

SDAccel Platform Reference Design User Guide

Kintex UltraScale KCU1500 Acceleration Development Board

UG1234 (v2017.2) August 16, 2017

Revision History

08/16/2017: Released with SDx™ Development Environments 2017.2 without changes from 2017.1.

Date	Version	Revision
06/20/2017	2017.1	Updated to a new board platform. Title changed from KU115 to KCU1500. Reference Design files updated to the 2017.1 SDAccel Environment and updated for the new platform and device.
02/01/2017	2016.4	Updated document and reference design files for compatibility with SDx Environments 2016.4 release.
11/30/2016	2016.3	Initial Xilinx release.

Table of Contents

Revision History	2
Chapter 1: Overview	
Introduction	5
Platform Features	6
The Platform Reference Design	6
Chapter 2: Platform Characteristics	
Introduction	8
Expanded Partial Reconfiguration	8
Use with the SDAccel Environment	10
Sparse Memory Connectivity	11
Stacked Silicon Interconnect (SSI) Technology Support	12
Chapter 3: Hardware Platform	
Introduction	14
Host Connectivity	16
Memory Control	18
SDAccel OpenCL Programmable Region IP and the Programmable Region	20
AXI Interconnectivity	23
Application Profiling and Other Features	26
Clocking and Reset	27
Chapter 4: Software Platform	
Introduction	31
Address Map	31
Software Layers	36
Chapter 5: Implementation	
Introduction	38
create_design.tcl Detail	40
Design Constraints Detail	41

Chapter 6: Install, Bring-Up, and Use

Introduction	44
Installation	44
Bring-Up Tests	46
Use with the SDAccel Environment	46

Appendix A: Additional Resources and Legal Notices

Xilinx Resources	50
Solution Centers	50
Documentation Navigator and Design Hubs	50
References	51
Please Read: Important Legal Notices	51

Overview

Introduction

This document describes the platform reference design for the Kintex® UltraScale™ KCU1500 Acceleration development board, which when implemented in the Vivado® Design Suite produces a device support archive (DSA) suitable for use with the SDAccel™ Environment. With an understanding of the platform, you can also adapt the reference design to your own needs.

The platform reference design targets a Kintex UltraScale (KU115) device, and is distinguished from other such platforms by its use of four DDR4 SDRAM channels and the *expanded partial reconfiguration* flow, described later in this document.

The DSA produced by this platform is available with the 2017.1 SDx™ Environments in the following file:

```
xilinx_kcu1500_4ddr-xpr_4_0.dsa
```

Though the platform is designed for use with the SDAccel Environment when implemented in the form of a DSA, the reference design itself is available as a Vivado project. The design is constructed using the IP integrator tool, enabling easy modifications and visualization of connectivity and addressing.



IMPORTANT: *Beginning with the 2017.1 SDx Environments release, this reference design and the DSA it produces are compatible only with the Kintex UltraScale **KCU1500** Acceleration Development Board. They are not compatible with the previous revision of the board (termed Kintex UltraScale Developer Board for Acceleration with KU115), which has different FPGA pinouts and DDR4 memory configurations.*

Platform Features

The features of the Kintex UltraScale KCU1500 Acceleration development board and the Xilinx Acceleration KCU1500 4DDR Expanded Partial Configuration platform are intended for use as a high-performance acceleration platform for the SDAccel Environment, as follows:

- Connectivity to the host computer uses the Xilinx DMA subsystem for PCI Express® (PCIe), containing a Scatter-Gather DMA and PCI Express 3.x integrated block. The Kintex UltraScale KCU1500 Acceleration development board supports Gen3 x8 connectivity.
- Four UltraScale Memory IP instances enable access to available DDR4 SDRAM. The Kintex UltraScale KCU1500 Acceleration development board contains four channels of DDR4-2400 SDRAM, at 4GB per channel for a total of 16GB global memory.
- AXI SmartConnect IP provides high-performance AXI memory-mapped connectivity among IP cores throughout the platform.
- The SDAccel OpenCL Programmable Region IP core and its designated level of hierarchy enable the platform to be used with the SDAccel System Compiler, C/C++, OpenCL, and RTL user kernels.
- Partitioning, clocking, and reset networks are designed to support the expanded partial reconfiguration flow, including two independent frequency-adjustable kernel clock sources.
- SDx Environments application profiling using trace offload hardware infrastructure, and for flash memory programming using SPI flash IP infrastructure are supported.

The Platform Reference Design

The platform reference design includes the necessary design sources, scripts, and instructions to build the Xilinx Acceleration KCU1500 4DDR Expanded Partial Configuration platform. The modular scripts are used with Vivado from the SDx Environments installation to construct the IP integrator block diagram, synthesize the design, implement the design, and produce a DSA file for use with the SDAccel Environment.

To modify and adapt the design, you must do the following steps:

1. Construct the IP integrator block diagram within Vivado.
2. Make the desired changes.
3. Continue with synthesis, implementation, and DSA creation.

These individual steps and the scripts to run them are described in the instructions, and available from this link: [Reference Design Files](#), from which you can download the required files from the Xilinx website.

Device drivers are not supplied with the reference design, but are available with the SDx Environments installation and are described in [Chapter 4, Software Platform](#). For more details on software and device driver installation, see the *SDx Environments Release Notes, Installation, and Licensing Guide* (UG1238) [\[Ref 1\]](#).

For detailed guidance on the DSA creation process, see the *SDAccel Environment Platform Development Guide* (UG1164) [\[Ref 2\]](#).

The following chapters describe the platform characteristics, the Hardware Platform, the Software Platform, implementation, as well as the installation, bring-up, and use.

Platform Characteristics

Introduction

The Xilinx® Acceleration KCU1500 4DDR Expanded Partial Reconfiguration platform is a high-performance acceleration platform for the SDAccel™ Environment. The following sections describe its distinguishing characteristics.

Expanded Partial Reconfiguration

Many device support archives (DSAs) use partial reconfiguration to enable compiled binary downloads to the accelerator device while it remains online and linked to the PCIe® bus of the host. The Xilinx Acceleration KCU1500 4DDR Expanded Partial Reconfiguration platform is also designed for partial reconfiguration, but provides a larger, “expanded region” that makes over 80% of the overall fabric resources on the KU115 device available to the kernels. Only the logic necessary for keeping the link active and device operational, primarily the Xilinx DMA subsystem for PCI Express, basic control interfaces, and clock sources, is contained within a static “base region” floorplanned to less than 8% of the device area that cannot be used for kernel resources.

To make these additional fabric resources available to implemented kernel logic, the data path SmartConnect IPs, trace offload hardware infrastructure, SDAccel OpenCL Programmable Region IP, and all four of the DDR4 memory controller IPs are contained within the partially reconfigurable “expanded region” level of hierarchy. When the SDAccel System Compiler executes, kernel logic is placed and routed among these necessary resources, and a partial bitstream containing them is produced and included in the compiled binary.



IMPORTANT: *An outcome of locating the DDR4 memory controllers in the reconfigurable region is that DDR4 global memory content is lost when a new partial bitstream is downloaded.*

Figure 2-1 shows the IP integrator block diagram view of the top-level of the platform, colored to show the static base logical hierarchy in yellow, and the reconfigurable expanded region in green.

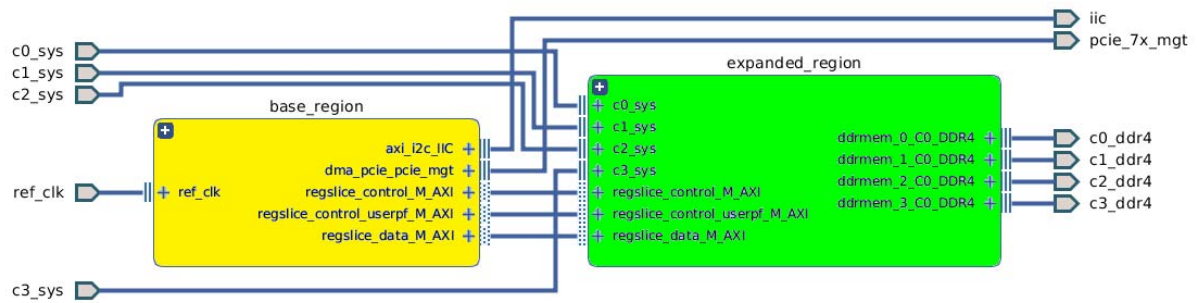


Figure 2-1: IP Integrator Top-Level View of Platform, Showing Base and Expanded Regions

Correspondingly, [Figure 2-2](#) shows the Vivado® device view of the implemented platform, where utilized and floorplanned static base region logic is highlighted yellow, and utilized expanded region logic is highlighted green.

In addition to the prominent rectangular regions of DDR4 memory controller IP instances, the reconfigurable expanded region contains the placeholder “add one” kernel, which is replaced by the implemented user kernel in the SDAccel System Compiler flow.

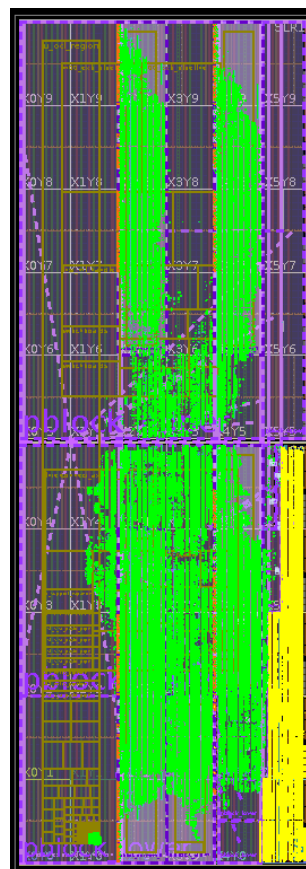


Figure 2-2: Device View of Implemented Platform, Showing Base and Expanded Regions

Use with the SDAccel Environment

When implemented and used to produce a DSA, the Hardware Platform is suitable for use with the SDAccel Environment and Software Platform. Generally speaking, this is enabled by the following characteristics, each of which is described in more detail throughout this document.

- **The SDAccel OpenCL Programmable Region IP core:** This IP core in the reconfigurable expanded region of the platform provides the SDAccel System Compiler with both a logical hierarchy and a large physical area (herein together referred to as the Programmable Region) to replace with the compiled user kernel and required interconnect, using partial reconfiguration. The physical area available to the Programmable Region with user kernels is the expanded region, as described above.
- **Software compatibility:** The memory-mapped IP cores, including the SDAccel OpenCL Programmable Region IP core, are configured to use address offsets and ranges that are compatible with the hardware abstraction Layer (HAL) driver provided with the SDx™ Environments installation. The kernel mode drivers for the Xilinx DMA subsystem for PCI Express are likewise provided with the SDx Environments installation.
- **Global memory access:** DDR4 SDRAM global memory is accessible to both the host and user kernels using AXI4 memory-mapped connectivity provided by AXI SmartConnect IP. Memory access capabilities are described in [Sparse Memory Connectivity](#).
- **DSA metadata:** The necessary Vivado project properties are applied such that the DSA creation phase captures the metadata for use with the SDAccel System Compiler.

For detailed guidance on the DSA creation process, see the *SDAccel Environment Platform Development Guide* (UG1164) [\[Ref 2\]](#).

Sparse Memory Connectivity

For maximum bandwidth, the Xilinx Acceleration KCU1500 4DDR Expanded Partial Reconfiguration platform connects the SDAccel OpenCL Programmable Region IP core (and therefore the Programmable Region for user kernels) to the four DDR4 memory controllers using an AXI4 memory-mapped 512-bit data path per instance. It also connects the Xilinx DMA subsystem for PCI Express to all four DDR4 memory controllers using an AXI4 memory-mapped 256-bit data path per instance. To simplify wiring so that routing and timing closure are feasible, the platform implements sparse connectivity. Specifically, within the platform:

- The host has access to the full 16GB of DDR4 SDRAM global memory space.
- Each of the four master interfaces on the SDAccel OpenCL Programmable Region IP core (and therefore the Programmable Region for user kernels) has access to one channel of DDR4 SDRAM; that is, 4GB of the 16GB space.

However, users can specify the required connectivity mapping of their kernels to master interfaces such that any kernel can access the amount of global memory it requires, up to and including all four channels. The SDAccel System Compiler then implements the appropriate wiring between kernels and Programmable Region master interfaces. In this way, while the default connectivity of the system does not impose an undue burden on the Vivado implementation tools, the user may specify higher connectivity at the expense of more challenging routing and timing closure. An abstract representation of the kernel to global memory connectivity is shown in [Figure 2-3](#). See [User-Specified Connectivity in Chapter 6](#) for more information.

