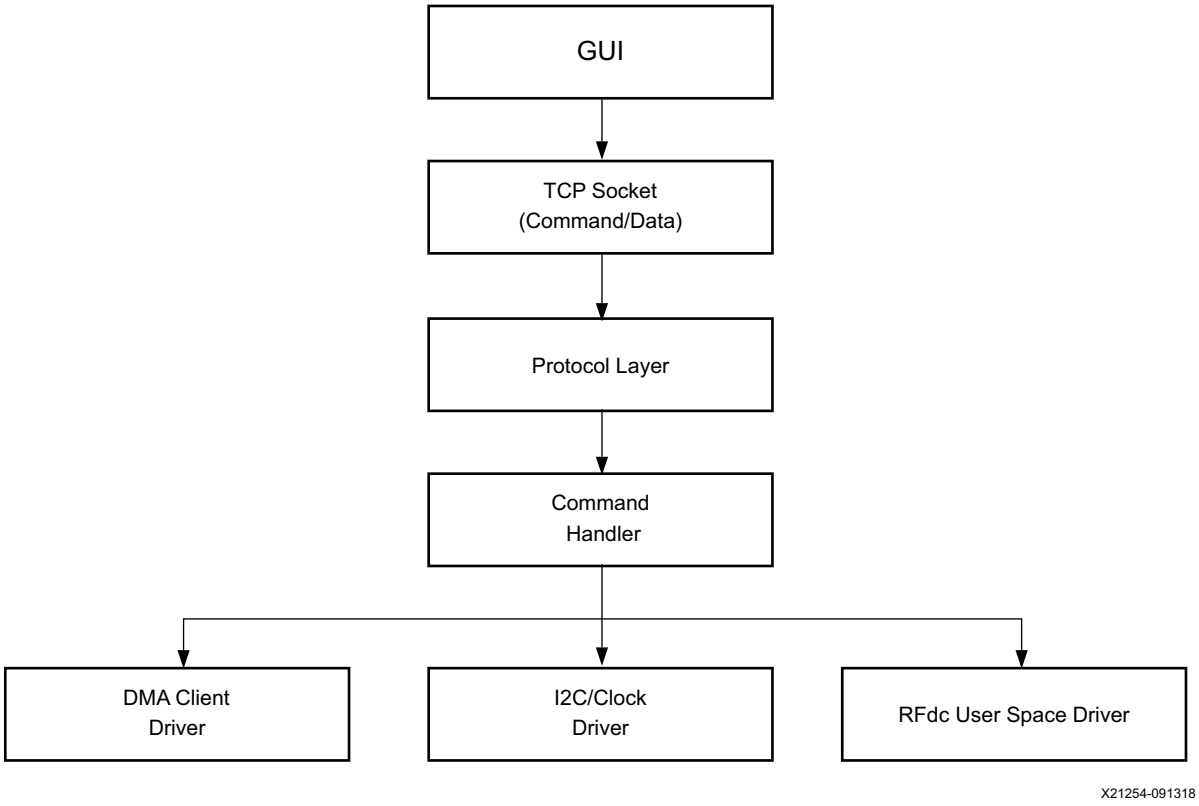
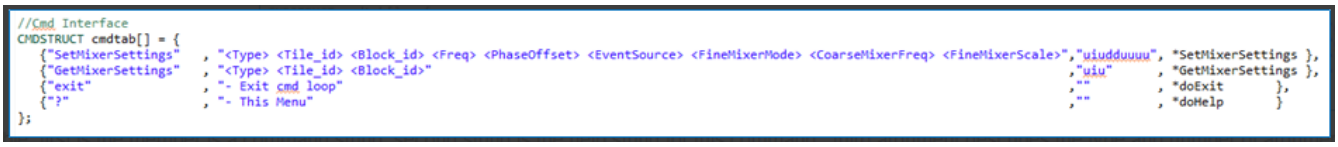


Figure 6-2: **Application Execution Flow**

Protocol Specification

The communication between the GUI and Linux application uses a string-based, space-separated command and response protocol. The Linux application maintains a table that maps command, arguments, and the corresponding method for that command (see Figure 7-1).



X21255-090918

Figure 7-1: **Command Table**

In the command table, the first member is a command string, the second string is the help string for the command, the third argument describes the type and number of arguments, and the last argument is the function pointer which needs to be called for the command.

For the SetMixerSettings command, the type of required arguments is "uiudduuuu". The type definitions are as follows:

- u: unsigned integer
- i: integer
- d: double

For example: GetMixerSetting takes three arguments: **Type**, **Tile**, **Block** and type "uiu".

```
GetMixerSetting 0 1 1
```

The above example means that it takes a type as an unsigned integer, tile ID as integer, and block ID as unsigned integer. So the command GetMixerSetting is for ADC, tile 1, and block 1.

The arguments should be in the same order as described by the third member string, e.g., **uiddduuuu**.

Socket Interface

The TCP/IP socket protocol is used for the datapath and control path. The LabVIEW GUI running on the host machine has TCP clients, and the TCP servers are running under the Linux application on the RFSoC PS. The control path works on TCP port 8081 and datapath works on TCP port 8082. The TCP socket uses a Linux TCP/IP stack, which uses the GEM Ethernet controller and driver for sending packets. Because the TCP socket is used, it is possible to run the GUI on any networked machines.

