Zynq UltraScale+ MPSoC Software Acceleration Targeted Reference Design

User Guide

UG1211 (v2016.3) January 30, 2017

Revision History

The following table shows the revision history for this document.

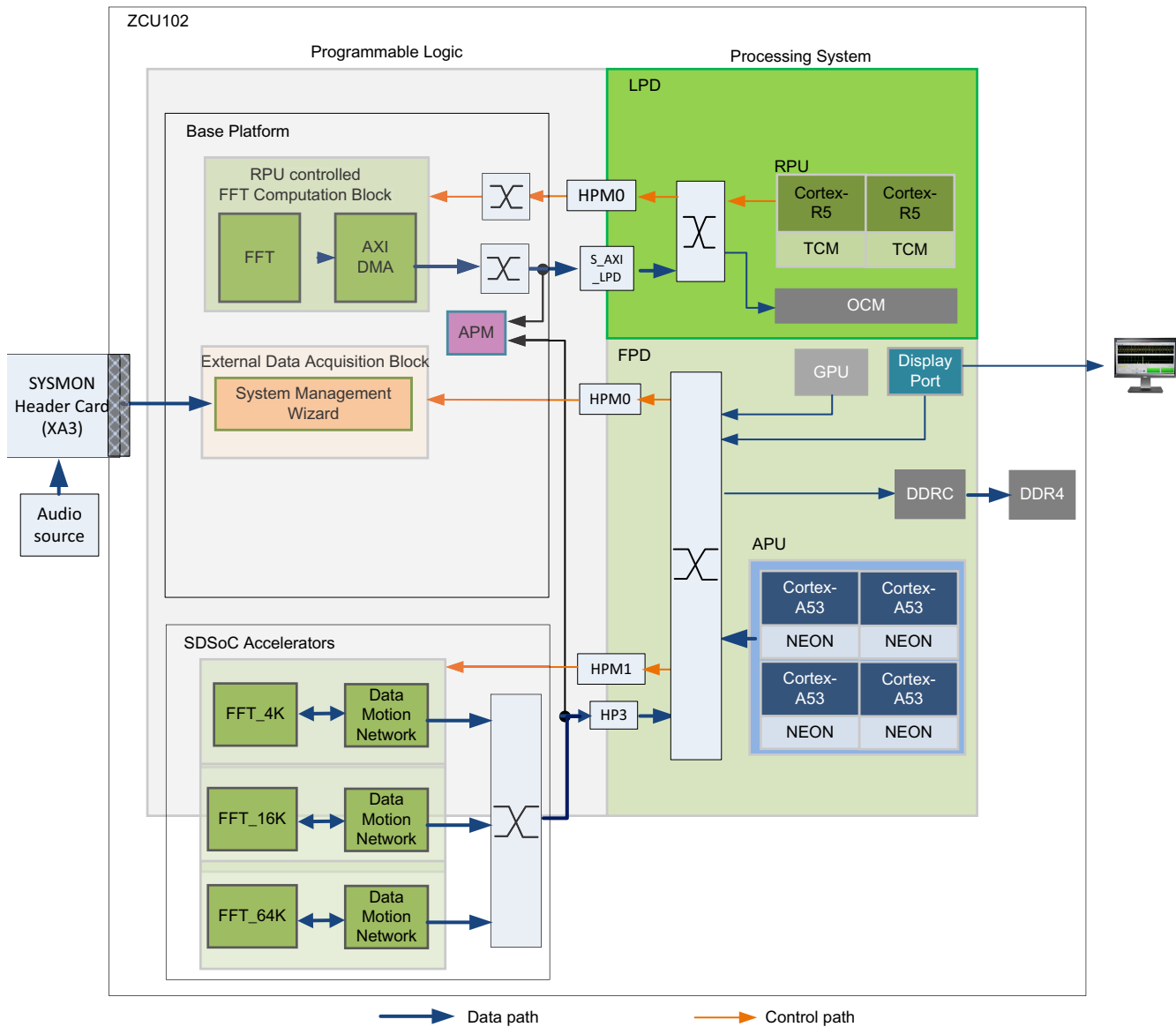| Date	Version	Revision
01/30/2017	2016.3	Vivado Design Suite was removed from the title page, because the design aligns with SDx releases. Removed hardware test pattern generator information from text and from Figure 1-1, Figure 2-9, and Figure 2-10. Updated Table 1-1. Added Software Test Pattern Generator and updated Software Architecture. Updated Linux attributes of Table 2-2. Updated Table 2-4. Updated Figure 2-3, Figure 2-5, and Figure 2-7.
08/30/2016	2016.2	Updated for Vivado Design Suite 2016.2. Updated Features. Changed supported FFT sizes. Updated Figure 1-1, Figure 2-1, Figure 2-4, Figure 2-5, Figure 2-8, Figure 2-9, and Figure 2-10. Updated Table 1-1, Resource Utilization. Updated Amplitude-Modulated Pattern Generator. User Interface—Qt-based GUI, SDSoC Hardware Accelerators, RPU-Controlled FFT Computation Block, Clocking, Resets, and Interrupts sections. Updated Table 2-2, Linux Kernel . . . row, Flow C. Updated Table 2-3 and Table 2-4. Added SDSoC resources to Appendix B, Additional Resources and Legal Notices **Limited distribution.**
05/18/2016	2016.1	Initial Xilinx release. **Limited distribution.**

Table of Contents

Send Feedback

Introduction

This document describes the features and functions of the Zynq® UltraScale+™ Software Acceleration targeted reference design (TRD) for the ZCU102 evaluation platform. The Software Acceleration TRD is an embedded signal processing application that is partitioned between the SoC processing system (PS) and programmable logic (PL) for optimal performance. The design demonstrates the value of offloading computation-intensive tasks such as Fourier Fast Transform (FFT) from PS to PL.

Overview

Figure 1-1 shows the functional block diagram of the Software Acceleration TRD.

Figure 1-1: **ZCU102 Software Acceleration Targeted Reference Design**

Note: The processing system (PS) block shown in Figure 1-1 is a simplified block for illustration purposes.

Send Feedback

The Software Acceleration TRD targets the Zynq UltraScale+ XCZU9EG-1-FFVB1156E MPSoC running on the ZCU102 evaluation board. The design is intended to demonstrate software acceleration and offloading. Acceleration is achieved by offloading the computationally intensive algorithm to the programmable logic (PL). This design demonstrates 4,096 (shown as FFT_4K in Figure 1-1), 16,384 (FFT_16K), and 65,536 (FFT_64K) point FFT transform sizes, but it can also be used as a basis for many software acceleration tasks.

Features

The TRD includes these features:

- Signal capturing capability (data acquisition)
    - External analog input capture through SysMon in PL (A-to-D converter).
    - PL-SysMon runs at a 10-bit resolution and at a maximum sample rate of 200 kilosamples per second (kSPS).
- Software-based Test Pattern Generator (TPG)
- Signal Processing capability—FFT is used as the signal processing algorithm. Run-time configurability of FFT size shows signal processing capability. These options for FFT compute engines are provided:
    - ARM® Cortex™-A53 processor alone (SMP Linux on Application Processing Unit (APU))
    - ARM Cortex-A53 processor with NEON instruction set
    - ARM Cortex-R5 real-time processing unit (RPU). (RPU is used as a coprocessor.)
    - Software-defined system-on-chip (SDSoC) based hardware accelerators—the APU-controlled PL accelerators are hardware accelerators in PL built using the SDSoC framework.
    - The RPU-controlled FFT computation block in PL using the LogiCORE FFT IP from the Vivado IP catalog

    Note: In this document, ARM Cortex-A53, A53, and APU are used interchangeably. Also ARM Cortex-R5, R5, and RPU are used interchangeably.