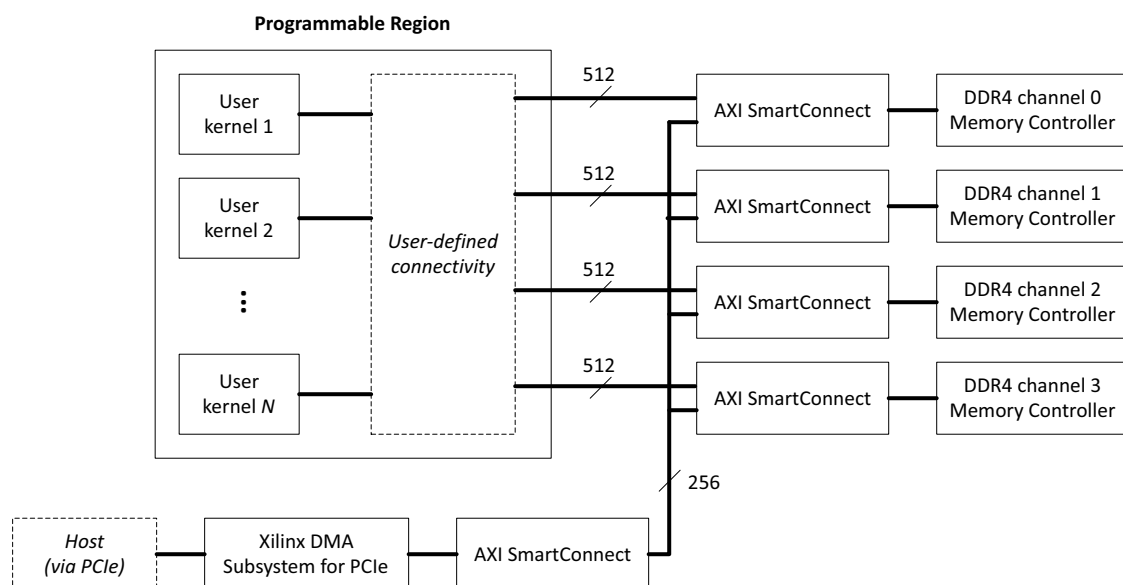


Figure 2-3: Sparse Memory Connectivity with User-Defined Kernel to Global Memory Connectivity

Stacked Silicon Interconnect (SSI) Technology Support

The Kintex® UltraScale™ KU115 device uses SSI technology and contains two super logic regions (SLRs). Though the high availability of total fabric resources is clearly beneficial, the delay penalty for routes that cross from one SLR to another can be significant, and make timing closure challenging in highly connected designs such as the Xilinx Acceleration KCU1500 4DDR Expanded Partial Reconfiguration platform. The AXI data paths used throughout the platform are wide and generally include combinatorial paths to implement the AXI4 memory-mapped protocol, making it imperative that such paths which cross into the other SLR are well-controlled and contain as few logic levels as possible. The platform uses optimized IP configurations, careful partitioning, and floorplanning where necessary to achieve this control and improve routability and timing closure.

The static base region of the design is floorplanned to the bottom-right corner of the lower SLR, corresponding to the location of the utilized PCI Express 3.x Integrated Block. Two of the four DDR4 SDRAM memory controller IP instances are floorplanned to the lower SLR, in regions around the High Performance I/O banks they utilize. The remaining two DDR4 SDRAM memory controller IP instances are floorplanned to the upper SLR, in regions around the High Performance I/O banks they utilize.

The required physical locations suggest the following design partitioning that is used in the platform:

- The Xilinx DMA subsystem for PCIe instance is floorplanned to the static base region of the lower SLR.
- The AXI SmartConnect 1x5 (1 slave interface, 5 master interfaces) instance which connects the Xilinx DMA subsystem for PCIe IP to the four AXI SmartConnect 2x1 instances and the AXI Performance Monitor is floorplanned to the reconfigurable expanded region of the lower SLR.
- The two AXI SmartConnect 2x1 instances which each connect the Programmable Region and host to a DDR4 SDRAM memory controller IP instance in the lower SLR are floorplanned to the lower SLR, except:
 - Input pipeline stages on the master interfaces connected to the Programmable Region are not floorplanned, facilitating potential SLR crossing through automatic placement when user kernels are automatically placed in the upper SLR.
- The two AXI SmartConnect 2x1 instances which each connect the Programmable Region and host to a DDR4 SDRAM memory controller IP instance in the upper SLR are floorplanned to the upper SLR, except:
 - Input pipeline stages on the master interfaces connected to the Programmable Region are not floorplanned, facilitating potential SLR crossing through automatic placement when user kernels are automatically placed in the lower SLR.

- Input pipeline stages on the master interfaces connected to the host are variously floorplanned to each SLR, straddling the boundary to facilitate known SLR crossing.

The following figure shows a simplified representation of the IP partitioning described above (and is not precisely representative of the device floorplan), where:

- The red horizontal line represents the boundary between the two SLRs.
- The small white squares represent pipeline stages floorplanned to a given SLR.
- The small gray squares with dotted outlines represent pipeline stages that connect to the Programmable Region and which, like the Programmable Region, are not floorplanned and can be placed on either side of the SLR as determined by the implementation tools.

Note: The pipeline stages are actually submodules of SmartConnect IP instances, but for clarity are shown as separate entities in the simplified picture.

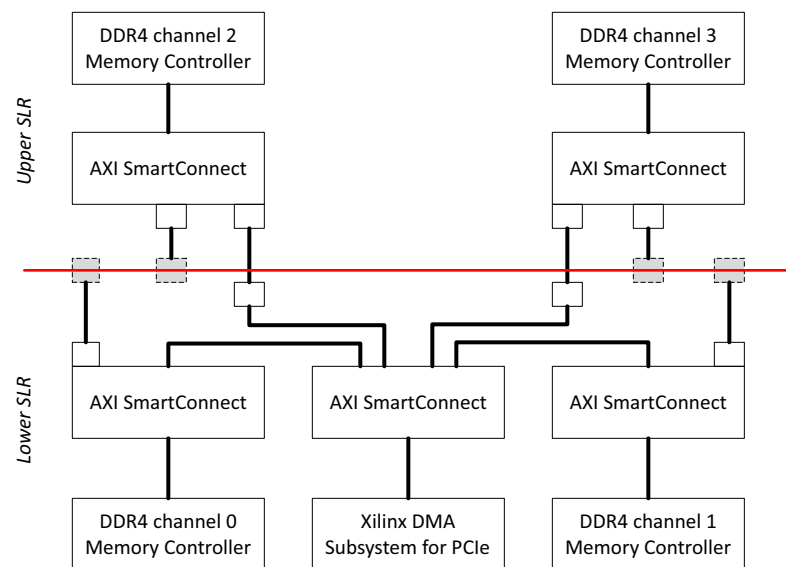


Figure 2-4: Simplified Representation of IP Partitioning for Controlled SLR Crossing

The combination of static base and reconfigurable expanded region partitioning, with the optimal configuration and careful floorplanning of IP instances, together facilitate the necessary controlled SLR crossing in the KU115 device to effectively utilize both SLRs.

Hardware Platform

Introduction

In this chapter, the hardware architecture of the Xilinx® Acceleration KCU1500 4DDR Expanded Partial Configuration platform is described from the perspective of its Vivado® IP integrator block diagram representation, in six parts, as follows:

- [Host Connectivity](#)
- [Memory Control](#)
- [SDAccel OpenCL Programmable Region IP and the Programmable Region](#)
- [AXI Interconnectivity](#)
- [Application Profiling and Other Features](#)
- [Clocking and Reset](#)

The top level of the platform block diagram in the default IP integrator view shows the static base region and reconfigurable expanded region as two hierarchical cells, connected by interfaces and wires. To simplify the view, the **Show interface connections only** option can be used. [Figure 3-1](#) and [Figure 3-2](#), show the views respectively. The remainder of this chapter makes use of both views as appropriate.

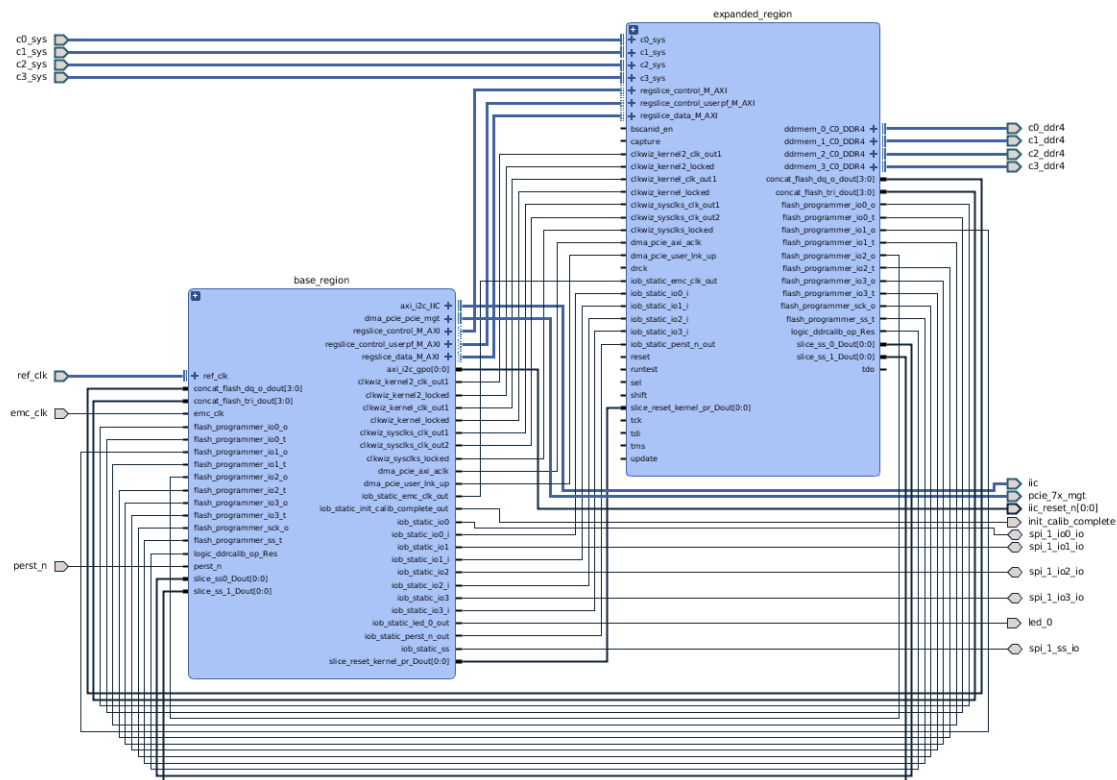


Figure 3-1: Top-Level IP Integrator Block Diagram - Default View with Interfaces and Wires

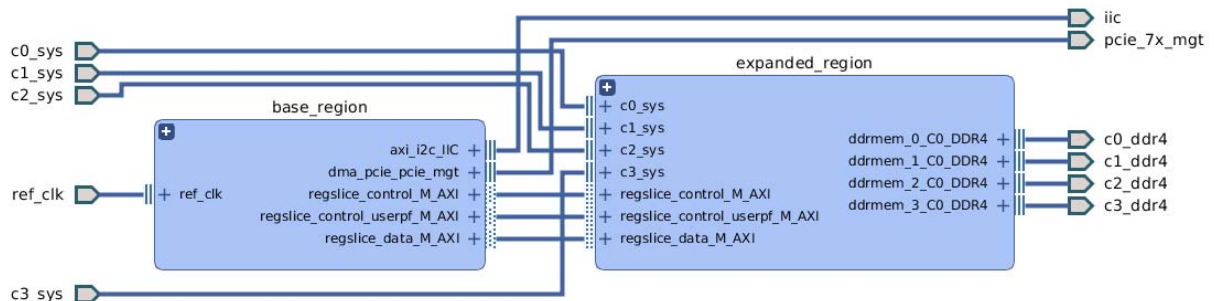


Figure 3-2: Top-level IP Integrator Block Diagram - Interface Connections Only View

Host Connectivity

The platform implements connectivity to the host computer using the Xilinx DMA subsystem for PCI Express® (PCIe) IP, which contains a scatter-gather DMA and a PCIe 3.x integrated block.