Control Path

The control interface is a generic command-response interface that can be used to call any software API or function. Its main purpose is to provide a robust communication method between a host and client application that allows RFDC hardware control commands to be issued and responses to be retrieved. All control commands are sent over TCP port 8081. At a high-level, it is *string-based*—meaning when supporting a new hardware interface, all that is required is the ability to send and receive terminated strings over that interface, and the core code remains unchanged.

Control Path State Machine

The control path is used to control and configure the RF-ADCs, RF-DACs, or board peripherals. It uses the I2C client driver or RFDC driver to configure peripherals. Figure 7-2 shows the Control/Command state machine.

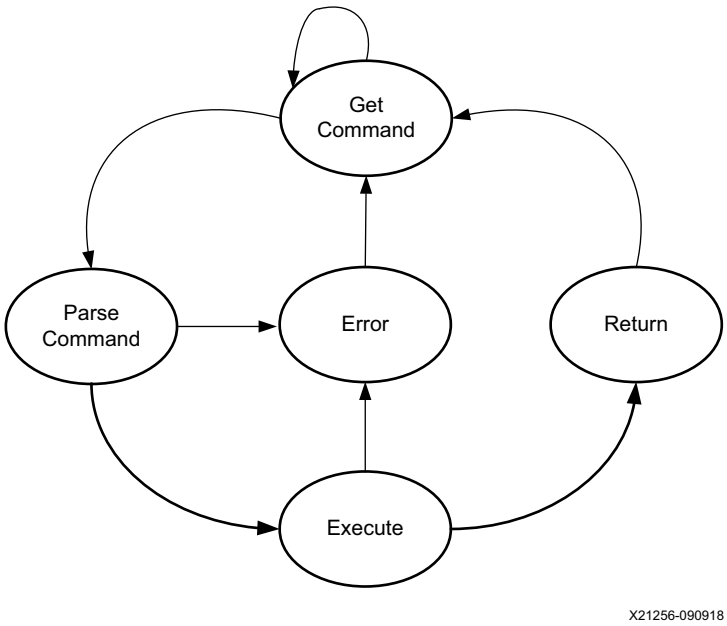


Figure 7-2: **Control/Command State Machine**

Datapath

The datapath interface is used to get the data and commands. The interface accepts the command and data on TCP port 8082. It accepts two commands: **`readdatafrommemory`** and **`writedatatomemory`**. Based on the received command, it triggers the DMA. It uses the DMA client driver to perform DMA operations. Like the control path, the datapath also accepts string-based commands.

On receiving a **`writedatatomemory`** command, the application receives the number of data bytes from the socket. After receiving data samples from the socket, the application triggers DMA. After DMA is successfully triggered, the **`writedatatomemory`** command returns status to the GUI.

On receiving a **`readdatafrommemory`** command, the application triggers DMA to receive the number of bytes specified in the command. After the DMA is done, data is sent to the GUI.

Datapath State Machine

The datapath is used to get data and commands related to data. Figure 7-3 shows the datapath state machine.

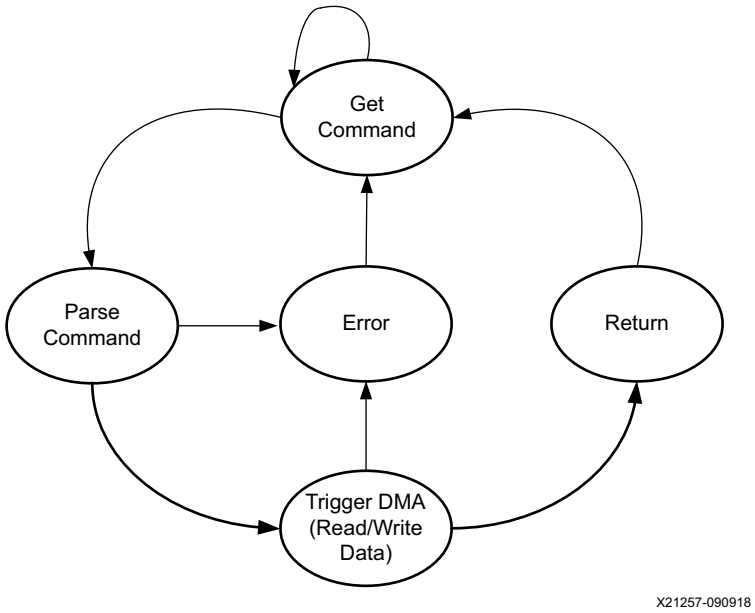


Figure 7-3: **Datapath State Machine**

Command Types

The commands for the evaluation tool fall into the following categories:

- Basic commands
- RFDC API commands—Arguments are 1:1 with the RFDC API definition, with the order specified by the order they appear in the API and C structures (Struct).
- ZCU111 onboard clock commands
- Memory read/write and data movement-related commands

Refer to Appendix A, Reference Design Protocol Specification for more details about commands and arguments.

Application Flow

Control/datapath description:

- Upon receiving a command, the parser parses it.
- If the command is incorrect, it returns an error.
- If the command is correct, it executes.
- If the execution fails, it returns an error with an explicit message.
- If the execution succeeds, it returns the command and optional return values.