- Open Asymmetric Multi-Processing (OpenAMP) framework [Ref 1]
    - Demonstrates communication between the APU running Linux and the RPU (in lock-step) running FreeRTOS [Ref 2]
- Display capability
    - Uses the Display Port subsystem in the PS for displaying the results/GUI.

- The graphical user interface (GUI):
    - Is based on Qt 5.7 [Ref 3].
    - Uses OpenGL and EGLFS (ARM Mali 400 GPU) libraries for rendering.

Contents of the TRD

The reference design includes an SDSoC-based hardware/software platform that can be used as a starting point for implementing custom embedded signal processing applications. It also contains an example application that implements a Radix-2 Cooley-Tukey Decimation in Time (DIT) FFT algorithm optimized for hardware acceleration using the SDSoC design flow.

IMPORTANT: *The Zynq UltraScale+ MPSoC Software Acceleration TRD 2016.3 wiki contains additional helpful information and a link to the targeted reference design ZIP file* `rdf0376-zcu102-swaccel-trd-2016-3.zip.`

The reference design ZIP file contains:

- An SDSoC hardware platform, based on the Vivado IP Integrator hardware project. It provides a pre-built design with external signal capture interface and RPU-controlled PL accelerator blocks instantiated.
- SDSoC software platform for APU host. It provides a complete Linux operating system including all required drivers and middleware layer.
- SDSoC software example for APU host. It includes a programmable size FFT algorithm that can be linked with the main application.
- XSDK projects for user applications and libraries
- APU main application and libraries
    - PetaLinux board support package (BSP) for the ZCU102 evaluation board.
    - Pre-built, bootable SD card image.

The wiki contains information on:

- Setup and bring-up of the TRD demonstration
- TRD package contents and directory structure
- Steps to run and test the TRD
- Instructions for building various components of the TRD

Resource Utilization

Table 1-1 lists the resources used by the ZCU102 Software Acceleration TRD after synthesis and before place and route. Place and route can alter these numbers based on placements and routing paths, so these numbers are to be used as a rough estimate of resource utilization. These numbers might vary based on the version of the design and the tools used to regenerate the design.

Table 1-1: **Resource Utilization**

Resource	Total Available	Usage	Percentage%
CLB Registers	548,160	105,870	19.31
CLB LUT	274,080	84,600	30.86
Block RAM	912	496	54.38
BUFG_GT	168	1	0.60
SYSMONE4	1	1	100
IOB	328	4	1.22
DSP48E2	2,520	334	13.25

Targeted Reference Design Details

This chapter describes the TRD hardware and software components in detail.
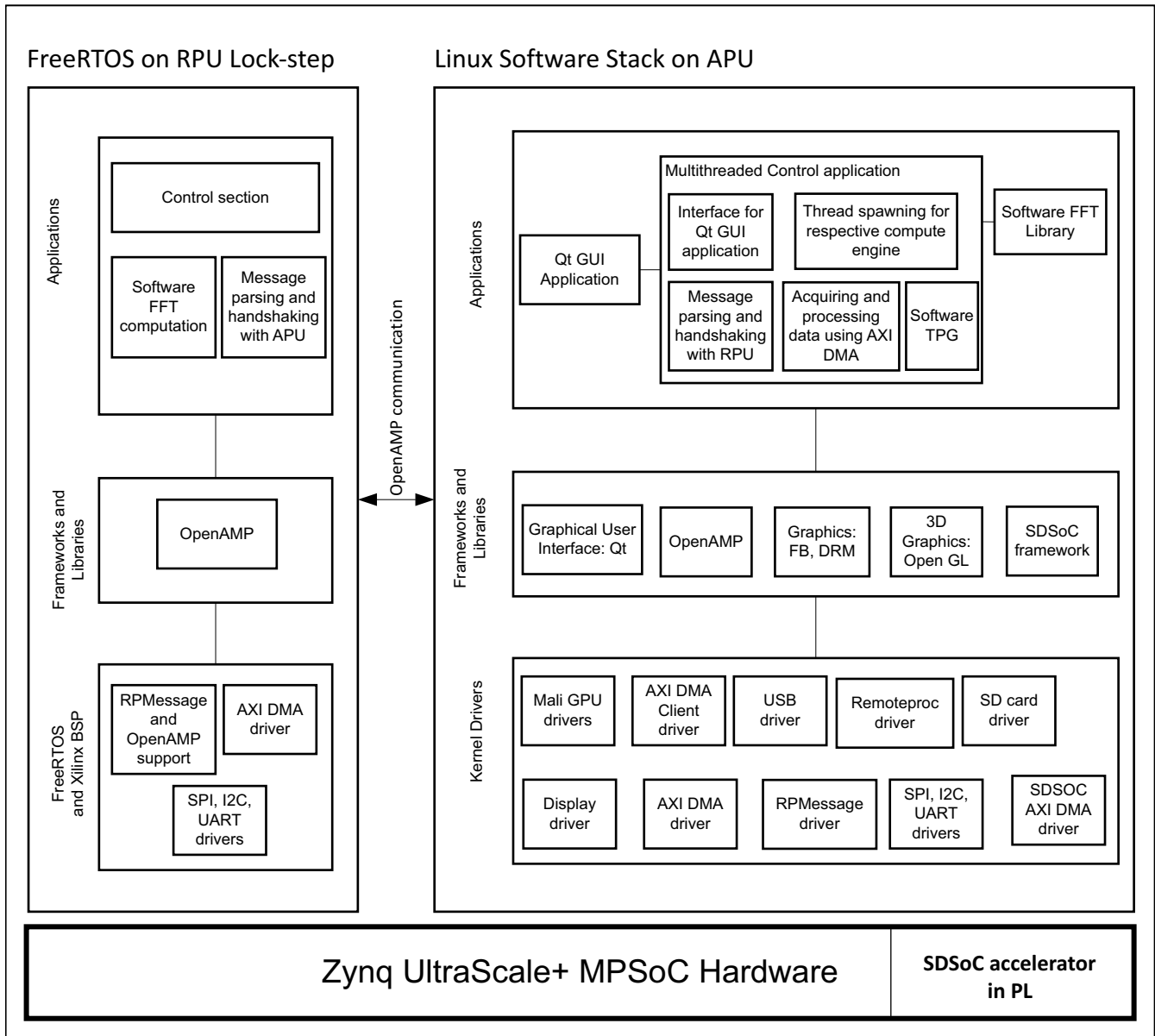
Software

This section covers the software architecture in detail.

Software Architecture

The core software components of the design are:

- Symmetric multi-processing (SMP) Linux running on APU
- FreeRTOS running on RPU
- User Interface — a Qt based GUI
- Application Library
- Device drivers (for Linux and FreeRTOS)
- OpenAMP framework
- FFT algorithm
- Test pattern generator

Figure 2-1 shows a high level software block diagram.

X16983-122016

Figure 2-1: **High Level Software Block Diagram**

Send Feedback

The flows listed in Table 2-1 are supported in the TRD.

Table 2-1: **Flows Supported in the TRD**

Flow	Description
Flow A	FFT computation in APU running Linux in symmetric multi-processing (SMP) mode
Flow B	FFT computation in APU with NEON as coprocessor
Flow C	FFT computation in SDSoC-generated PL accelerator
Flow D	FFT computation in RPU
Flow E	FFT computation in PL controlled by RPU

Note: In all flows described in Table 2-1, the data movement of input samples to the relevant FFT compute engine is controlled by software running in the APU.

All the above components are discussed as per the specific design flow.