For best performance on the Kintex® UltraScale™ KCU1500 Acceleration development board, the PCIe integrated block can negotiate a link up to Gen3 x8 connectivity, or 8.0 GT/s.

The Kintex UltraScale KCU1500 Acceleration development board supplies a 100 MHz reference clock and uses PCIe block location X0Y0. The platform uses an AXI memory-mapped interface operating at 250 MHz with a 256-bit data width. The basic customization of the Xilinx DMA subsystem for PCIe IP core (herein referred to as the XDMA IP core) is shown in the following figure.

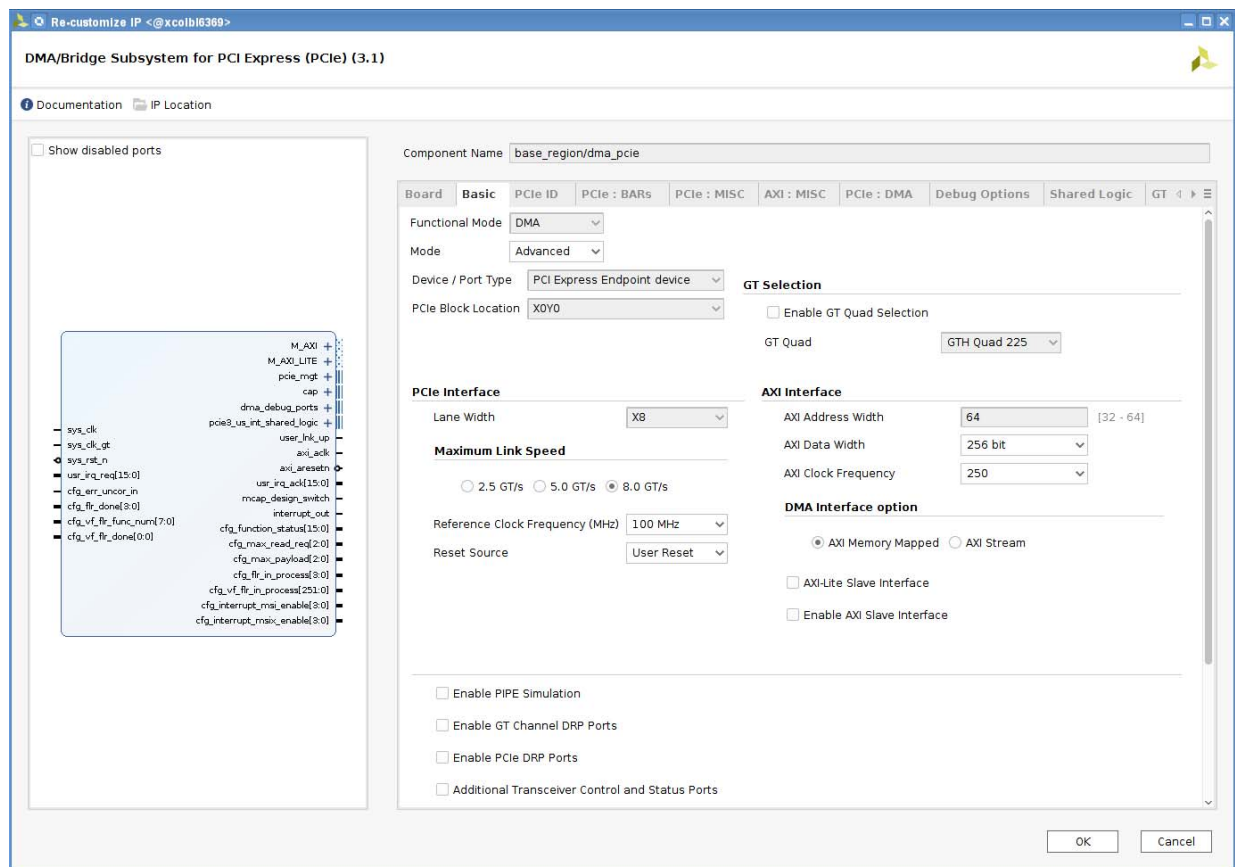


Figure 3-3: XDMA IP customization - Basic Tab

For compatibility with the provided kernel mode and HAL drivers, the XDMA IP instance is customized to use Vendor ID 0x10EE (Xilinx Vendor ID), Device IDs 0x4B27 and 0x4B28, and Subsystem ID 0x4340.



IMPORTANT: Derived platforms with user modifications must not use IDs identical to the reference design platform. See the *SDAccel Environment Platform Development Guide (UG1164)* [Ref 2] for more information on device drivers for SDx™ accelerator devices.

As shown in the following figure, the XDMA IP is customized to provide the host access to up to 4MB of memory-mapped platform resources, over its AXI4-Lite control interface.

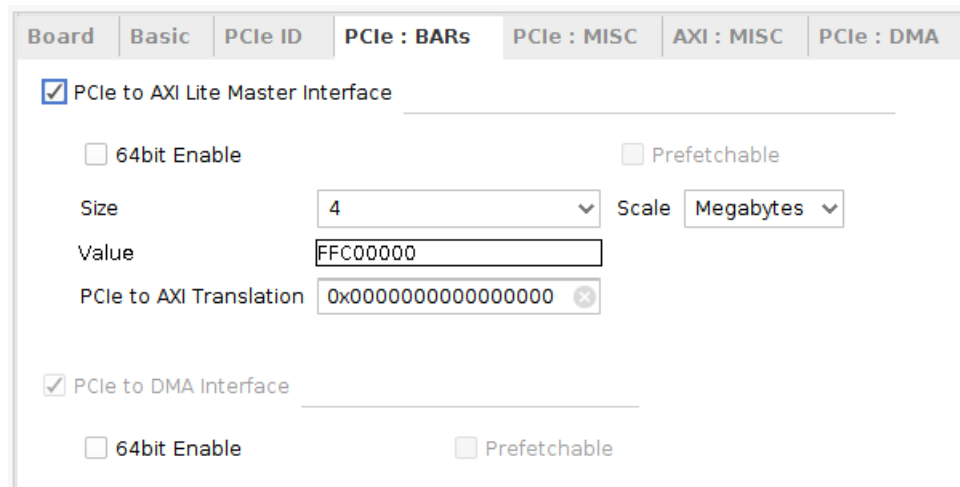


Figure 3-4: XDMA IP customization - PCIe: BARs Tab

As shown in the following figure, the XDMA IP is customized to use two DMA read and two DMA write channels, with 32 read IDs and 16 write IDs, for a balance of performance and area.

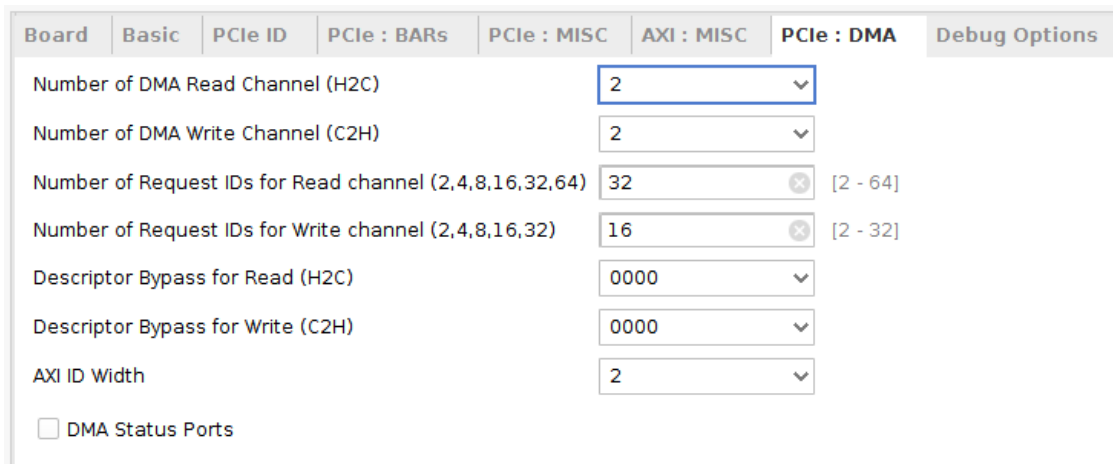


Figure 3-5: XDMA IP Customization - PCIe: DMA Tab

See the IP Customization GUI of the XDMA instance in the provided platform reference design for all XDMA IP customization settings. See the *DMA Subsystem for PCI Express v3.0 Product Guide (PG195)* [Ref 3] for more information on the XDMA IP core, its features, and customizations options.

Memory Control

The Kintex UltraScale KCU1500 Acceleration development board uses four channels of DDR4-2400 SDRAM at 4GB per channel for a total of 16GB. One instance of UltraScale Memory IP (herein referred to as DDR4 IP) per channel is used as the memory controller for that channel.

The DDR4 IP instances are customized to target the MT40A512M16HA-083E components on the Kintex UltraScale KCU1500 Acceleration development board, with a 512-bit data width, 32-bit address width AXI memory-mapped interface providing high-bandwidth platform fabric access to the full 4GB per channel.

As described in [Sparse Memory Connectivity in Chapter 2](#), the host has access to all global memory and the user kernels have access to the user-defined range. The DDR4 IP Customization GUI with populated Basic tab values is shown in the following figure.

Note: Three of the four DDR4 IP instances are identical and enable ECC. One DDR4 channel does *not* support ECC.

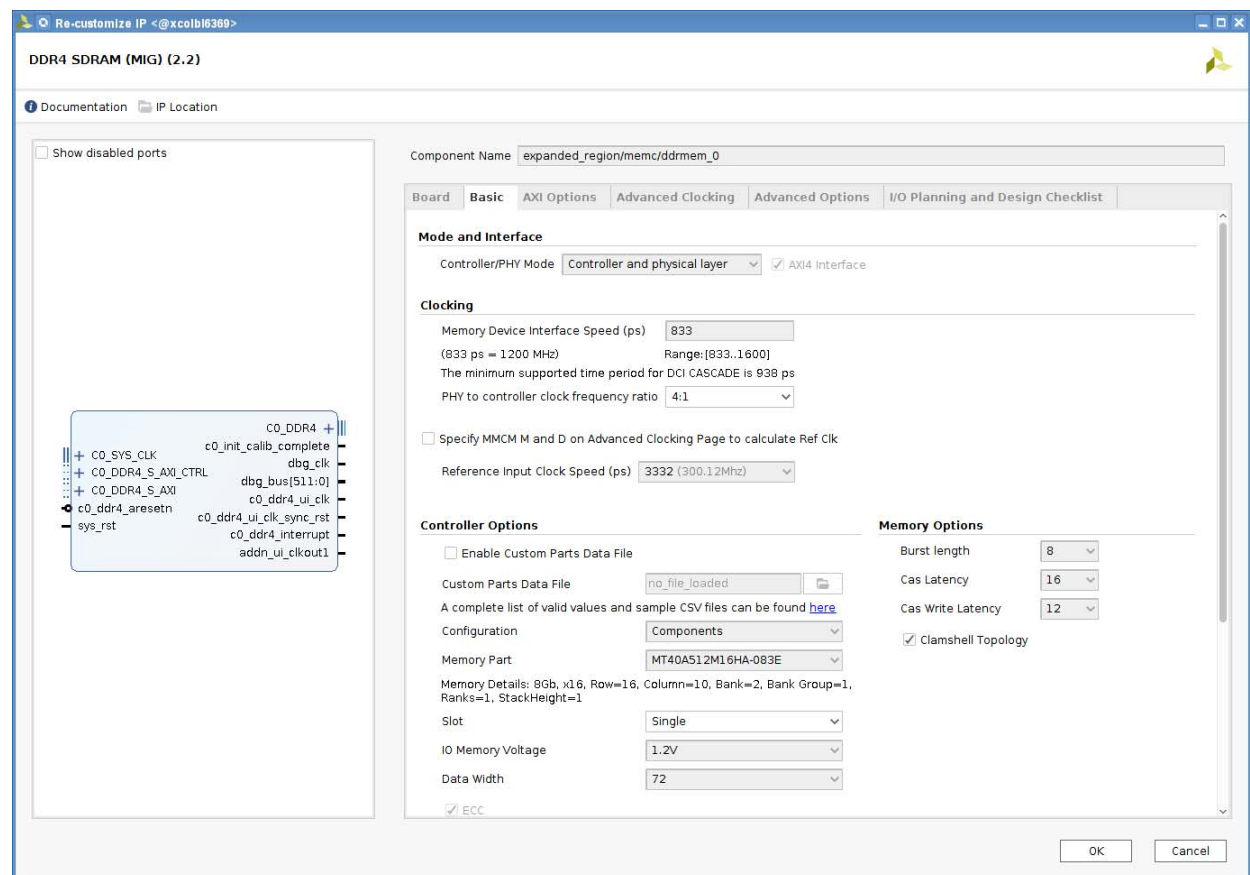


Figure 3-6: DDR4 IP Customization - Basic Tab

Contained within the `memc` sub-hierarchy of the reconfigurable expanded region as shown in the following figure, the four DDR4 IP instances are individually accessed, and individually indicate their calibration status using the `c0_init_calib_complete` output port. The logical AND of these signals is available to the device driver using a memory-mapped GPIO IP instance in the static base region.