Input command format:

- **`CMD PARAM1 PARAM2 PARAM3 (`**with any number of parameters)
- Commands and parameters are separated by an ASCII space.
- Commands and parameters are human readable, i.e., sent in ASCII format.
- Receive end of line is **`\n`**.
- Control path command syntax is the same as the RFSoC driver with "XRFdc_" removed.

Parameters are the same as in the RFSoC drivers.

  - Parameters for C structures are provided in the order they appear in the structure.
  - The command format is provided in the code.
  - Datapath command syntax is as below:

```
writedatatomemory <tile_id> <block_id> <Number_of_bytes> <il_pair> <DDR/BRAM>
```

1. Depending on the memory type, select **BRAM/DDR** and set GPIOs. (If it is BRAM, the hardware loops back the data; if it is DDR, DMA loops back the data.)
2. Map the memory, copy content, and trigger DMA.
3. Book-keep the memory type and addresses used for every pipeline for future use.

   ```
   readdatafrommemory <tile_id> <block_id> <Number_of_bytes> <il_pair> <BRAM/DDR>
   ```

   Check on the data length request based on mem_type. If it is BRAM, the size limit is 128K. If it is DDR, the limit is 128 MB.

For example, to send 1024-byte samples to firmware, the following command is used:

```
writedatatomemory 0 1 1024 0
```

Output return format:

- If an input command is not recognized, the parser errors out and sends **`ERROR: CMD: Invalid Command\n`**.
- If an input command is recognized, the parser checks the number of arguments, and sends
  `ERROR: CMD: Invalid Number of Arguments\n`
- If the argument count is correct, it passes the command and parameters to the **`execute`** state.
- If execution succeeds, the result is returned
  `CMD param1 param2 ... paramX value1 value2 ... valueY\n`
- Commands that are not expected to return values return their name if execution succeeds: [**`CMD`**]
- CMD and values are separated by a space.
- Values are the same as returned by the RFSoC driver.
- End of a line is **`\n`**
- If execution fails (e.g., in case of XRFdc driver failure), an error is returned
  `ERROR: CMD: Execution\n`
- Any log or messages from metal-log can be returned via the **`getlog`** command.
  - **`\r\n`** characters are replaced from any log messages with "|" and a single **`\n`** appended at the end.

Figure 7-4 shows the control path and datapath flow.