User Interface—Qt-based GUI

The GUI for this TRD is based on Qt 5.7, which uses OpenGL libraries and the EGLFS platform framework for rendering a GUI through the graphics processing unit (GPU) to the Display Port.

The Qt application is statically linked to Application Library. Select various configurable parameters through the user interface, and the Application Library configures the flow accordingly.

The Qt GUI provides the following controls:

• Test start/stop control—Control to start and stop the test.

• Input source selection—There are two sources of inputs, internal TPG and external analog input

• FFT size—Supported sizes are:

  ◦ 4,096

  ◦ 16,384

  ◦ 65,536

• FFT computation resource—Options supported:

  ◦ APU running Linux in SMP mode

  ◦ APU and NEON as coprocessor

  ◦ APU-controlled PL accelerator

  ◦ RPU (data controlled by APU)

  ◦ RPU-controlled PL accelerator

Send Feedback

- Windowing type—Options supported:
    - Hann
    - Hamming
    - Blackman
    - Blackman-Harris
- Frequency plot voltage scale—Options supported:
    - 0.1V
    - 0.25V
    - 0.5V
    - 1V
- Sampling rate

    The following options are supported:
    - 50 kSPS
    - 100 kSPS
    - 200 kSPS

The Qt GUI provides the following plots:

- Time domain representation of the input source
- Frequency domain representation of the input after FFT compute
- CPU utilization for the APU (A53 cluster)
- FFT computation time
- Performance plots of PS-PL interfaces used in the design
- PL die temperature display

Application Library

The Application Library is linked statically to the Qt-based GUI. It is responsible for the following:

- Getting control inputs from the GUI
- Moving data from input source to compute engine
- Performing the FFT operation using the compute engine you select
- Providing FFT computation time, PS-PL throughput, and PL die temperature information to the GUI

Send Feedback

Operating Systems

In this TRD, two operating systems are run:

• APU (Master) - Running Linux in SMP mode on A53 cluster

• RPU (Slave) - Running FreeRTOS

APU Linux in SMP Mode

After Linux is booted, it invokes the Qt GUI linked to the Application Library. The Qt GUI and the Application Library perform functions as mentioned above.

The AXI-DMA driver on the APU conforms to the Linux DMA framework. It registers itself as a DMA controller for each instance of AXI-DMA IP according to the information present in the device tree. To enable registration of a DMA client driver with the required channels, proper entries mentioning each instance of the AXI-DMA channel of interest have to be included in the device tree. One such DMA client driver which registers itself with all the required channels of multiple AXI-DMA instances has been developed. The DMA client driver exposes a character driver interface through which user applications can initiate stream to memory-mapped (S2MM) and memory-mapped to stream (MM2S) transfers through write and read calls.

After getting the channel allocated, the client driver can initiate DMA transfers. The client DMA transfer requests to the AXI-DMA driver involved in creating a Scatter Gather (SG) list (a representation of different physical pages of the user space buffer), and submits it for read/write operations. This buffer (SG list) is cache-flushed and invalidated as required before submitting the list to the AXI-DMA driver through the framework-supplied APIs. The user space application can poll for the Completion status of these DMA transfers in an ioctl call.

FreeRTOS on RPU

After Linux in APU is booted, it loads the RPU firmware through OpenAMP. FreeRTOS-based RPU firmware executes out of tightly coupled memory (TCM) in Lock-step mode.

The AXI-DMA driver developed for FreeRTOS running on RPU works with the flat memory module because there is no memory management unit (MMU) providing address virtualization. The FFT application directly accesses its APIs without requiring another DMA client driver.

OpenAMP Framework

OpenAMP framework provides a master-slave communication channel between the Linux operating system on APU and the FreeRTOS operating system on RPU.

OpenAMP communication for this TRD is between the Linux master and the FreeRTOS slave performing the FFT task either by software or by the PL FFT module (controlled by the RPU application). The flow is described in Appendix A, OpenAMP Communication.

The Xilinx OpenAMP solution provides fully functional remoteproc and RPMsg components usable for a Linux master, FreeRTOS remote configuration.

- **Remoteproc:** This is used to control the life cycle management (LCM) of the remote processors from the master processor. The remoteproc API that is provided with OpenAMP is compliant with the infrastructure present in the upstream Linux kernel 3.18 onwards. The remoteproc uses information published through the firmware resource table to allocate system resources and to create virtIO devices.

- **RPMsg**: The RPMsg API allows interprocess communications (IPC) between software running on independent cores in an AMP system. This is also compliant with the RPMsg bus infrastructure present in the upstream Linux kernel, version 3.18 onward.

The remote application is loaded as firmware by Linux after bootup using remoteproc API(s). The remote application for FreeRTOS is developed using the xilopenamp library, and this library can be included using the Xilinx SDK.

The general OpenAMP flow is as follows:

1. The Linux master configures the remote processor and creates shared memory.

2. The master boots the remote processor.

3. The remote processor calls remoteproc_resource_init(), which creates the virtIO and RPMsg channels and informs the master.

4. The master receives this information and invokes the created callback channel.

5. The master responds to the remote context, acknowledging the remote processor and application.

6. After acknowledgment, the remote processor invokes the RPMsg channel that was registered.

7. The RPMsg channel is now established, and both sides can use the RPMsg calls to communicate.

To shut down the remote processor:

- The master application sends an application-specific shutdown message to the remote application.

- The remote application cleans up resources on the remote side and sends an acknowledgment to the master.

- The remote application calls the remoteproc_resource_deinit() function to free the remoteproc resources on the remote side.

- The master application shuts down the remote processor and frees the remoteproc on the side of the master.

- A typical slave (FreeRTOS) side initial sequence looks like this:

  ◦ Initialize the communication channel by calling remote_proc_resource_init. Provide a resource table and interprocessor interrupt (IPI) information. In this API, the callbacks are registered for channel create, channel delete, and read.

- After this channel is created, it is called by the framework after the channel is established.

Shared Memory Usage

The OpenAMP framework requires shared memory (PS DDR) that is reserved in the device tree. The shared memory is used by RPMsg to pass messages between the processors. The amount of memory is determined by the number of communication buffers multiplied by the buffer size. These are user-configurable, but in this design, 256 buffers of 512 bytes each are reserved in PS DDR and used.

Configuring the RPU

For this TRD, RPU is configured in Lock-step mode, that is, both R5_0 and R5_1 cores execute the same instructions in parallel (the option is selected in device tree entry in Linux). A Linux driver loads the RPU firmware using the remoteproc driver.

Pointer Passing Mechanism

Each RPMsg buffer is 512 bytes. This limits the amount of data being passed between the master and slave per message. To avoid this limitation, the application copies the input data buffers from the PS DDR to on-chip memory (OCM). The address in the OCM where buffers are copied to is stored in the PS DDR and that pointer is passed to RPU using RPMsg. This reduces the amount of data to be passed on the RPMsg channel.

Software Test Pattern Generator

The software test pattern generator (TPG) generates samples corresponding to a frequency or amplitude-modulated waveform. The samples are written into the PS DDR for processing.

Software Algorithm for FFT

The ARM NE10 project provides an ARM-optimized FFT computation library (open source code). This code is well tested and can be picked and used by other applications [Ref 8].

FFT computation in software can be done by using either of the following NE10 APIs for this TRD:

- ARM mode computation of FFT using ne10_fft_c2c_1d_float32_c0 API

- NEON mode computation of FFT using ne10_fft_c2c_1d_float32_neon0 API. This uses NEON intrinsic APIs for FFT computation.

   Note: When FFT is computed in RPU, NEON-specific APIs are not used. Instead, only the ARM mode computation API (ne10_fft_c2c_1d_float32_c()) is used.