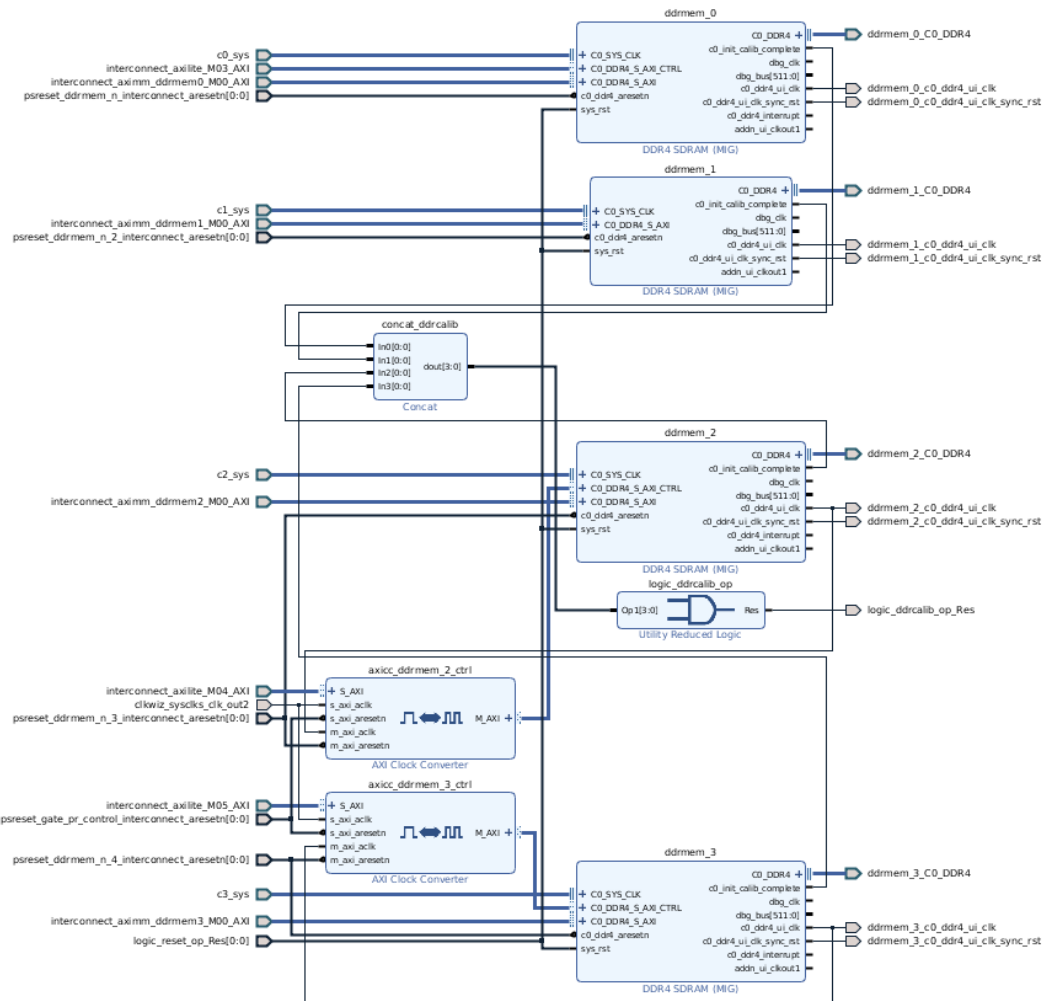


Figure 3-7: DDR4 IP Instances and Supporting IP in Reconfigurable Expanded Region `memc` Sub-Hierarchy

As introduced in [Stacked Silicon Interconnect \(SSI\) Technology Support in Chapter 2](#), the `ddrmem_0` and `ddrmem_1` instances are located in the lower SLR of the device, the `ddrmem_2` and `ddrmem_3` instances are located in the upper SLR.

AXI Clock Converter IP instances facilitate lower-speed SLR crossing of control signals by converting from a 50 MHz clock domain source in the lower SLR into the DDR4 IP native 300 MHz AXI clock domain for those DDR4 IP instances in the upper SLR.



IMPORTANT: An outcome of locating the DDR4 memory controllers in the reconfigurable region is that DDR4 global memory content is lost when a new partial bitstream is downloaded.

SDAccel OpenCL Programmable Region IP and the Programmable Region

Instantiated within the reconfigurable expanded region, the SDAccel OpenCL Programmable Region IP core instance is used to designate a level of platform hierarchy to be used with the SDAccel System Compiler and C/C++, OpenCL, and RTL user kernels. While the IP instance in the IP integrator block diagram of the platform contains only a simple “add one” placeholder kernel (also known as a *training* kernel, which is necessary to avoid excessive logic pruning throughout the platform), the IP instance importantly designates a logical programmable region level of hierarchy in the resulting DSA that the SDAccel System Compiler later replaces with user-defined kernels and the surrounding AXI connectivity to the remainder of the platform.

As previously described in [Expanded Partial Reconfiguration in Chapter 2](#), recall that although only the Programmable Region is replaced with user-defined kernel content during the SDAccel System Compiler flow, the generated partial bitstream corresponds to the entirety of the expanded region level of hierarchy. Because the expanded region is a superset of the Programmable Region and other platform logic, each SDAccel System Compiler runs place and route on the entirety of that expanded region, allowing the user kernels to be flexibly placed and routed together with the remainder of the expanded region logic; thereby providing higher fabric capacity to the user kernel content than if only the Programmable Region were contained in the partial bitstream.

The following figure shows the SDAccel OpenCL Programmable Region IP core instance.

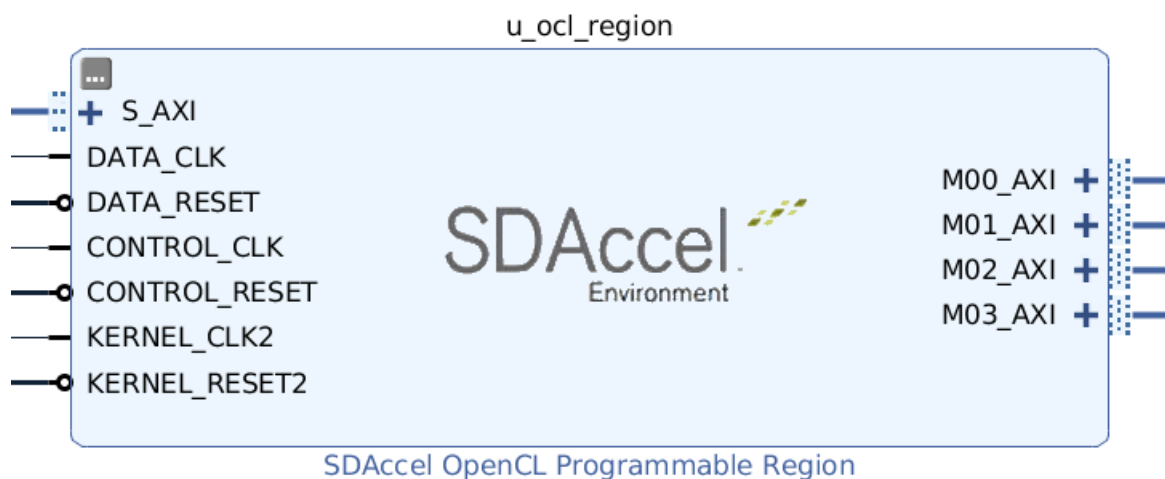


Figure 3-8: SDAccel OpenCL Programmable Region IP Core Instance and its Interfaces

- The `S_AXI` interface is an AXI4-Lite slave interface, allowing host control of user kernels.

- The M00_AXI through M03_AXI interfaces are high-bandwidth, 512-bit AXI memory-mapped master interfaces that indirectly connect to the four DDR4 IP memory controller instances as previously described in [Chapter 2, Platform Characteristics](#), sections [Sparse Memory Connectivity](#) and [Stacked Silicon Interconnect \(SSI\) Technology Support](#).
- The CONTROL_CLK and CONTROL_RESET ports are clock and reset synchronous to the slower S_AXI (AXI4-Lite control) interface.
- The DATA_CLK and DATA_RESET ports are clock and reset for the primary data paths used in the kernel(s) as well as the M00_AXI through M03_AXI interfaces.
- The KERNEL_CLK2 and KERNEL_RESET2 ports are clock and reset for an optional second domain which can be used only in RTL user kernels. As needed, this domain provides users the flexibility to run logic in RTL kernels at a different frequency than, or generally asynchronous to, the DATA_CLK and DATA_RESET domain.



IMPORTANT: Data paths must be synchronous to the DATA_CLK and DATA_RESET domain on the output interfaces of the kernel(s), making the RTL kernel developer responsible for transferring data from the KERNEL_CLK2 domain into the DATA_CLK domain on egress of the kernel.

The SDAccel OpenCL Programmable Region IP core is a hierarchical IP. Viewing its contents reveals the subsystem block diagram shown in the following figure.

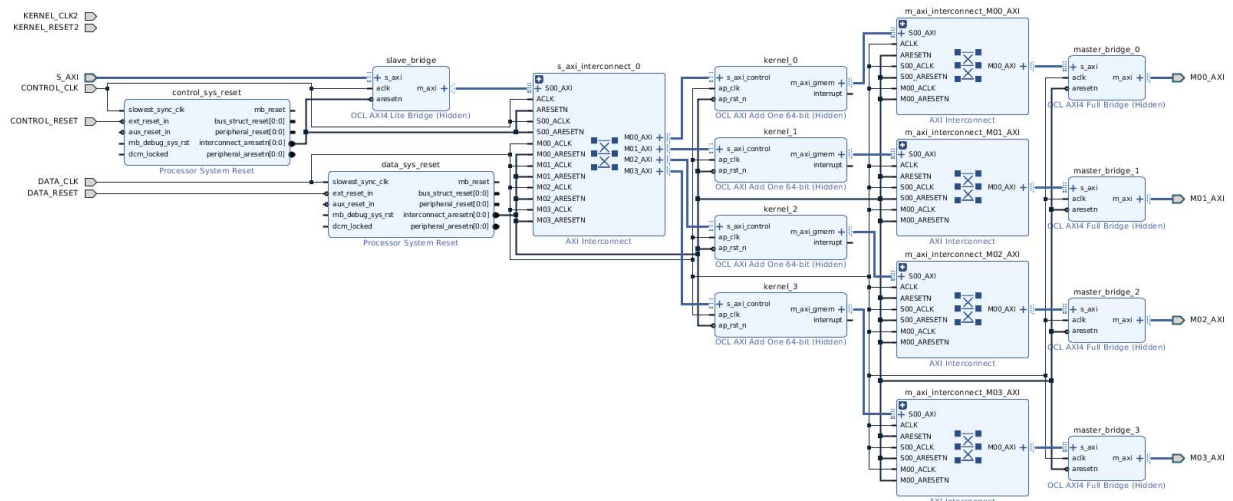


Figure 3-9: SDAccel OpenCL Programmable Region Hierarchical IP Subsystem

- The four “add one” training kernel instances, `kernel_0` through `kernel_3`, are connected to the single `S_AXI` slave control interface of the Programmable Region using a 1x4 AXI Interconnect instance, which also performs domain crossing between `CONTROL_CLK` and `CONTROL_RESET` on its slave interface (Programmable Region boundary-facing) side, and `DATA_CLK` and `DATA_RESET` on its master interface (kernel-facing) side.

- One AXI Interconnect instance is present per master interface, M00_AXI through M03_AXI.

Though providing 1x1 connectivity in the platform with training kernels, the SDAccel System Compiler flow implements the user-defined connectivity from kernel(s) to downstream SmartConnect instances and then to DDR4 memory, as defined in [Sparse Memory Connectivity](#). In general, the SDAccel System Compiler instantiates user kernels, then the AXI Interconnect instances on slave and master sides of those kernels, and makes all necessary connections to the pre-defined Programmable Region boundary. In this way, although the contents of the Programmable Region depend on the user code, the necessary connectivity to the platform is maintained.