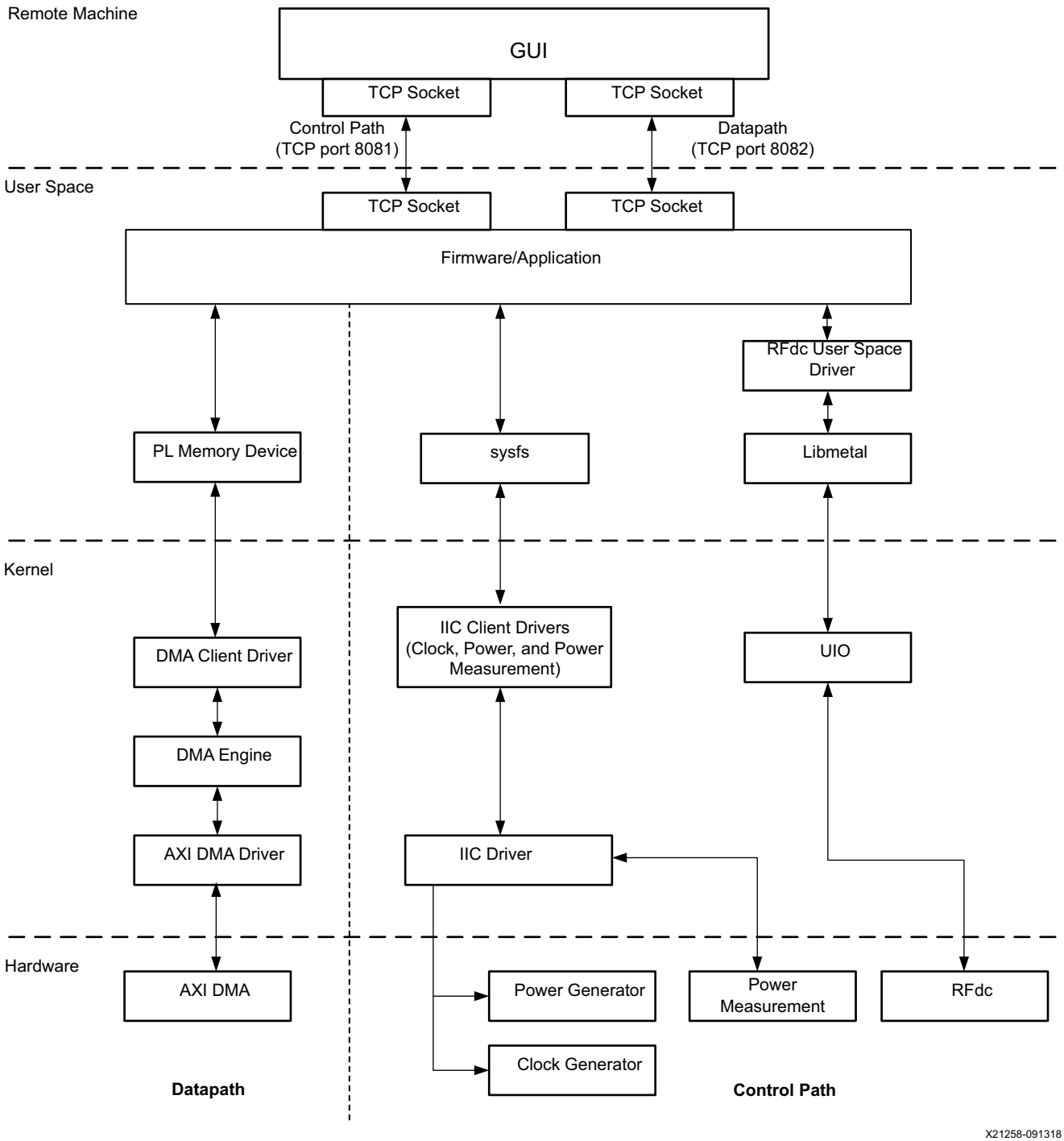


Figure 7-4: **Control and Datapath**

At a high-level, all datapaths do the same function, i.e., writing and reading data to and from memories. At a low-level, data is read and written to and from PS/PL DDR and DMA is triggered to copy data to RFDC. Datapath supports two commands: **`writedatatomemory`** and **`readdatafrommemory`**.

Using the **`writedatatomemory`** command, you can send data from a remote PC to the RFSoC board via Ethernet. On receiving this command, the socket application reads data over Ethernet and copies data to the PL-DDR. After data is copied to PL-DDR, DMA is triggered to transfer data from PL-DDR to DAC.

Using the **`readdatafrommemory`** command, you can receive data from the RFSoC board over Ethernet. On receiving this command, the socket application triggers DMA to transfer data from ADC to PL-DDR. After data is available in the PL-DDR, the socket application sends this data over Ethernet to a remote PC.

To support PL DDR mode, firmware checks the selected channels. Based on this information, the firmware creates a bitmask for the selected channels and updates the hardware register, so that the hardware knows the enabled channels.

Depending on the numbers of channels, invoke dmaengine_prep_dma_cyclic (dma_addr, length) and dmaengine_submit() multiple times. These functions prepare the BD list in the DMA driver. For each iteration, increment dma_addr by 128k for each channel, depending on the size. For example, if the number of channels is 4 and the size of the channel is 128 MB, invoke the above two functions 4096 times. Invoke dma_async_issue_pending () to start DMA.

Multi-Tile Sync

For details about the multi-tile sync feature, see the "Multi-Converter Synchronization" section of the *Zynq UltraScale+ RFSoC RF Data Converter LogiCORE IP Product Guide* (PG269) [Ref 9].

DAC Flow for Non-MTS

1. This sequence needs to be followed for each **`writedatatomemory ()`** command per DAC channel:
   a. Get Tile Id, Block ID, and size of data.
   b. Get DAC memory pointer for the corresponding DAC channel.
   c. Read data from the GUI.
   d. Disable Channel X Control GPIO (X = 0...7) for corresponding DAC.
   e. Select requested DAC channel by configuring streaming MUX GPIO.
   f. Assert external FIFO RESET for corresponding DAC channel.

g. Deassert external FIFO RESET for corresponding DAC channel.

h. Enable RFDC FIFO for corresponding DAC channel.

i. Trigger SG DMA.

j. On DMA completion, enable Channel X Control GPIO (X = 0...7) as per selected DAC.

DAC Flow for MTS

1. **`MTS_Setup (enable, DAC)`** command
   a. Disables RFDC FIFOs for all eight channels (DAC0 to DAC7).
   b. Asserts the external Memory Loopback Reset signal for all eight DAC pipelines (DAC0 to DAC7).
   c. Configures the Multi-Tile Control select signal to enable PL CLK out of BUFGMUX and to enable Channel 0 Control (common channel control signal).
   d. Deasserts external FIFO RESET signal for all eight DAC pipelines (DAC0 to DAC7).
   e. Enables RFDC FIFOs for all eight channels.
2. **`MultiConverter_Init (DAC)`** command
   a. Triggers **`XRFdc_MultiConverter_Init()`** command and returns status. This is an RFDC driver internal function.
3. **`MultiConverter_Sync (DAC, latency)`** command
   a. Triggers **`XRFdc_MultiConverter_Sync ()`** command and returns the status.
4. This sequence needs to be followed for each **`writedatatomemory()`** command per DAC channel:
   a. Get tile Id, block ID, and size of data.
   b. Get DAC memory pointer for the corresponding DAC channel.
   c. Read data from the GUI.
   d. Select the requested DAC channel by configuring streaming MUX GPIO.
   e. Trigger SG DMA (non-cyclic).
   f. Wait for DMA completion. (*Wait for DMA completion* means that TLAST is asserted at the input of the DAC path.)
   g. **`LocalMemTrigger (DAC)`** command
   h. Enable Channel 0 Control GPIO.

Note: For information on how buffers are set up, see Memory Mapping for RF-DAC/RF-ADC.