There is a dynamic memory allocation requirement for this algorithm. The dynamic allocation is on the order of eight times the length of FFT.

The size of each input sample from PL is 4 bytes (real part). Four bytes for the imaginary part is appended to each sample.

- The size of one complex sample = 8 bytes

- Dynamic memory allocation =

   ◦ size_of_one _complex_sample x FFT length (buffer size)

   ◦ + 8 bytes x FFT length (twiddle factor)

   ◦ + other bookkeeping structure elements (312 bytes)

- For a 4,096 point FFT, see Equation 2-1.

$$\text{Dynamic memory allocation} = 8 \times 4{,}096 + 8 \times 4{,}096 + 312 = 65{,}848 \text{ bytes} \qquad \textit{Equation 2-1}$$

This requirement for dynamic allocation limits the max FFT length supported in some flows where the available execution memory is limited.

Memory Requirement

There are two kinds of memory required for this TRD:

- Memory needed for input and output data buffers of the FFT.

- Memory for the FFT computations if the software algorithm is used

Memory Allocation

The memory is allocated from DDR, OCM, and TCM per the requirement of the flow:

- DDR memory is allocated using the malloc calls from Linux applications running on the APU.

- TCM memory is allocated using the malloc calls from FreeRTOS applications running on the RPU.

- OCM memory is accessed using mmap from the host Linux application running on the APU.
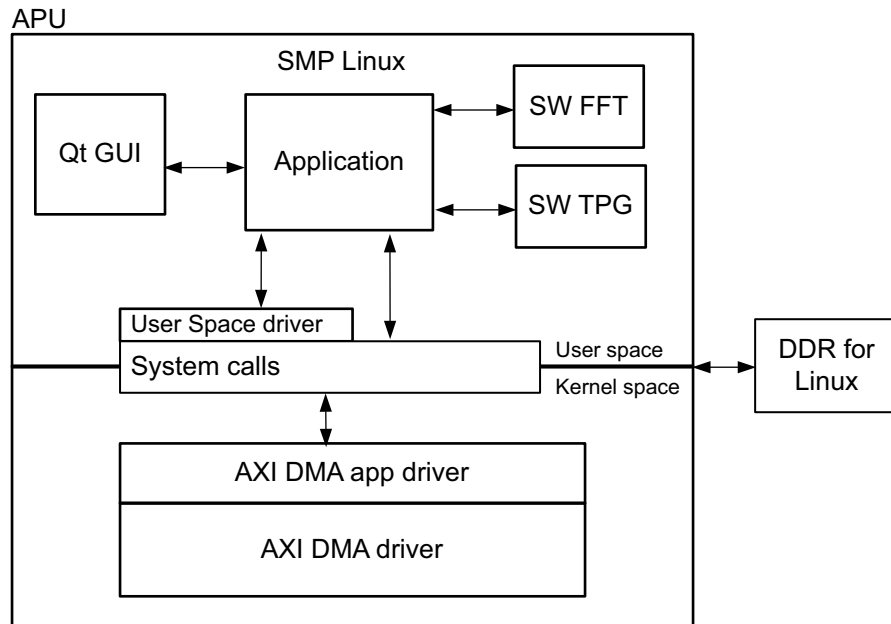
FFT Computation Flows

This section describes software aspects in terms of the operating system, memories, frameworks, libraries, processing engine, and latencies involved in various FFT computation flows described in the beginning of the Software Architecture section.

Table 2-2 describes the various parameters for each flow.

Table 2-2: **Attributes for Each Flow**

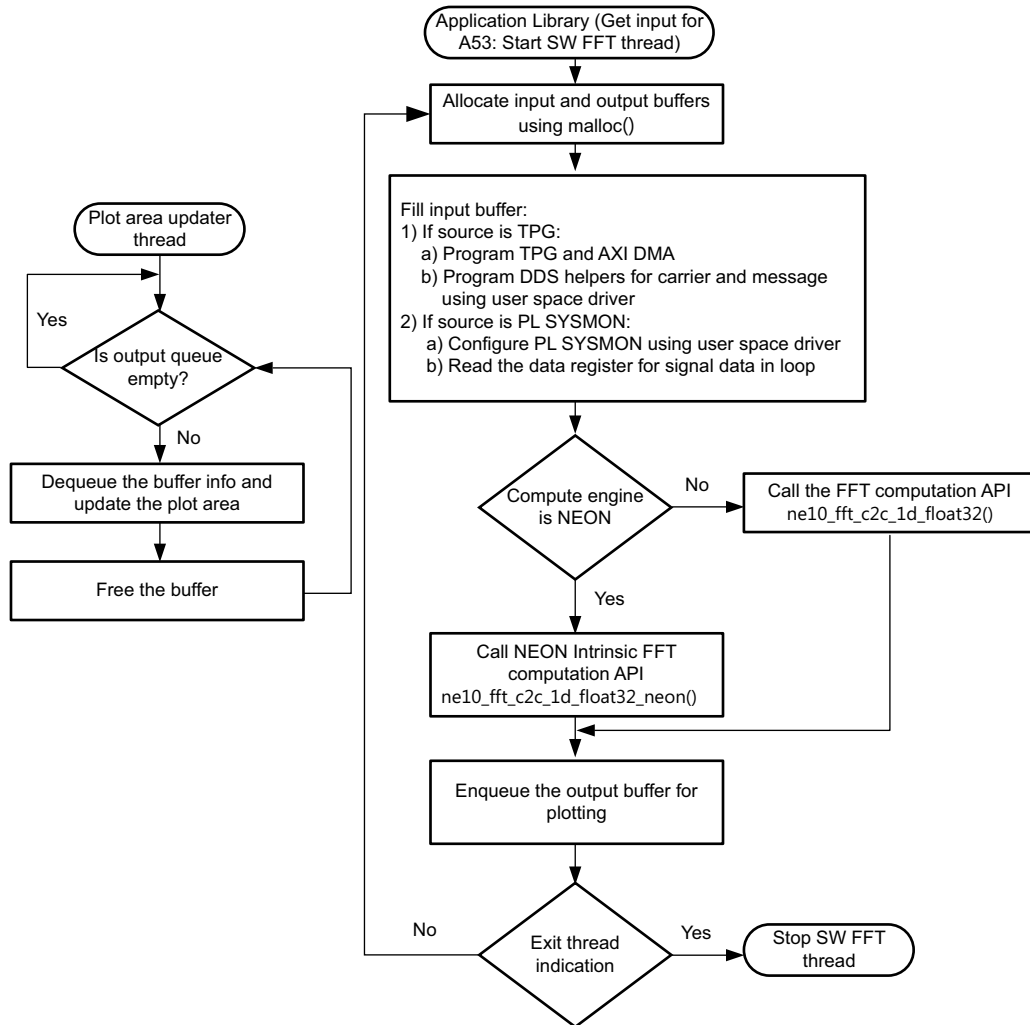| Attribute	Flow A	Flow B	Flow C	Flow D	Flow E
Application Library and GUI run on	Linux SMP on APU
FFT compute engine	APU (SW)	NEON (SW)	PL (accelerator controlled by APU)	RPU (SW)	PL (accelerator controlled by RPU)
Memory used by compute engine to read input samples	DDR	DDR	DDR	OCM	OCM
Memory for FFT algorithm computation	DDR	DDR	PL-BRAM	TCM	PL-BRAM
Memory used by compute engine to write output samples	DDR	DDR	DDR	OCM	OCM
FFT sizes supported	4,096 16,384 65,536	4,096 16,384 65,536	4,096 16,384 65,536	4,096	4,096
Interface between user application and FFT engine	Statically linked software library	Statically linked software library	AXI-DMA (using Linux driver)	OpenAMP	OpenAMP + AXI-DMA (using FreeRTOS driver on RPU)
Linux Kernel space drivers configure and manage buffers of:			SDSoC AXI DMA driver		
Linux user space drivers configure:	PL-SysMon and FFT helper				

Figure 2-2 and Figure 2-3 show the block diagram and flow chart of the software architecture for Flow A and B, respectively.