Note: The optional KERNEL_CLK2 and KERNEL_RESET2 ports are unused in the platform training kernels, but are available to user RTL kernels as needed.

The SDAccel OpenCL Programmable Region IP instance customization Basic tab is shown in the following figure.

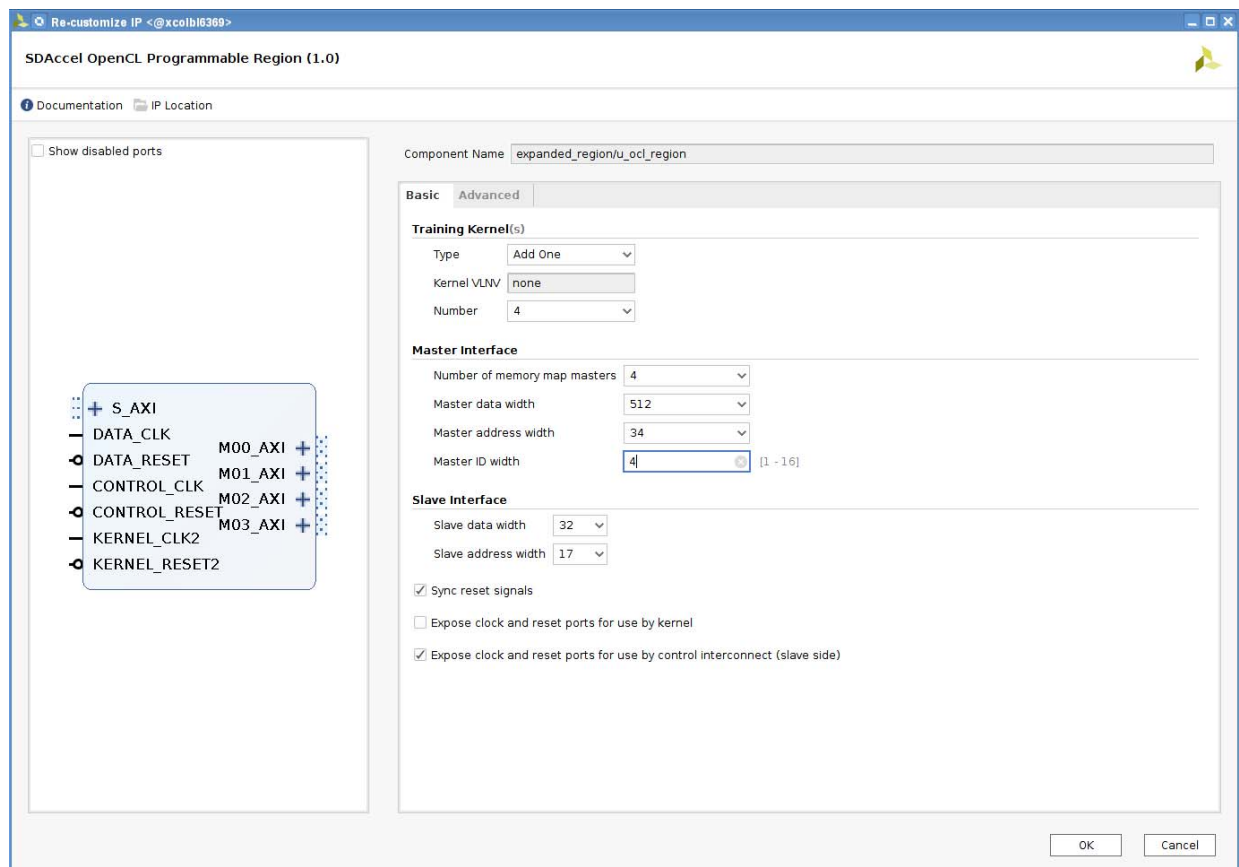
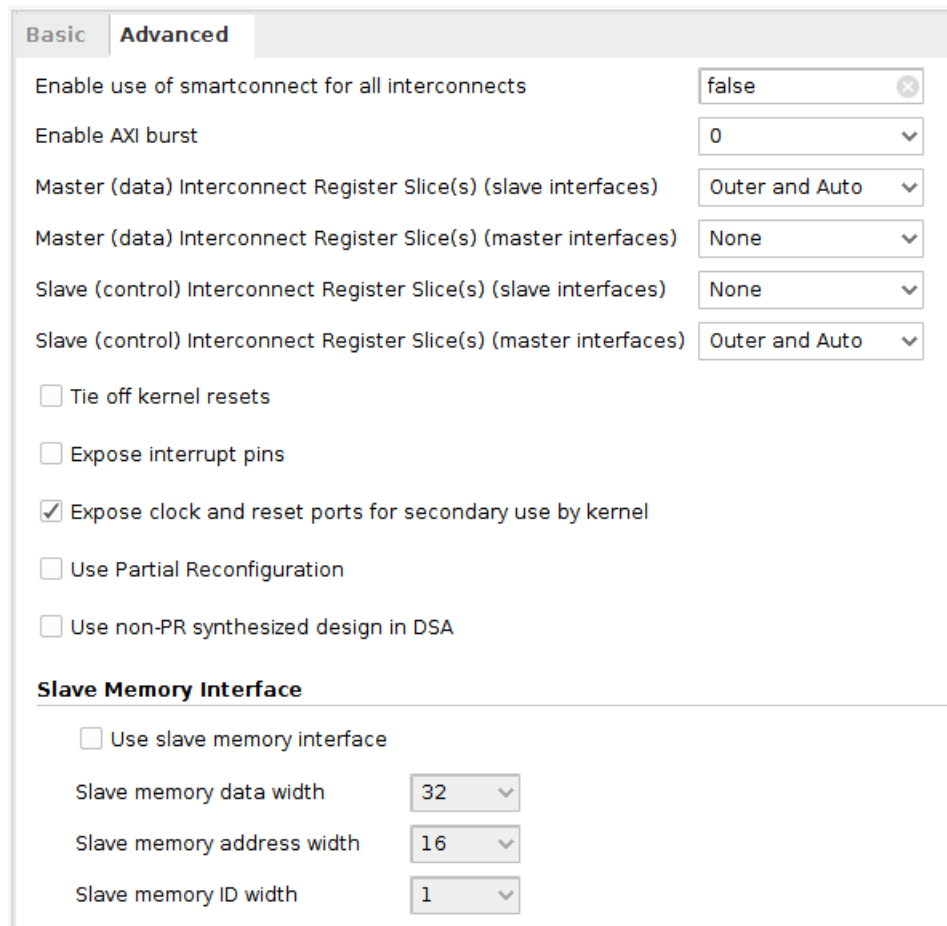


Figure 3-10: SDAccel OpenCL Programmable Region IP Customization - Basic Tab

The following figure shows the Advanced tab.



The screenshot shows the 'Advanced' tab of the 'SDAccel OpenCL Programmable Region IP Customization' window. The 'Basic' tab is also visible. The 'Advanced' tab contains the following settings:

- Enable use of smartconnect for all interconnects: false
- Enable AXI burst: 0
- Master (data) Interconnect Register Slice(s) (slave interfaces): Outer and Auto
- Master (data) Interconnect Register Slice(s) (master interfaces): None
- Slave (control) Interconnect Register Slice(s) (slave interfaces): None
- Slave (control) Interconnect Register Slice(s) (master interfaces): Outer and Auto
- ☐ Tie off kernel resets
- ☐ Expose interrupt pins
- ☒ Expose clock and reset ports for secondary use by kernel
- ☐ Use Partial Reconfiguration
- ☐ Use non-PR synthesized design in DSA

Below these settings is the 'Slave Memory Interface' section, which is currently collapsed. It contains the following settings:

- ☐ Use slave memory interface
- Slave memory data width: 32
- Slave memory address width: 16
- Slave memory ID width: 1

Figure 3-11: SDAccel OpenCL Programmable Region IP Customization - Advanced Tab

For more information on the Programmable Region, the IP that defines it, and the DSA creation flow, see the *SDAccel Environment Platform Development Guide* (UG1164) [Ref 2].

AXI Interconnectivity

The primary data path of the platform consists of AXI memory-mapped access from the host (using an XDMA IP instance) to all global memory (using individual DDR4 IP instances), and from the user kernels to user-defined regions of global memory. High-performance data path connectivity is implemented using five instances of AXI SmartConnect IP, with the topology previously described in [Sparse Memory Connectivity in Chapter 2](#).

Contained within the interconnect sub-hierarchy of the reconfigurable expanded region as shown in [Figure 3-12](#), the five SmartConnect IP instances together provide both the host and the user kernels with high-performance access to the four DDR4 IP memory controllers.

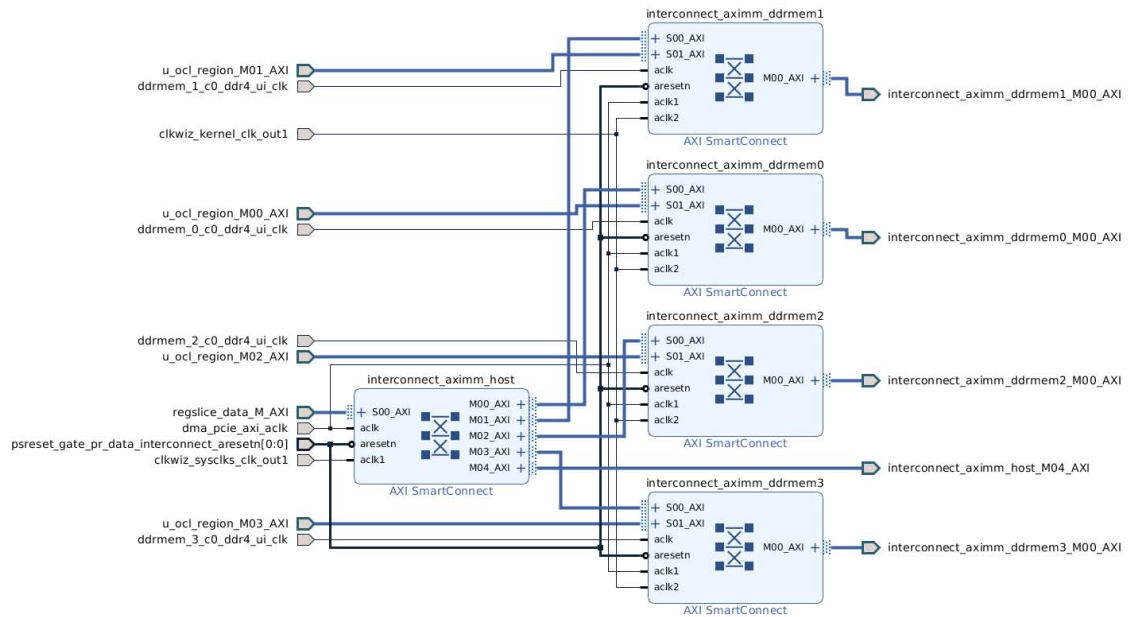


Figure 3-12: AXI SmartConnect IP Instances in Reconfigurable Expanded Region Interconnect Sub-Hierarchy

- The `interconnect_aximm_host` instance of SmartConnect IP is customized to use 1 slave interface and 5 master interfaces; a 1x5 customization, where each interface uses a 256-bit data path.
 - The `S00_AXI` slave interface is connected to the XDMA IP using AXI Register Slice IP for partial reconfiguration isolation (see [Partial Reconfiguration Isolation](#) for details), and is synchronous to the XDMA clock from PCIe link.
 - Four master interfaces, `M00_AXI` through `M03_AXI`, connect to the four 2x1 SmartConnect instances, which in turn each connect to one DDR4 IP instance and are also synchronous to the XDMA clock from PCIe link.
 - The `M04_AXI` master interface connects to the AXI Performance Monitor IP for application profiling support (see [Application Profiling and Other Features](#) for details), and is synchronous to the dedicated AXI Performance Monitor clock.
- The `interconnect_aximm_ddrmem0` through `interconnect_aximm_ddrmem3` instances of SmartConnect IP are each customized to use 2 slave interfaces and 1 master interface; 2x1 customizations.
 - The `S00_AXI` interface of each is connected to one of the master interfaces of the `interconnect_aximm_host` instance for host access to global memory, and are synchronous to the XDMA clock from PCIe link, driving the `aclk1` port.
 - The `S01_AXI` interface of each is connected to one of the four master interfaces of the SDAccel OpenCL Programmable Region IP (and therefore the Programmable Region) for kernel access to global memory, and is synchronous to the kernel clock, driving the `aclk2` port.