MTS Disable Flow for DAC

1. Send **`MTS_Setup (disable, DAC)`** command with disable argument.
   a. Configure Multi-Tile Control select signal to enable Tile0_DAC_Clock and Tile1_DAC_Clock out of BUFGMUX and to disable Channel 0 Control (common channel control signal).
   b. Disable channel control GPIOs (Channel 'X' Control) for all DACs (X = 0...7).
   c. Disable RFDC FIFO for all DAC pipelines (DAC0 to DAC7).

DAC Flow for PL DDR

1. Select DDR mode.
2. Generate equal sizes of patterns while loading more than one (firmware tracks the tile/block IDs and increments the channel count).
3. Write to the scratch pad register to route the BDs to the corresponding FIFOs of the DAC channel.
4. The firmware prepares the BD chain and triggers the DMA transfer.
5. Send the **`Stop`** command to reset the currently running DMA transfers (before selecting any other mode or sending new data or selection of new channels in DDR mode).

ADC Flow for MTS

1. **`MTS_Setup (enable, ADC)`** command
   a. Disable RFDC FIFOs for all eight channels (ADC0 to ADC7).
   b. Assert external FIFO RESET signal for all eight DAC pipelines (ADC0 to ADC7).
   c. Configure Multi-Tile Control select signal to enable PL CLK out of BUFGMUX and to enable Channel 0 Control (common channel control signal).
   d. Deassert external FIFO RESET signal for all eight DAC pipelines (ADC0 to ADC7).
   e. Enable RFDC FIFOs for all eight channels.
2. **`MTS_init (ADC)`** command
   a. Trigger **`XRFdc_MultiConverter_Init()`** command and return the status.
3. **`MTS_Sync (ADC, latency)`** command
   a. Trigger **`XRFdc_MultiConverter_Sync()`** command and return the status.
4. **`LocalMemTrigger (ADC)`** command
   a. Enable Channel 0 Control GPIO.

5. This sequence needs to be followed for each **`readdatafrommemory ()`** command per ADC channel.
   a. Get Tile Id, Block ID, and size of data.
   b. Select requested ADC channel by configuring streaming MUX GPIO.
   c. Enable IQ GPIO, if IQ mode is selected.
   d. Trigger DMA.
   e. Send requested ADC data to GUI.

ADC Flow for Non-MTS

This sequence needs to be followed for each **`readdatafrommemory`**() command per ADC channel:

1. Get Tile Id, Block ID, and size of data.
2. Get ADC memory pointer for the corresponding channel.
3. Select requested ADC channel by configuring streaming MUX GPIO.
4. Enable IQ GPIO, if IQ mode is selected.
5. Assert external FIFO RESET for corresponding ADC channel.
6. Deassert external FIFO RESET for corresponding ADC channel.
7. Enable RFDC FIFO of corresponding ADC channel.
8. Enable Channel X Control GPIO (X = 0...7) as per selected ADC.
9. Trigger SG DMA.
10. Disable RFDC FIFO of the corresponding ADC channel.
11. Send requested ADC data to the GUI.

MTS Disable Flow for ADC

1. Send **`MTS_Setup (disable, ADC)`** command with the disable argument.
   a. Configure Multi-Tile Control select signal to enable Tile0_ADC_Clock, Tile1_ADC_Clock, Tile2_ADC_Clock, and Tile3_ADC_Clock, out of BUFGMUX and to disable Channel 0 Control (common channel control signal).
   b. Disable channel control GPIOs (Channel 'X' Control) for all ADCs.
   c. Disable RFDC FIFO for all ADC pipelines (ADC0 to ADC7).

XILINX

Chapter 8

Zynq UltraScale+ RFSoC Data Converter Bare-metal/Linux Driver

The Linux APIs for the Zynq® UltraScale+™ RFSoC Data Converter is described in the *Zynq UltraScale+ RFSoC Data Converter Bare-metal/Linux Driver* appendix of *Zynq UltraScale+ RFSoC RF Data Converter LogiCORE IP Product Guide* (PG269) [Ref 9].

System Considerations

This chapter describes the boot process and address mapping.

Boot Process

The design uses a non-secure boot flow and SD boot mode. The sequence diagram in Figure 9-1 shows the steps and order in which the individual boot components are loaded and executed.