X16897-122016

Figure 2-2: **Block Diagram of the Software Architecture for Flow A and B**

Figure 2-3: **Block Diagram of the Flow Chart of Software Architecture for Flow A and B**

Note: The FFT algorithm used in Flows A and B is from the ARM NE10 project [Ref 8].

Figure 2-4 and Figure 2-5 show the block diagram and flow chart of the software architecture for Flow C, respectively.

Note: The FFT algorithm used in Flow C is a custom high-level synthesis (HLS) optimized C code (developed by Xilinx), different from the algorithm used in Flows A and B.
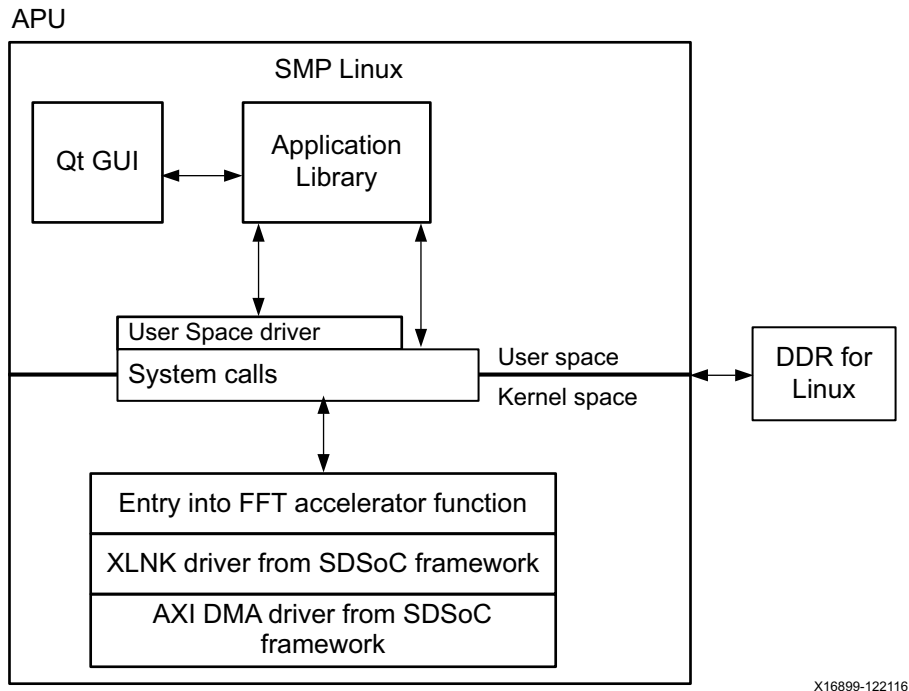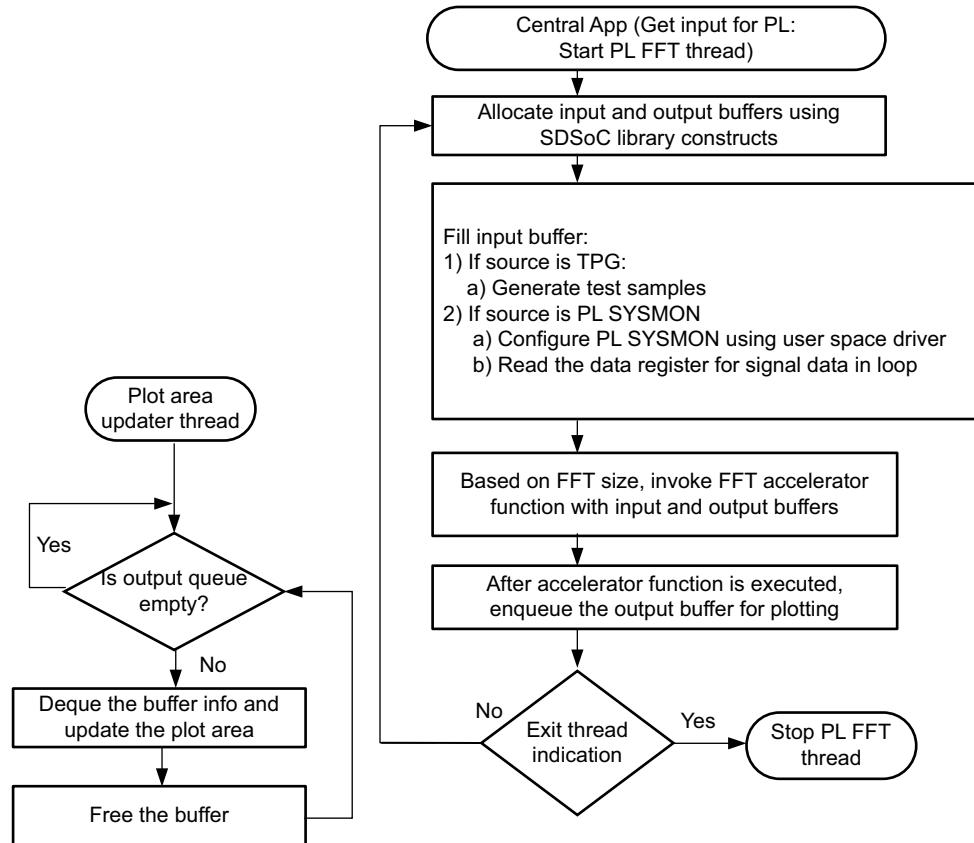

Figure 2-4: **Block Diagram of the Software Architecture for Flow C**

X18611-122116

Figure 2-5: **Block Diagram of the Flow Chart of Software Architecture for Flow C**

For Flow D, the software FFT algorithm is based on the NE10 project and compiled for ARM R5. Non-NEON implementation of NE10 FFT computation is compiled for R5.
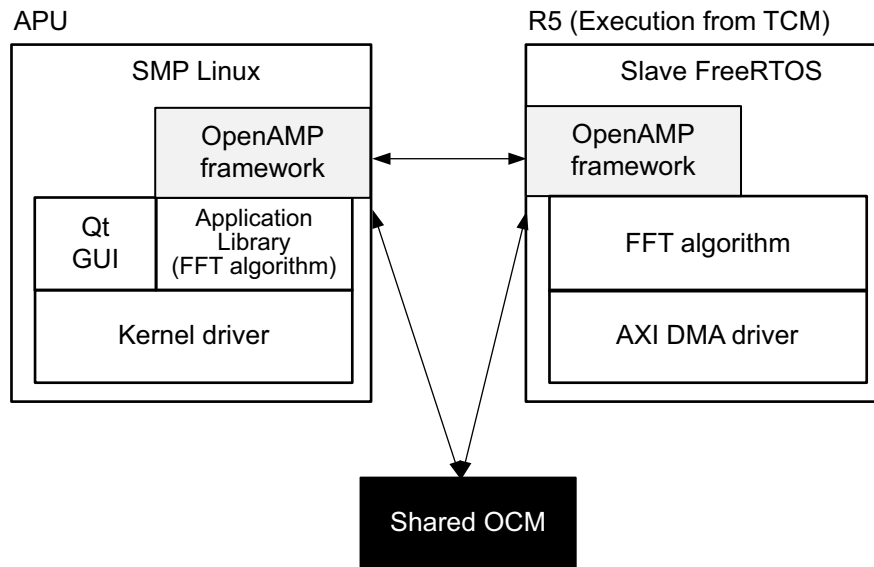
As described in the Software Algorithm for FFT section, the computation algorithm needs a dynamic allocation of the memory, which is of the order of 16 times the FFT size.