- The M00_AXI interface of each is connected to the corresponding DDR4 IP instance; therefore, the `ac1k` port of each instance is driven by the connected AXI interface clock DDR4 IP instance. See [Sparse Memory Connectivity in Chapter 2](#) for context.

The following figure shows the connectivity from the XDMA IP master, through the `dma_pf_demux` to produce the management and user paths, to the two AXI Interconnect instances which connect to the relevant peripherals on each path.

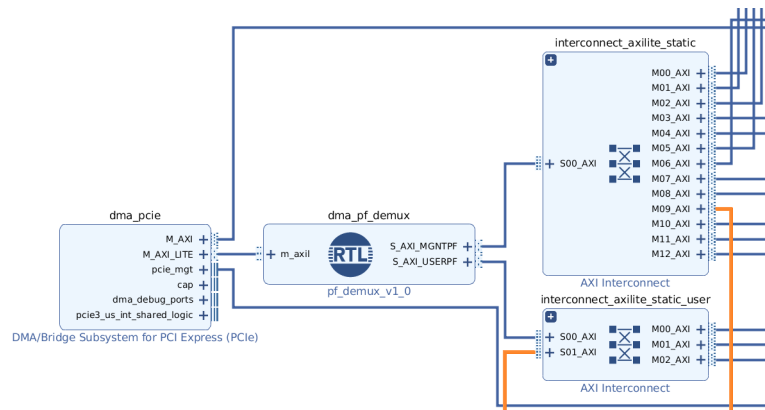


Figure 3-13: AXI4-Lite Control Path Connectivity

- Host access using XDMA IP is the control path master of the platform. The XDMA IP is configured for two physical functions, which are then split into the two distinct low-speed, 32-bit AXI-Lite paths using the `dma_pf_demux` instance in the static base region. A management path corresponds system peripherals, while a user path controls user kernels and associated peripherals.
- The management path driven by the `dma_pf_demux S_AXI_MGNTPF` interface provides the XDMA IP master with memory-mapped access to peripherals in both the static base regions using a 1x13 AXI Interconnect IP and a downstream 1x4 AXI Interconnect IP in the reconfigurable expanded region. Those two AXI Interconnect instances are isolated from one another by partial reconfiguration isolation logic (see [Partial Reconfiguration Isolation](#) for details).
- The user path driven by the `dma_pf_demux S_AXI_USERPF` interface provides the XDMA IP master with memory-mapped access to peripherals in both the static base region using a 2x3 AXI Interconnect IP and a downstream 1x3 AXI Interconnect IP in the reconfigurable expanded region. Those two AXI Interconnect instances are isolated from one another by partial reconfiguration isolation logic (see [Partial Reconfiguration Isolation](#) for details). The second master interface on the static base region 2x3 AXI Interconnect IP allows the management path to access all peripherals on the user path.

Application Profiling and Other Features

Beyond the fundamentals of host connectivity, memory control, SDAccel System Compiler support enabled by the Programmable Region, and AXI interconnectivity, the platform offers other features through a combination of Hardware Platform IP and Software Platform driver support:

- SDx Environments application profiling support, using the trace offload hardware infrastructure
- Kintex UltraScale KCU1500 Acceleration development board flash memory programming using SPI flash IP infrastructure
- Kintex UltraScale KCU1500 Acceleration development board FPGA fan speed control using memory-mapped I2C controller
- AXI Firewall IP protection of the static base region hardware against potential AXI protocol violations from kernels
- SDx Feature ROM, to assist the SDAccel Software Platform in interpreting the Hardware Platform feature set
- Xilinx Virtual Cable usage using the Debug Bridge
- System Management Wizard IP-based device temperature and voltage monitoring

The following figure shows the `apm_sys` sub-hierarchy of the reconfigurable expanded region.

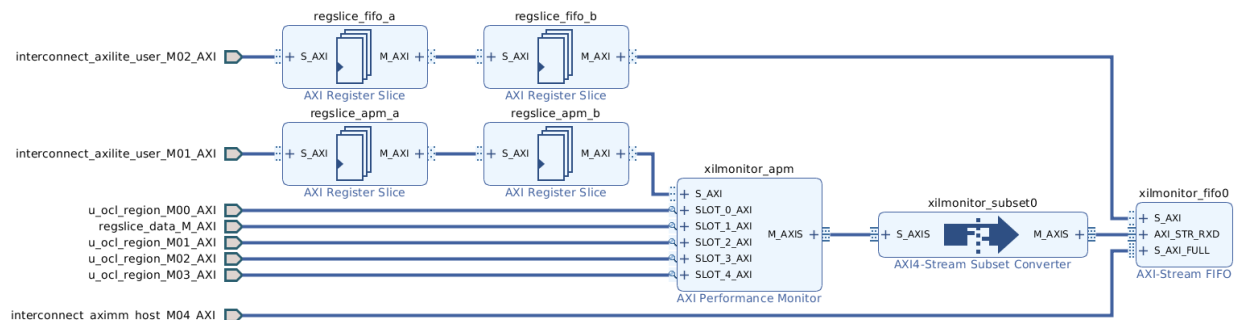


Figure 3-14: Trace Offload Hardware Infrastructure in Reconfigurable Expanded Region `apm_sys` Sub-Hierarchy

- The trace offload hardware infrastructure supports SDx Environments application profiling by monitoring and characterizing AXI transactions.
- The AXI Performance Monitor IP instance uses five AXI memory-mapped monitor interfaces, driven by the four master interfaces of the Programmable Region (for kernel monitoring) and the XDMA IP using AXI Register Slice IP for partial reconfiguration isolation (for host monitoring).

After tabulation, key profiling data is transformed using an AXI4-Stream Subset Converter and buffered in an AXI-Stream FIFO for interpretation by the SDx Environments application profiling feature. AXI Register Slice instances on the control path ease automatic placement.

Clocking and Reset

The following sections describe clocking and reset on the Kintex UltraScale KCU1500 Acceleration development board:

- [Platform Clocking](#)
- [Platform Resets](#)
- [Partial Reconfiguration Isolation](#)

Platform Clocking

The following table describes the primary clock domains of the platform, their sources, frequency, and usage in the platform.

Table 3-1: Platform Clock Domains

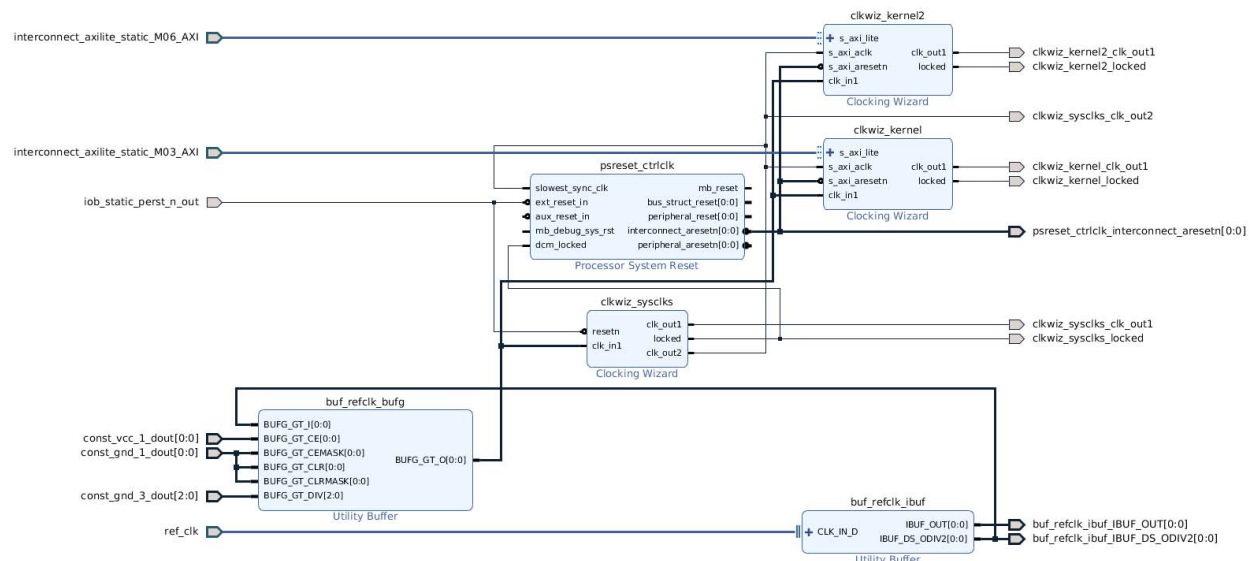
Clock Domain	Source	Frequency	Use in Platform
XDMA	XDMA IP core	250 MHz	Within XDMA IP core, and its AXI master interfaces
AXI4-Lite control	Clocking Wizard IP instance clkwiz_sysclks	50 MHz	The majority of AXI4-Lite control paths throughout the platform
Kernel clock	Clocking Wizard IP instance clkwiz_kernel	Default of 300 MHz. Can be overridden by the user at compile time, as well as automatically scaled at runtime.	The clock domain for C/C++ and OpenCL kernels, and the primary clock domain for RTL kernels. Also used for the AXI memory-mapped connections between the Programmable Region and AXI SmartConnect IP instances.
Kernel clock 2	Clocking Wizard IP instance clkwiz_kernel2	Default of 500 MHz. Can be overridden by the user at compile time, as well as automatically scaled at runtime.	The optional secondary clock domain for RTL kernels. When utilized, available internal to those RTL kernels only.

Table 3-1: Platform Clock Domains (Cont'd)

Clock Domain	Source	Frequency	Use in Platform
AXI Performance Monitor	Clocking Wizard IP instance clkwiz_sysclks	300 MHz	Exclusive to trace offload hardware infrastructure
DDR4 memory controller clock (one per instance)	DDR4 IP instance ddrmem_[0 1 2 3]	300 MHz	Within the ddrmem_[0 1 2 3] IP instance, and for the AXI connectivity to the corresponding SmartConnect IP instance

The XDMA, AXI4-Lite control, kernel clock, kernel clock 2, and AXI Performance Monitor clocks are all derived from the PCIe 100 MHz differential `ref_clk` top-level input from the Kintex UltraScale KCU1500 Acceleration development board.

The following figure shows the `base_clocking` sub-hierarchy of the static base region.


 Figure 3-15: Clocking Resources in Static Base Region `base_clocking` Sub-Hierarchy

Contained within that region are the reference clock that drives clock buffer IP instances `buf_refclk_ibuf` and `buf_refclk_bufg`, for serial transceiver and fabric clock access, respectively. The `buf_refclk_bufg` instance then drives the three Clocking Wizard IP instances in the system.

Although the AXI4-Lite control and AXI Performance Monitor clocks are both produced from a single Clocking Wizard IP instance using a PLL, the kernel clock and kernel clock 2 are each generated from a separate, memory-mapped Clocking Wizard IP instance using an MMCM. This is to support the automatic frequency scaling feature of the SDAccel Software Platform runtime, which at the time the partial bitstream is loaded, adjusts the operating frequency of the kernel clock and kernel clock 2 independently in accordance with the timing results of the implemented design. See [Timing Closure in Chapter 6](#) for details.

For example, if user kernel logic operating entirely on the kernel clock domain was constrained to the default of 300 MHz (3.333ns period) but finished implementation with a -235ps WNS setup violation on paths limited to the kernel clock domain, then the SDAccel runtime will adjust the kernel clock MMCM source to produce a 280 MHz (3.568ns) kernel clock instead. The new operating frequency, if any, is reported as a warning near the end of the SDAccel System Compiler flow. Timing paths on other clock domains must successfully close timing in the SDAccel System Compiler flow, however; and for that reason, the `HIGH_PRIORITY` clock property is used as described in [Design Constraints Detail in Chapter 5](#).

Platform Resets

A Processor System Reset IP core instance exists per clock domain, and drives a reset output to the reset interfaces of the IP core, synchronous to that domain. Generally:

- The `dcm_locked` (clock is stable) input of a reset controller operating synchronously to a clock produced by a Clocking Wizard IP instance is driven by the `locked` output of that instance.
- The `dcm_locked` (clock is stable) input of a reset controller operating synchronously to the XDMA clock is driven by `user_lnk_up` output of the XDMA IP instance.
- The `ext_reset_n` (reset source) input of a reset controller in the reconfigurable expanded region is driven by the memory-mapped `gate_pr` GPIO IP core instance, as described in [Partial Reconfiguration Isolation](#).