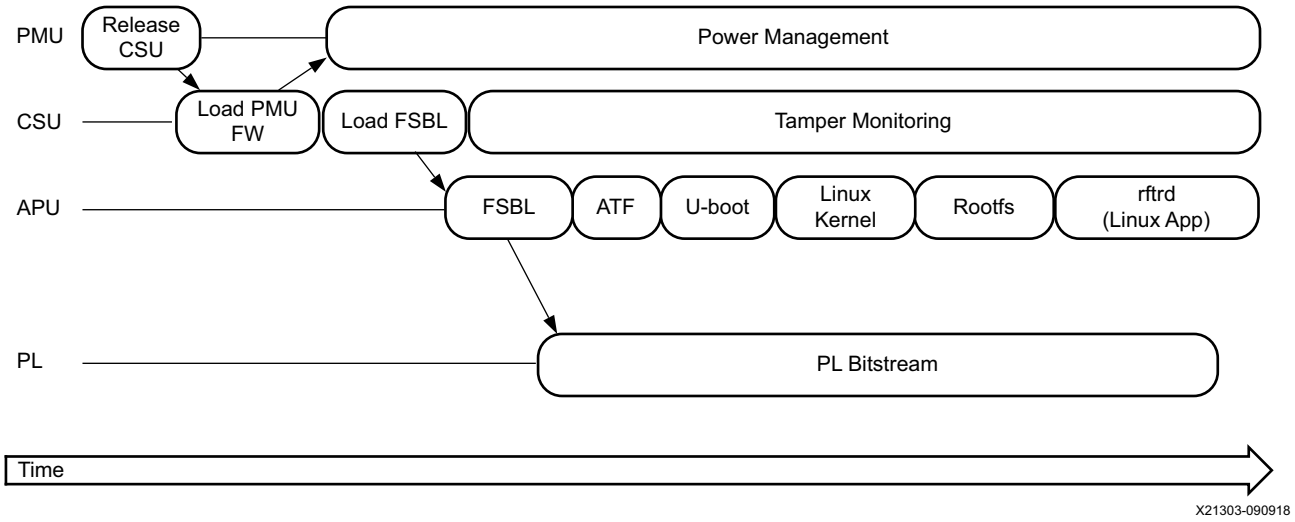


Figure 9-1: **Boot Flow Sequence**

The platform management unit (PMU) is responsible for handling primary pre-boot tasks and is the first unit to wake up after power-on reset (POR). After the initial boot process, the PMU continues to run and is responsible for handling various clocks and resets of the system as well as system power management. In the pre-configuration stage, the PMU executes the PMU ROM and releases the reset of the configuration security unit (CSU). It then enters the PMU server mode where it provides platform management functions.

The CSU handles the configuration stages and executes the boot ROM as soon as it comes out of reset. The boot ROM determines the boot mode by reading the boot mode register, it initializes the on-chip memory (OCM), and reads the boot header. The CSU loads the PMU firmware into the PMU RAM and signals to the PMU to execute the firmware, which

provides advanced management features instead of the PMU ROM. It then loads the first stage boot loader (FSBL) into OCM and, if enabled, switches into tamper monitoring mode.

In this design, the FSBL is executed on APU-0. It initializes the PS and configures the PL and APU based on the boot image header information. The following steps are performed:

1. The PL is configured with a bitstream and the PL reset is deasserted.
2. The ARM trusted firmware (ATF) is loaded into OCM and executed on APU-0.
3. The second stage boot loader U-Boot is loaded into DDR to be executed by APU-0.

For more information on the boot process, see chapters *Programming View of Zynq UltraScale+ MPSoC Devices* and *System Boot and Configuration* in the *Zynq UltraScale+ MPSoC Software Developer Guide* (UG1137) [Ref 11], and chapter *Boot and Configuration* in the *Zynq UltraScale+ Device Technical Reference Manual* (UG1085) [Ref 3].

Global Address Map

For more information on system addresses, see the *System Addresses* chapter in the *Zynq UltraScale+ Device Technical Reference Manual* (UG1085) [Ref 3].

Memory

The DMA instances in the PL use a 36-bit address space so they can access the DDR Low and DDR High address. Table 9-1 lists the APU software components used in this design and where they are stored or executed from in memory.

Table 9-1: **Software Executables and Their Memory Regions**

Component	Processing Unit	Memory
FSBL	APU-0	OCM
ARM trusted firmware (ATF)	APU-0	OCM
U-Boot	APU-0	DDR
Linux kernel/device tree/rootfs	APU (SMP)	DDR
rftool application (Linux)	APU (SMP)	DDR

Memory Mapping for RF-DAC/RF-ADC

PL DDR is divided into four partitions (reserved memory), and each partition is 512 MB. It can run either RF-DAC/RF-ADC from these partitions. It can run either RF-DAC/RF-ADC from this partition. Table 9-2 lists partitions, their starting address, and their size.

Table 9-2: **Memory Mapping for RF-DAC/RF-ADC**

Partition	DDR	Start Address	Size	Component
1	PL	0x410000000	128MB	RF-DAC/RF-ADC
2	PL	0x418000000	128MB	RF-DAC/RF-ADC
3	PL	0x420000000	128MB	RF-DAC/RF-ADC
4	PL	0x428000000	128MB	RF-DAC/RF-ADC
5	PL	0x430000000	128MB	RF-DAC/RF-ADC
6	PL	0x438000000	128MB	RF-DAC/RF-ADC
7	PL	0x440000000	128MB	RF-DAC/RF-ADC
8	PL	0x448000000	128MB	RF-DAC/RF-ADC
9	PL	0x450000000	128MB	RF-DAC/RF-ADC
10	PL	0x458000000	128MB	RF-DAC/RF-ADC
11	PL	0x460000000	128MB	RF-DAC/RF-ADC
12	PL	0x468000000	128MB	RF-DAC/RF-ADC
13	PL	0x470000000	128MB	RF-DAC/RF-ADC
14	PL	0x478000000	128MB	RF-DAC/RF-ADC
15	PL	0x480000000	128MB	RF-DAC/RF-ADC
16	PL	0x488000000	128MB	RF-DAC/RF-ADC

XILINX

Appendix A

Reference Design Protocol Specification

Commands

Table A-1 lists the commands used in the evaluation tool design.