As R5 is executing from TCM, which is 256 KB, there cannot be more than 256 KB of memory, out of which 125 KB is needed for self-code. So the maximum dynamic memory that can be allocated is less than 128 KB.

Due to this limitation, the maximum FFT size for this mode is 4 KB.
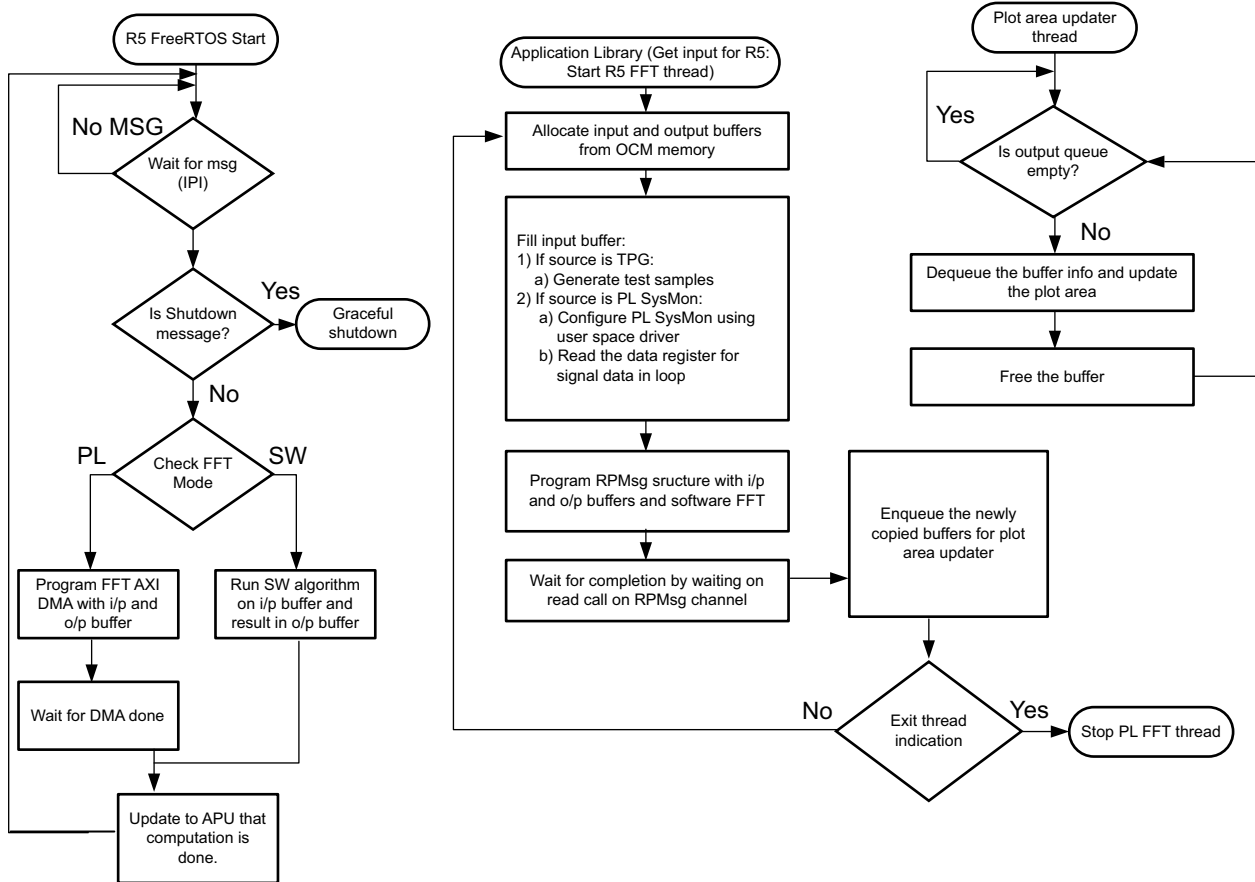
Note: The interface between the Application Library and RPU as FFT Engine is over OpenAMP and covered in Appendix A, OpenAMP Communication.

Figure 2-6 and Figure 2-7 show the block diagram and flow chart of the software architecture for Flow D and E, respectively.

Figure 2-6: **Block Diagram of the Software Architecture for Flow D and E**

X16902-122016

Figure 2-7: **Block Diagram of the Flow Chart of the Software Architecture for Flow D and E**

All FFT Compute Engines Together

This is a special mode where one set of input samples is processed on various FFT engines to compare latencies among each of the flows. The computation is done one compute engine after another per the defined flow charts, but not simultaneously. It is maintained and synchronized by the Application Library.

Latency Computation

Latency is computed in µs using the *gettimeofday* Linux API. Latency or time required for any FFT computational flow is obtained by adding the following:

• Time consumed to fetch input from memory

• Time consumed by the processing engine to compute the FFT.

• Time consumed to load the result to the output buffer in the memory.

Note: Table 2-4 in the Conclusion presents the latency information for the supported FFT sizes.

Hardware Platform

This section describes internal hardware design details of the Software Acceleration TRD.

Figure 1-1 shows a block diagram of the design and the hardware components involved in the design. The following sections describe each of these components in detail.
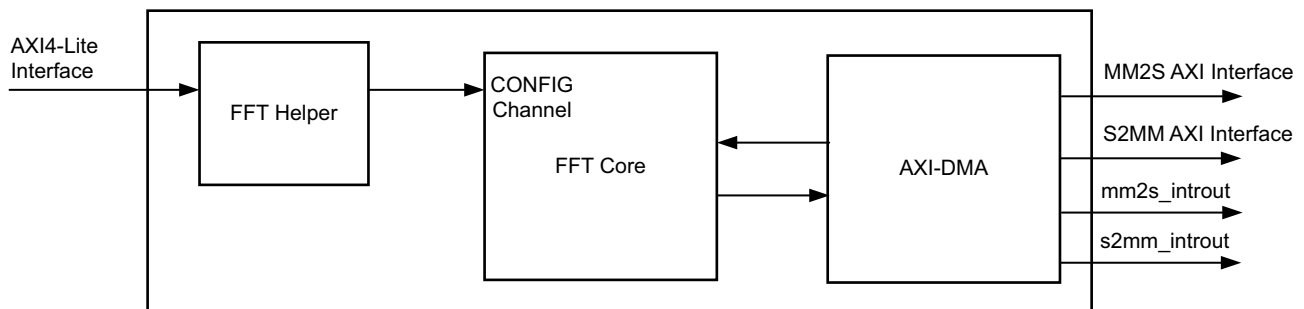
SDSoC Hardware Accelerators

The FFT computation blocks are entirely generated by the SDSoC tool based on a C-code description. The FFT functions are translated to RTL using the Vivado HLS compiler. The data motion network used to transfer samples to or from memory is inferred automatically by the SDSoC compiler.

The data motion network consists of the AXI DMA engine. The AXI-DMA engines are connected to the S_AXI_HP3_FPD high performance PS-PL interface through an AXI interconnect. For AXI4-Lite control, a dedicated PS-PL interface, M_AXI_HPM1_FPD is used. M_AXI_HPM0_FPD is used by the base platform.

RPU-Controlled FFT Computation Block

The FFT computation block in PL is connected to the low power domain master (M_AXI_HPM0_LPD) and slave (S_AXI_LPD) interfaces, and the data transfers are controlled by the software running in the RPU. The S_AXI_LPD port is used for data transfers, and the M_AXI_HPM0_LPD port is used for configuring the AXI slaves (control path).