Note: The DDR4 IP reset controllers do not use this scheme and are reset by the IP instances themselves.

The platform reset methodology is simple, except for the need to isolate the expanded region logic during partial reconfiguration, which is described in the following section.

Partial Reconfiguration Isolation

The XDMA IP is contained in the static base region level of hierarchy, and must be isolated from changes to the reconfigurable expanded region. In this way, the PCIe link is not disrupted when a new binary is downloaded to the reconfigurable expanded region area of the accelerator device.

The device driver manages this non-disruptive partial bitstream download process with the aid of the platform hardware. Contained within the `pr_isolation_expanded` sub-hierarchy of the static base region, a memory-mapped GPIO IP core instance named `gate_pr` is used to hold the reconfigurable expanded region logic, as well as flip-flops at the boundary of the static base region, in reset.

When a new partial bitstream is to be downloaded, the device driver writes to a `gate_pr` register that causes its output to do the following:

- Hold the static base region reset controller `psreset_regslice_data_pr` in reset, which issues a synchronous reset to the `regslice_data` AXI Register Slice IP, and thereby holds the XDMA data path flop-flops at the boundary of the static and reconfigurable regions in reset.
- Hold the static base region reset controller `psreset_regslice_ctrl_pr` in reset, which issues a synchronous reset to the `regslice_control` AXI Register Slice IP, and thereby holds the XDMA control path flop-flops at the boundary of the static and reconfigurable regions in reset.
- Hold logic in the reconfigurable expanded region in reset by driving a reset source signal to the reset controllers for synchronization of that region into the appropriate clock domains.

After the new partial bitstream is downloaded and the accelerator device is ready to operate with the new binary, the device driver clears the `gate_pr` register, causing downstream reset assertions to be synchronously removed from the reconfigurable expanded region and the static base region boundary flip-flops.

The following figure shows the `pr_isolation_expanded` static base region level of sub-hierarchy.

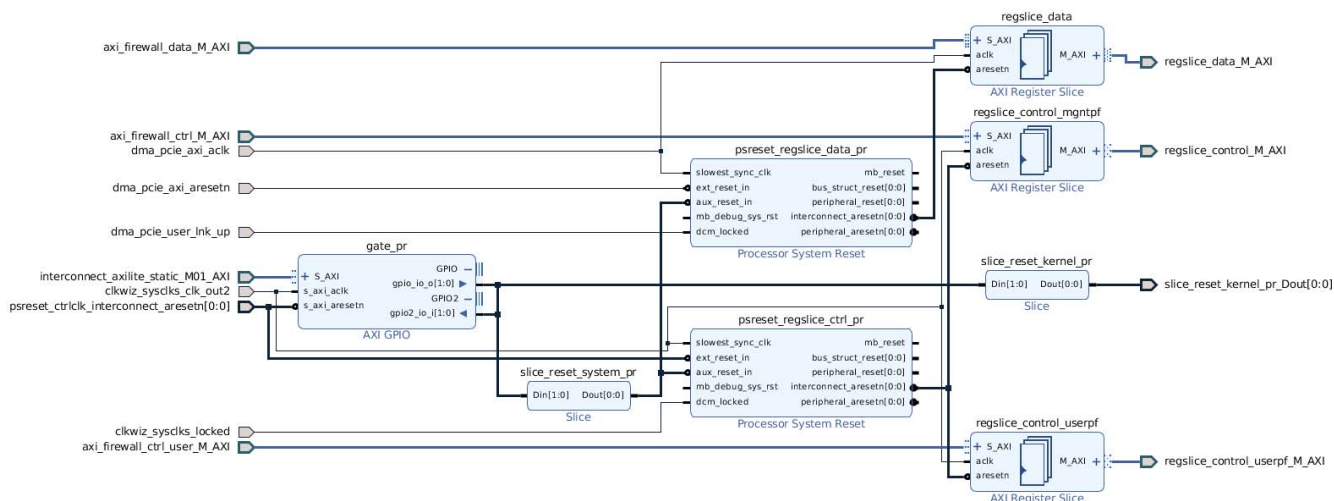


Figure 3-16: Partial Reconfiguration Support Logic in `pr_isolation_expanded` Sub-Hierarchy

Software Platform

Introduction

The Xilinx® Acceleration KCU1500 4DDR Expanded Partial Configuration platform is a memory-mapped system with PCIe® host connectivity supported by a kernel mode DMA driver for the XDMA IP. A hardware abstraction layer (HAL) driver isolates the SDAccel™ Software Platform runtime software from the implementation details of the kernel mode drivers, which interact with the platform hardware based on a known address mapping.

This chapter provides the address map and the software layers for the design.

Address Map

The IP integrator block diagram specifies the following address map for the IP cores in the reference design. The `xlcdma` driver defines these offsets in its header files.

Table 4-1: Reference Design Address Map

Master IP Core	AXI Master Interface	Slave IP Core	AXI Slave Interface	Offset Address	Range	High Address
XDMA, through <code>dma_pf_demux</code>	S_AXI_MGNTPF AXI4-Lite control interface for Management Physical Function	<code>u_ocl_region</code> SDAccel OpenCL Programmable Region	S_AXI	0x0000_0000	128K	0x0001_FFFF
		<code>axi_hwicap</code> AXI HWICAP for partial bitstream download	S_AXI_LITE	0x0002_0000	64K	0x0002_FFFF
		<code>gate_pr</code> GPIO for partial reconfiguration isolation	S_AXI	0x0003_0000	4K	0x0003_0FFF

Table 4-1: Reference Design Address Map (Cont'd)

Master IP Core	AXI Master Interface	Slave IP Core	AXI Slave Interface	Offset Address	Range	High Address
XDMA, through dma_pf_demux (continued)	S_AXI_MGNTPF AXI4-Lite control interface for Management Physical Function (continued)	scratchpad_ram_ctrl AXI BRAM Controller for driver scratchpad RAM	S_AXI	0x0003_1000	4K	0x0003_1FFF
		ddr_calib_status GPIO for DDR4 calibration status	S_AXI	0x0003_2000	4K	0x0003_2FFF
		flash_programmer QSPI flash memory controller	AXI_LITE	0x0004_0000	4K	0x0004_0FFF
		axi_i2c I2C controller for Kintex UltraScale KCU1500 Acceleration Developer Board fan	S_AXI	0x0004_1000	4K	0x0004_1FFF
		clkwiz_kernel Clocking Wizard kernel clock source	S_AXI_LITE	0x0005_0000	4K	0x0005_0FFF
		clkwiz_kernel2 Clocking Wizard kernel clock 2 source	S_AXI_LITE	0x0005_1000	4K	0x0005_1FFF
		ddrmem_0 DDR4 channel 0 controller	C0_DDR4_S_AXI_CTRL	0x0006_0000	64K	0x0006_FFFF
		ddrmem_2 DDR4 channel 2 controller	C0_DDR4_S_AXI_CTRL	0x0007_0000	64K	0x0007_FFFF
		ddrmem_3 DDR4 channel 3 controller	C0_DDR4_S_AXI_CTRL	0x0008_0000	64K	0x0008_FFFF

Table 4-1: Reference Design Address Map (Cont'd)

Master IP Core	AXI Master Interface	Slave IP Core	AXI Slave Interface	Offset Address	Range	High Address
XDMA, through dma_pf_demux (continued)	S_AXI_MGNTPF AXI4-Lite control interface for Management Physical Function (continued)	sys_mgmt_wiz System Management Wizard	S_AXI_LITE	0x000A_0000	64K	0x000A_FFFF
		feature_rom_ctrl AXI BRAM Controller for Feature ROM	S_AXI	0x000B_0000	4K	0x000B_0FFF
		debug_bridge_xvc Debug Bridge for Xilinx Virtual Cable	S_AXI	0x000C_0000	64K	0x000C_FFFF
		axi_firewall_ctrl AXI Firewall to protect Management Physical Function control path	S_AXI_CTL	0x000D_0000	64K	0x000D_FFFF
		axi_firewall_ctrl_user AXI Firewall to protect User Physical Function control path	S_AXI_CTL	0x000E_0000	64K	0x000E_FFFF
		axi_firewall_data AXI Firewall to protect data path	S_AXI_CTL	0x000F_0000	64K	0x000F_FFFF
		xilmonitor_apm AXI Performance Monitor for application profiling	S_AXI	0x0010_0000	64K	0x0010_FFFF

Table 4-1: Reference Design Address Map (Cont'd)

Master IP Core	AXI Master Interface	Slave IP Core	AXI Slave Interface	Offset Address	Range	High Address
XDMA, through dma_pf_demux (continued)	S_AXI_MGNTPF AXI4-Lite control interface for Management Physical Function (continued)	xilmonitor_fifo0 Trace offload FIFO for application profiling	S_AXI	0x0011_0000	4K	0x0011_0FFF
	S_AXI_USERPF AXI4-Lite control interface for User Physical Function	u_ocl_region SDAccel OpenCL Programmable Region	S_AXI	0x0000_0000	128K	0x0001_FFFF
		feature_rom_ctrl AXI BRAM Controller for Feature ROM	S_AXI	0x000B_0000	4K	0x000B_0FFF
		debug_bridge_xvc Debug Bridge for Xilinx Virtual Cable	S_AXI	0x000C_0000	64K	0x000C_FFFF
		xilmonitor_apm AXI Performance Monitor for application profiling	S_AXI	0x0010_0000	64K	0x0010_FFFF
		xilmonitor_fifo0 Trace offload FIFO for application profiling	S_AXI	0x0011_0000	4K	0x0011_0FFF

Table 4-1: Reference Design Address Map (Cont'd)

Master IP Core	AXI Master Interface	Slave IP Core	AXI Slave Interface	Offset Address	Range	High Address
XDMA	M_AXI AXI memory-mapped data interface	ddrmem_0 DDR4 channel 0 controller	C0_DDR4_ S_AXI	0x0000_0000 - 0000_0000	4G	0x0000_0000_ FFFF_FFFF
		ddrmem_1 DDR4 channel 1 controller	C0_DDR4_ S_AXI	0x0000_0001 - 0000_0000	4G	0x0000_0001_ FFFF_FFFF
		ddrmem_2 DDR4 channel 2 controller	C0_DDR4_ S_AXI	0x0000_0002 - 0000_0000	4G	0x0000_0002_ FFFF_FFFF
		ddrmem_3 DDR4 channel 3 controller	C0_DDR4_ S_AXI	0x0000_0003 - 0000_0000	4G	0x0000_0003_ FFFF_FFFF
		xilmonitor_ fifo0 Trace offload FIFO for application profiling	S_AXI_ FULL	0x0000_0020 - 0000_0000	4G	0x0000_0020_ FFFF_FFFF
SDAccel OpenCL Programmable Region	M00_AXI AXI memory-mapped data interface	ddrmem_0 DDR4 channel 0 controller	C0_DDR4_ S_AXI	0x0_0000_00 00	4G	0x0_FFFF_ FFFF
	M01_AXI AXI memory-mapped data interface	ddrmem_1 DDR4 channel 1 controller	C0_DDR4_ S_AXI	0x1_0000_ 0000	4G	0x1_FFFF_ FFFF
	M02_AXI AXI memory-mapped data interface	ddrmem_2 DDR4 channel 2 controller	C0_DDR4_ S_AXI	0x2_0000_ 0000	4G	0x2_FFFF_ FFFF
	M03_AXI AXI memory-mapped data interface	ddrmem_3 DDR4 channel 3 controller	C0_DDR4_ S_AXI	0x3_0000_ 0000	4G	0x3_FFFF_ FFFF

Software Layers

The SDAccel runtime software is layered on top of a common low-level software interface called the hardware abstraction layer (HAL). The HAL driver provides APIs to runtime software which abstract the kernel mode driver details. The XDMA and management kernel mode drivers interface with the memory-mapped platform over PCIe. The following figure shows the layers of the Software Platform.