Table A-1: **Command List**

Command	Arguments	Return	Purpose/Comments
Basic Commands			
RFdcVersion	None	Version Number(s)	RFdc API version, for example, "v3_1"
Version	None	Version Number(s)	Protocol version, for example, "v1_0"
TermMode	mode	None	Switch between UI mode (mode=0) and Interactive mode (mode=1). Interactive mode prints more status information to the terminal, and should not be used with the Evaluation Tool GUI.
GetLog	None	Logged Strings	Uses metal_log to return any error, warning, or informational messages returned from the API or command interface.
iic_write	iic instance, slave address, register offset, size, data	Success/Fail	Write number of bytes to I2C slave.
iic_read	iic instance, slave address, size	Success/Fail	Read number of bytes from I2C slave.
RFdc API Commands			
SetMixerSettings	Type, Tile, Block, MixerSettings	None	Where MixerSettings is 1:1 with the API: Freq, PhaseOffset, EventSource, FineMixerMode, CoarseMixerFreq, FineMixerScale
GetMixerSettings	Type, Tile, Block	Type, Tile, Block, MixerSettings	Get mixer settings for the requested type, tile, and block.

Table A-1: **Command List *(Cont'd)***

Command	Arguments	Return	Purpose/Comments
GetQMCSettings	Type, Tile, Block	Type, Tile, Block, QMCSettings	Get QMC settings for the requested type, tile, and block.
ShutDown	Type, Tile	None	Shuts down a tile (the tile must be enabled in the bitstream).
StartUp	Type, Tile	None	Starts up a tile (the tile must be enabled in the bitstream).
GetTileStatus	Type, Tile	TileStatus	Returns the TileStatus returned from an XRFdc_IPStatus function call.
GetBlockStatus	Type, Tile, Block	BlockStatus	SamplingFreq, AnalogDataPathStatus, DigitalDataPathStatus, DataPathClocksStatus, IsFIFOFlagsEnabled, IsFIFOFlagsAsserted
GetPLLConfig	Type, Tile	RefClkFreq, SampleRate, RefClkDivider, FeedbackDivider, OutputDivider	Get PLL configuration for the requested type and tile.
DynamicPLLConfig	Type,Tile, Source, Refclkfreq, sampling rate	RefClkDivider, FeedbackDivider, OutputDivider	Configure the PLL sampling rate depending on the source.
GetLinkCoupling	Tile, Block	Tile, Block, Mode	Get link coupling mode for the requested tile and block.
SetCoarseDelaySettings	Type, Tile, Block	Success/Fail	Set Coarse delay settings for the requested type, tile, and block.
GetCoarseDelaySettings	Type, Tile, Block	Type, Tile, Block, CoarseDelay, EventSource	Get coarse delay settings for the requested type, tile, and block.
SetInterpolationFactor	Tile, Block	Success/Fail	Set the interpolation factor for the requested tile and block.
GetInterpolationFactor	Tile, Block	Tile, Block, Interpolationfactor	Get interpolation factor for the requested tile and block.
SetDecimationFactor	Tile, Block	Success/Fail	Set decimation factor for the requested tile and block.
GetDecimationFactor	Tile, Block	Tile, Block, Decimationfactor	Get decimation factor for the requested tile and block.
MultiConverter_Init	Type	Success/Fail	Initializes multi-tile synchronization feature.
MultiConverter_Sync	Type, Latency, Tile	Latency, Offsets, Marker delay, Sysrefenable	Synchronize between tiles.
MTS_Setup	Type, Enable	Success/Fail	Setup resources for multi-tile synchronization feature.

Table A-1: **Command List *(Cont'd)***

Command	Arguments	Return	Purpose/Comments
SetMemtype	Mem_type	Success/Fail	Set memory type to DDR or BRAM.
GetMemtype	None	Mem_type	Get selected memory type.
ZCU111 Board Commands			
SetExtParentclk	board id, frequency	Success/Fail	Configure PLL clock with requested frequency.
SetExtPllClkRate	board id, pll source, frequency	Success/Fail	Configure PLL with requested frequency.
SetDACPowerMode	board id, Tile, Block, output current	Success/Fail	Configure DAC power.
GetDACPower	Board, Tile	Board, Tile, DAC_AVTT, DAC_AVCC_AUX, DAC_AVCC, ADC_AVCC_AUX, ADC_AVCC	Get DAC power mode for the requested board and tile.
Memory Access Commands			
WriteDataToMemory	Tile, Block, number_of_bytes, interleaved_pair	Actual bytes written	In this case, the buffer address is allocated by Linux (CMA pool) from PS DDR and firmware writes the data to the buffer by reading data from a socket.
ReadDataFromMemory	Tile, Block, number_of_bytes, interleaved_pair	Actual bytes read	In this case, the buffer address is allocated by Linux (CMA pool) from PL DDR and firmware reads the data from the buffer and sends it to a socket. In this command, the number of bytes should be aligned to 32.

Example Commands and Responses

A list of example commands and responses follows:

#invalid command name

SetMixerSett 0 1 2 ...

Error: SetMixerSett: Invalid command

#invalid number of arguments

SetMixerSettings 0

Error: SetMixerSettings: Invalid Number of Arguments

valid command and response (no data returned - args: Type, Tile, Block, Freq, PhaseOffset, EventSource, FineMixerMode, CoarseMixerFreq, FineMixerScale)

SetMixerSettings 0 1 2 3.4 0.0 ..