The FFT computation logic consists of three blocks, namely FFT core, FFT helper, and AXI-DMA. See Figure 2-8 for components of the FFT computation block.

Figure 2-8: **FFT Computation Block**

FFT Core

The FFT core is configured to compute N-point forward Discrete Fourier Transform (DFT), where N can be 2^m and m = 3 to 16. The point size N of the Fourier transform is run-time configurable.

The FFT core uses on-chip block RAM to store the data samples. The FFT core is used in Pipelined Streaming I/O mode, which allows for continuous data processing. In this mode, each processing engine has its own memory banks to store the input and intermediate data. The core has the ability to simultaneously perform transform calculations on the current frame of data, load input data for the next frame of data, and unload the results of the previous frame.

See *Fast Fourier Transform LogiCORE IP Product Guide* (PG109) [Ref 5] for more details on the operation of the FFT module.

FFT Helper

The CONFIG Channel of the FFT IP core provides an AXI streaming interface. To change the run-time programmable parameters of the core (like FFT point size), the configuration channel needs to be driven as described in *Fast Fourier Transform LogiCORE IP Product Guide* (PG109) [Ref 5] depending on the value to be programmed. The FFT helper module translates the control communication from the APU using the AXI4-Lite interface to the configuration channel as required by the FFT core.

AXI-DMA

The AXI-DMA core is used as a data mover between system memory (PS DDR) and FFT core in the PL fabric. The AXI-DMA IP is configured to run both memory-mapped to stream (MM2S) and stream to memory-mapped (S2MM) channels. The AXI-DMA instance used in the APU-controlled PL accelerator block is configured in Scatter Gather mode and the AXI-DMA instance used in the RPU-controlled PL accelerator block is configured in Simple mode. The software configures the AXI-DMA core with information such as where to fetch the data from memory (MM2S path) and where to write the data back in memory after FFT computation (S2MM path). The AXI-DMA core fetches the data samples from memory and presents them on the streaming interface. The FFT core loads these data samples into its memory, computes the FFT, and presents the result of the FFT algorithm on the output streaming interface. The AXI-DMA writes the data samples back to system memory in the S2MM direction.

Note: The FFT core takes only complex input and gives out complex output. The data sourced by the TPG is always real. Software appends an imaginary part (32 bits tied to 0) to the data received from the TPG and makes it a 64-bit complex sample. The samples fed to the FFT core have an imaginary part of zeros.

Send Feedback

External Data Acquisition Block

The external data acquisition block is configured by a System Management wizard that instantiates the System Monitor block. The System Monitor is a 10-bit 200 kSPS analog-to-digital converter (ADC) and has on-chip sensors for sensing die temperature and power supply voltages. The SysMon block can take up to 17 external analog inputs (1 dedicated and 16 auxiliary analog inputs). In this design, only one external analog input is used, and that is connected to dedicated differential analog input (VP/VN) of the ADC. The ADC conversion data is stored in dedicated status registers. The System Monitor block provides AXI4-Lite and dynamic reconfiguration port (DRP) interfaces through which the internal control and status registers can be accessed. The digital samples after ADC conversion are read by the software running in the APU.

Note: The ADC always produces a 16-bit conversion result, and the full 16-bit result is stored in the 16-bit status registers. The 10-bit transfer functions correspond to the 10 most significant bits (MSBs)—or left-most bits—in the 16-bit status registers.

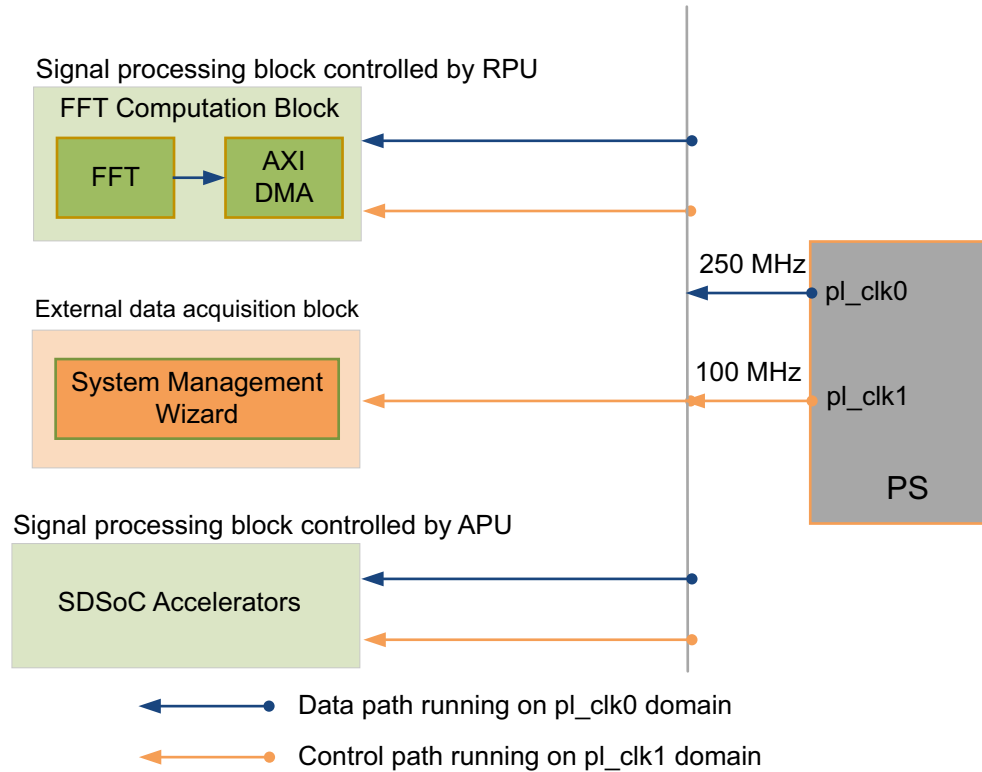In the TRD, SysMon is used in unipolar mode with continuous sampling.

The sampling rate can be changed by writing into the control registers of the core using the AXI4-Lite interface. The maximum frequency of the input analog signal that the PL-SysMon block can sample varies according to the sampling rate of the PL-SysMon.

See *System Management Wizard LogiCORE IP Product Guide* (PG185) [Ref 6] for more details on the System Management wizard and *UltraScale Architecture System Monitor* (UG580) [Ref 7] for a detailed description of the UltraScale SysMon block.

The PL-SysMon is accessed by the software running on the APU over the AXI4-Lite interface. The PS reads the samples from the SysMon after every end of conversion by polling the end of conversion (EOC) bit in the status register.

Clocking

The clocking scheme used in the design is shown in Figure 2-9. The clocks pl_clk0 (250 MHz) and pl_clk1 (100 MHz) from PS are used in the design.

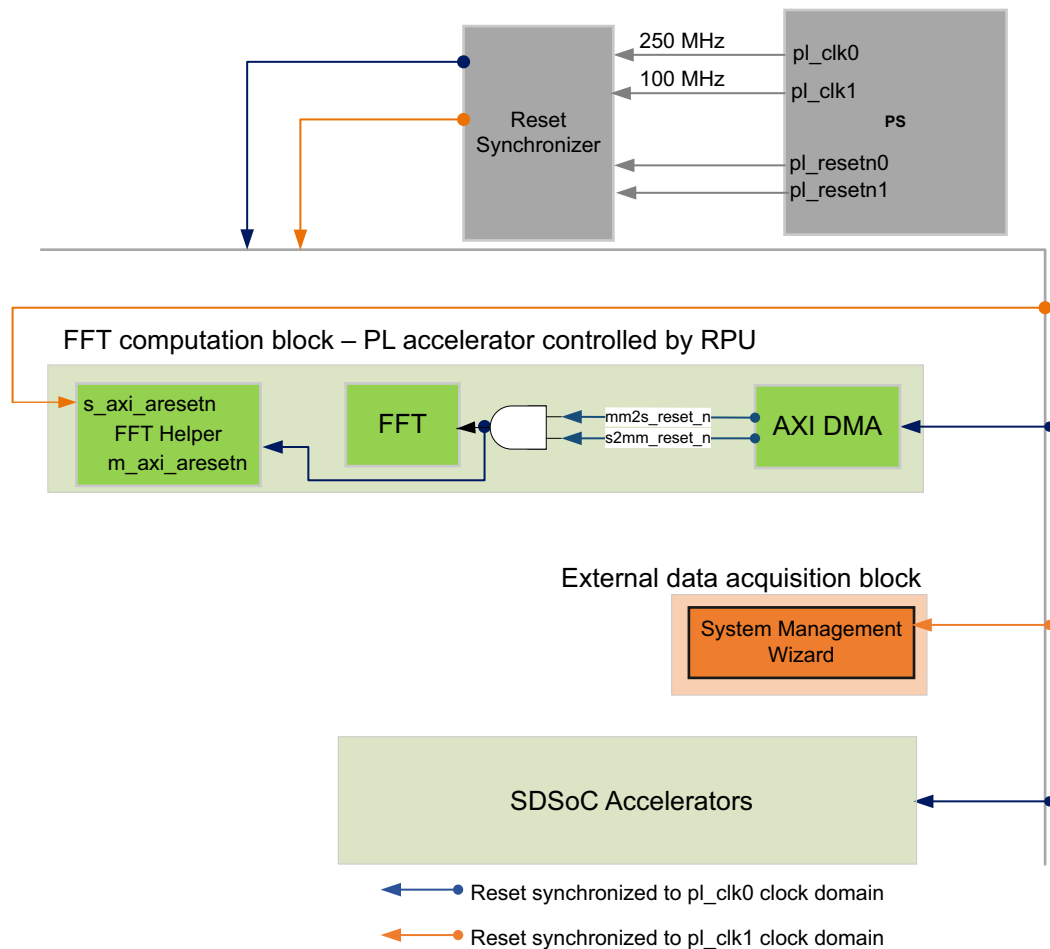*Figure 2-9:* **Clocking Mechanism Used in the Design**

Resets

The extended multiplexed input/output (EMIO) general purpose input/output (GPIO) ports (pl_resetn0 and pl_resetn1) of the PS is used as resets for the PL. The reset pl_resetn0 is synchronized (by using Processor System Reset IP) with respect to the 250 MHz clock and 100 MHz clock domains as shown in Figure 2-10. The S2MM and MM2S channels of the AXI-DMA core can be issued a soft reset (by writing into the AXI-DMA channel control register), in which case the blocks connected to it also get reset.

Interrupts

There are multiple sources of interrupts in the programmable logic (see Table 2-3). All the interrupts are concatenated and connected to the PL_PS_IRQ0 (PL to PS interrupts) interface of the PS.

Table 2-3: **Interrupt Sources and Their IRQ IDs**

Source	IRQ ID
MM2S channel interrupt of RPU-controlled PL accelerator	90
S2MM channel interrupt of RPU-controlled PL accelerator	91
MM2S channel interrupt of APU-controlled SDSoC hardware accelerator	93
MM2S channel interrupt of APU-controlled SDSoC hardware accelerator	94

Figure 2-10: **Reset Mechanism Used in the Design**

Send Feedback

Conclusion

The APU-controlled PL accelerators offer 4X, 6X, and 9X performance acceleration compared to the APU for 4,096, 16,384, and 65,536 point FFT sizes, respectively. The RPU-controlled PL accelerator offers 2X performance for 4K point FFT when compared to the APU. The performance difference between APU- and RPU-controlled accelerators is due to the OpenAMP overheads. The TRD shows the advantage of having a PL tightly coupled to PS, which allows computation-intensive tasks to be offloaded, resulting in software acceleration.

Table 2-4 compares computation times for various engines.

Table 2-4: **Average Computation Time Taken by Various Engines for Supported FFT Sizes**

FFT Computation Engine vs. FFT Size	4,096 (in µs)	16,384 (in µs)	65,536 (in µs)
1. APU	497	2,850	18,000
2. NEON	375	2,630	15,000
3. APU-controlled SDSoC PL accelerator	108	410	1,720
4. RPU as coprocessor	1,455[1]	N/A[2]	N/A[2]
5. RPU-controlled PL accelerator	136[1]	N/A[2]	N/A[2]

Notes:

1. RPU is running at 500 MHz and APU is running at 1.1G. OpenAMP communication latency is also included, which is approximately 100 µs.

2. For compute engines 4 and 5, only 4,096 point FFT is supported by the design and hence compute numbers for 16,384 and 65,536 point FFTs are not applicable (N/A).

OpenAMP Communication

Description

OpenAMP framework is used for communication between the APU master and RPU slave.

Messages are transferred between the APU and RPU to indicate the start and done state of the FFT engine using the RPMsg interface.

The APU application opens the RPMsg interface (the channel is established before this step, as described in the OpenAMP Framework section), and uses it for communication with the RPU by passing messages.

A typical flow of this implementation is as follows:

1. [APU] allocates the OCM memory (which is physically contiguous) for the input data buffer per the size of FFT.

2. [APU] allocates the OCM memory (which is physically contiguous) for the output data buffer per the size of FFT.

3. [APU] copies input samples from the PS DDR to OCM.

4. [APU] fills up pointers (physical address) for FFT input and output data buffers in the communication structure (which has some other bookkeeping fields, including information on the computation method: RPU software or PL).

5. [APU] writes the communication message to the RPMsg interface.

6. [APU] keeps waiting for a status message on the RPMsg read channel.

7. [RPU], upon receiving the message:

   ◦ For Flow D, computes FFT.

   ◦ For Flow E, configures AXI-DMA for FFT with the input and output buffer pointers received.

8. [RPU], after finishing the computation, sends a communication message using the rpmsg_send() API to indicate the status of the last computation.

9. [APU], upon receiving the successful message from RPU, plots the graphs per the received data.

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.

References

The most up-to-date information for this design is available on these websites:

Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit

Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit Documentation

Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit Master Answer Record (AR 66752)

These documents and sites provide supplemental material:

1. Open Asymmetric Multi Processing (OpenAMP) framework repository (github.com/OpenAMP/open-amp)
2. FreeRTOS (www.freertos.org)
3. Qt 5.7 Release Page (www.qt.io/qt5-7/)
4. *DDS Compiler LogiCORE IP Product Guide* (PG141)
5. *Fast Fourier Transform LogiCORE IP Product Guide* (PG109)
6. *System Management Wizard LogiCORE IP Product Guide* (PG185)

7. *UltraScale Architecture System Monitor* ([UG580](#))

8. ARM® NE10 Project ([projectne10.github.io/Ne10/](#))

9. *Xilinx AXI Reference Guide* ([UG761](#))

10. *SDSoC Environment User Guide* ([UG1027](#))

11. *SDSoC Environment User Guide: An Introduction to the SDSoC Environment* ([UG1028](#))

12. *SDSoC Environment User Guide: Platforms and Libraries* ([UG1146](#))

13. *ZCU102 Evaluation Board User Guide* ([UG1182](#))

Please Read: Important Legal Notices

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos.

Automotive Applications Disclaimer

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.