SetMixerSettings:

#valid command and response (data returned - Freq, PhaseOffset, EventSource, FineMixerMode, CoarseMixerFreq, FineMixerScale)

GetMixerSettings 0 1 2

GetMixerSettings: 3.14 ...

Control Path Core Implementation

Structure

The control path core code is implemented in C code and runs on the APU. The main functions are separated into files so the protocol-core is independent from the interface input or output and the commands themselves. This allows easier re-use and extension.

Files:

- **`io_interface (.c/.h)`**
  - This file contains the communication specific code that implements getString/sendString
  - The sub-functions here are used to separate the core (cmd_interface) from the physical communications channel.
  - To port to another communications medium, this is the only file that should be edited.

- **`cmd_interface (.c/.h)`**
  - This is the core of the command protocol. It contains a menu/table of commands and their expected arguments.
  - This menu is implemented as an array of command-structures.
    - The command-structures contain the command name, the expected arguments, the argument number format (long, unsigned, double), and a function-pointer to call if the command matches.
    - Extending to add a new command involves adding a new entry in the table, adding the associated function-pointer, and adding the function itself.

  When a command is received, it is checked for a match in the table, and the number of arguments received is checked against the expected number of arguments from the table. Arguments are parsed from the command string and converted based on the expected data type. The arguments are put into an *array of unions* to allow the same data type to be passed to functions that expect different argument data types.

  The associated wrapper function is called by the function-pointer, and a return value/message string generated.

- **`rfdc_commands (.c/.h)`**
  - This file contains the ADC/DAC wrapper functions that are called from the menu (via the function pointer).
  - Each wrapper function calls the corresponding API function.

Porting

Porting the control path to support a new communications interface involves only editing the io_interface functions. The key functions that must be addressed are:

- GetString
  - This function must return a full command string, which is terminated by **`\n`**.
  - If a string is partially received (i.e., still being received), this function should not block (see the Non-blocking function), and return 0.
  - When a string is fully received, it returns a non-zero number to reflect there is a new command to be parsed.
- SendHW
  - This function is called from SendString which pre-parses a string to remove embedded **`\r`** and **`\n`** characters and append a single **`\r\n`** at the end.
  - SendHW is essentially a print function that actually transmits the information at the end.

    - This function should be pointed to the method used to actually transmit strings or characters over the desired communication interface.
- Non-blocking
    - Implementation of command and datapath is non-blocking.

Appendix B

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

Solution Centers

See the Xilinx Solution Centers for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

Documentation Navigator and Design Hubs

Xilinx® Documentation Navigator provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open the Xilinx Documentation Navigator (DocNav):

- From the Vivado® IDE, select **Help > Documentation and Tutorials**.
- On Windows, select **Start > All Programs > Xilinx Design Tools > DocNav**.
- At the Linux command prompt, enter `docnav`.

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In the Xilinx Documentation Navigator, click the **Design Hubs View** tab.
- On the Xilinx website, see the Design Hubs page.

Note: For more information on Documentation Navigator, see the Documentation Navigator page on the Xilinx website.

References

The most up to date information related to the ZCU111 board and its documentation is available on the following websites.

Zynq UltraScale+ RFSoC ZCU111 Evaluation Kit

ZCU111 Evaluation Kit Master Answer Record 70958

ZCU111 RF Data Converter Evaluation Tool Master Answer Record 71424

These Xilinx documents provide supplemental material useful with this guide:

1. *Zynq UltraScale+ RFSoC Data Sheet: Overview* (DS889)
2. *Zynq UltraScale+ RFSoC Data Sheet: DC and AC Switching Characteristics* (DS926)
3. *Zynq UltraScale+ Device Technical Reference Manual* (UG1085)
4. *ZCU111 RFSoC RF Data Converter Evaluation Tool Getting Started Guide* wiki
5. *ZCU111 UltraScale+ RFSoC ZCU111 Evaluation Kit Quick Start Guide* (XTP490)
6. Vivado Design Suite
7. Xilinx Software Development Kit (XSDK)
8. PetaLinux Tools
9. *Zynq UltraScale+ RFSoC RF Data Converter LogiCORE IP Product Guide* (PG269)
10. *RF Data Converter Interface User Guide* (UG1309)
11. *Zynq UltraScale+ MPSoC Software Developer Guide* (UG1137)
12. C*oherent Sampling vs. Window Sampling* application note from Maxim Integrated (Tutorial 1040)
13. *PCB Design Guidelines for Zynq UltraScale+ RFSoCs User Guide* (UG582) (Registration Required)
14. *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
15. *Vivado Design Suite User Guide: Designing with IP* (UG896)
16. *Vivado Design Suite User Guide: Getting Started* (UG910)
17. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
18. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
19. *Zynq UltraScale+ Device Packaging and Pinouts User Guide* (UG1075)
20. *Zynq UltraScale+ MPSoC: Embedded Design Tutorial* (UG1209)
21. *ZCU111 Evaluation Board User Guide* (UG1271)

22. *AXI4-Stream Infrastructure IP Suite, LogiCORE IP Product Guide* (PG085)

Please Read: Important Legal Notices