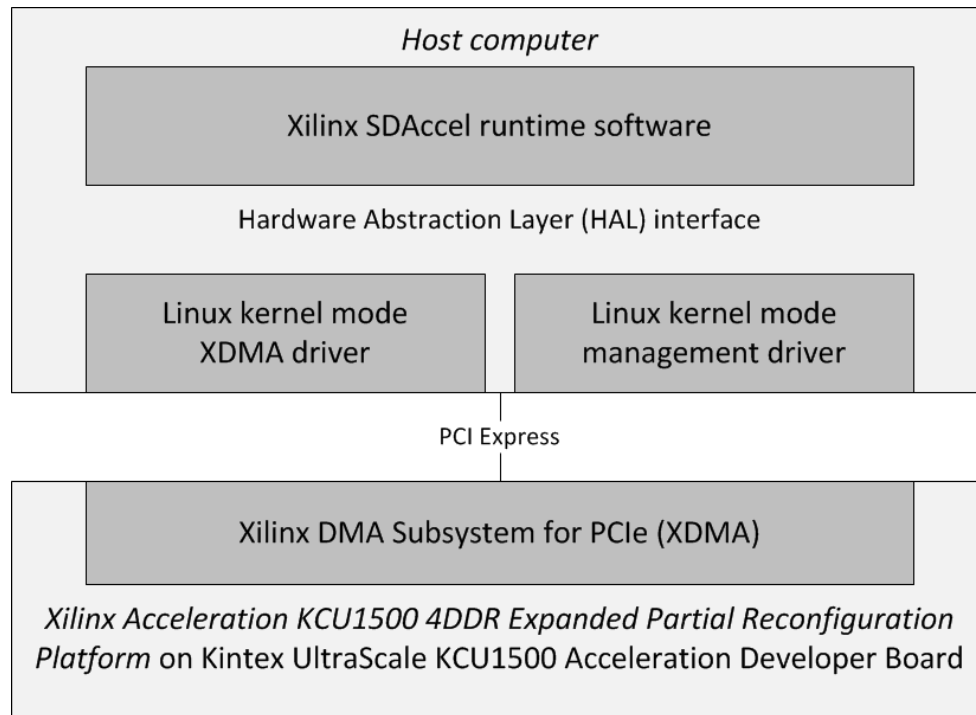


Figure 4-1: **Software Platform Layers**

Linux Kernel Mode Drivers

The Linux kernel mode drivers are included with the SDx Environments installation and are developed for the specified address mapping and memory-mapped IP functionality. When the platform is implemented as a DSA and used with the SDAccel Environment, the hardware abstraction layer (HAL) driver insulates the SDAccel runtime software from the implementation details of the kernel mode drivers.

If you do not use the SDx Environments or the provided HAL driver, you can still interact with the memory-mapped IP cores of the platform by using the provided kernel mode drivers.

As provided, the Xilinx Acceleration KCU1500 4DDR Expanded Partial Configuration platform uses an address map compatible with other Xilinx-provided platforms, so the kernel mode drives are not specific to this platform.

For more information about the common kernel mode drivers for Hardware Platforms targeting PCIe accelerator cards with XDMA IP, see the *SDAccel Environment Platform Development Guide* (UG1164) [Ref 2].



IMPORTANT: *User modifications to the address map, DMA, or host interface of the provided platform could necessitate a new or modified kernel mode driver.*

Hardware Abstraction Layer (HAL) Driver

The HAL driver is included with the SDx Environments installation and is required by the SDAccel runtime to communicate with the PCIe accelerator card. It is used for downloading FPGA bitstreams; allocating, deallocating, and migrating buffers; device profiling; and for communicating with the device and its kernels. The HAL driver is layered on top of the kernel mode drivers, and exposes C-style APIs to the runtime or other user of the driver.

The HAL driver is not specific to the Xilinx Acceleration KCU1500 4DDR Expanded Partial Configuration platform. For more information about the common HAL driver for Hardware Platforms that utilize the kernel mode drivers, see the *SDAccel Environment Platform Development Guide* (UG1164) [Ref 2].



IMPORTANT: *User modifications to the kernel mode drivers or replacement of a kernel mode driver could necessitate a new or modified HAL driver.*

Implementation

Introduction

Implementing the Xilinx® Acceleration KCU1500 4DDR Expanded Partial Reconfiguration platform in the Vivado® Design Suite is straightforward. The platform reference design includes modular scripts that are used with Vivado from the SDx™ Environments 2017.1 installation to construct the IP integrator block diagram, synthesize and implement the design, and produce a DSA file for use with the SDAccel Environment.

The individual scripts provided with the reference design are, as follows:

- `create_design.tcl`: Creates a Vivado project, sets the necessary properties and parameters, imports sources, and constructs the IP Integrator block diagram. The outcome of this script is a new project with constructed and validated block diagram, ready to synthesize.
- `run_synth.tcl`: Synthesizes the IP cores of the design, followed by the top level. This script is to be run after `create_design.tcl`.
- `run_impl.tcl`: Implements the full design, including optimization, placement, and routing. The script is to be run after `run_synth.tcl`.
- `write_dsa.tcl`: Creates the `.dsa` file for use in the SDAccel Environment from the implemented design. This script is to be run after `run_impl.tcl`.
- `run.tcl`: Optionally runs all of the above scripts in sequence.



IMPORTANT: Use the version of Vivado included with your SDx Environments 2017.1 installation. The standard Vivado 2017.1 release is not supported when building the Xilinx Acceleration KCU1500 4DDR Expanded Partial Reconfiguration platform.

There are two usage flow options for the scripts:

1. For an automated flow that involves no user interaction, from the Linux command line invoke the following command:

```
vivado -source run.tcl
```

Vivado proceeds through all steps and produces a `.dsa` file after some time.

2. For more opportunity to interact with Vivado, including the ability to refine the block diagram before running synthesis, to report timing between stages, and so forth; begin by invoking `vivado -source create_design.tcl` from the Linux command line.

This creates the new project and produces the validated block diagram. From within the same Vivado session, proceed with manually invoking `source run_synth.tcl`, `source run_impl.tcl`, and `source write_dsa.tcl` in sequence, after each stage and any additional desired operations have successfully completed.

Note: If you do not want to create a DSA for the SDAccel Environment you can simply generate a bitstream after `run_impl.tcl` completes, following option 2 above but forgoing the use of `write_dsa.tcl`.

The following figure illustrates the invocation of the two flows, their stages, as well as interim and final outputs. The blue shows the more automated option 1 usage flow (described above), and the red shows the more manual option 2 usage flow (described above).

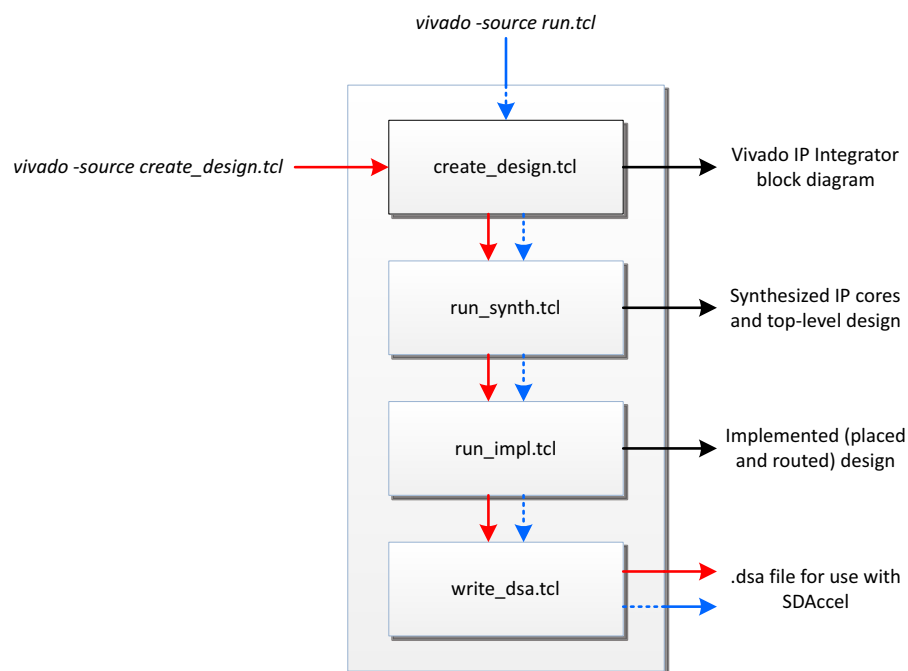


Figure 5-1: Platform Reference Design Implementation Script Usage Flow Options

create_design.tcl Detail

Some aspects of the `create_design.tcl` script benefit from further explanation. The following tables describe the purpose of some commands in the scripts.

Table 5-1: Properties and Parameters, from `create_design.tcl`

Command	Purpose
<code>set ::env(OCLOCK_BLOCK_ADVANCED) 1</code>	Enables advanced customization options of the SDAccel OpenCL Programmable Region IP core.
<code>set_param dsa.expandedPRRegion 1</code>	Required to enable the expanded partial reconfiguration flow in the DSA creation phase.
<code>set_param chipscope.enablePRFlow true</code>	Enables debug IP capabilities with partial reconfiguration in coordination with this version of Vivado.

Table 5-2: IP Repository Paths, from `create_design.tcl`

Command	Purpose
<code>set_property ip_repo_paths "\${sourcesDir}/iprepo/axi_perf_mon_v5_0"\ [current_project]</code>	Loads locally-modified version of the axi_perf_mon_v5_0 Xilinx IP core needed in this release for platform timing closure improvements.

Table 5-3: Project Properties Used for DSA Creation, from `create_design.tcl`

Command	Purpose
<code>set_property dsa.vendor "xilinx"\ [current_project]</code>	Sets the vendor field value in the DSA metadata.
<code>set_property dsa.board_id "kcu1500"\ [current_project]</code>	Sets the board_id field value in the DSA metadata.
<code>set_property dsa.name "4ddr-xpr"\ [current_project]</code>	Sets the name field value in the DSA metadata.
<code>set_property dsa.version "4.0"\ [current_project]</code>	Sets the version field value in the DSA metadata.
<code>set_property dsa.flash_offset_address\ "0x4000000" [current_project]</code>	Sets the flash_offset_address field value in the DSA metadata.

Table 5-3: Project Properties Used for DSA Creation, from `create_design.tcl` (Cont'd)

Command	Purpose
<code>set_property dsa.flash_interface_type \</code> <code>"spix8" [current_project]</code>	Sets the <code>flash_interface_type</code> field value in the DSA metadata.
<code>set_property dsa.description "This</code> platform targets the Kintex UltraScale KCU1500 Acceleration Development Board. This high-performance acceleration platform features four channels of DDR4-2400 SDRAM, the expanded partial reconfiguration flow for high fabric resource availability, and Xilinx DMA Subsystem for PCI Express with PCIe Gen3 x8 connectivity." [current_project]	Sets the <code>description</code> field value in the DSA metadata.

Table 5-4: Run properties used for platform implementation, from `create_design.tcl`

Command	Purpose
<code>set_property STEPS.OPT_DESIGN.TCL.PRE</code> <code>\${commonDir}/misc/xpr_preopt.tcl \</code> <code>[get_runs impl_1]</code>	Loads a <code>pre-opt_design</code> Tcl hook file needed in the implementation of this platform.
<code>set_property STEPS.OPT_DESIGN.TCL.POST</code> <code>\${sourcesDir}/misc/xpr_postopt.tcl \</code> <code>[get_runs impl_1]</code>	Loads a <code>post-opt_design</code> Tcl hook needed in the implementation of this platform.
<code>set_property STEPS.ROUTE_DESIGN.TCL.POST</code> <code>\${commonDir}/misc/xpr_postroute.tcl \</code> <code>[get_runs impl_1]</code>	Loads a <code>post-route_design</code> Tcl hook file needed in the implementation of this platform.

Design Constraints Detail

Some aspects of the `xilinx_kcu1500_4ddr-xpr_4_0.xdc` design constraints file benefit from further explanation. The following describes representative commands.

Some platform clock nets have the `HIGH_PRIORITY` property set to `true`. These commands instruct the implementation tools to spend additional effort on the identified paths such that they are more likely to meet timing, even in the presence of kernel clock timing failures, because only the latter can use automatic frequency scaling to compensate for setup violations. The following code snippet is an example:

```
set_property HIGH_PRIORITY true [get_nets
xcl_design_i/expanded_region/memc/ddrmem_0/inst/u_ddr4_infrastructure/div_clk]
```

The reconfigurable expanded region cell has `DONT_TOUCH` and `HD.RECONFIGURABLE` properties set to `true`, in support of partial reconfiguration of that region. The following code snippet is an example:

```
set_property DONT_TOUCH true [get_cells xcl_design_i/expanded_region]
set_property HD.RECONFIGURABLE true [get_cells xcl_design_i/expanded_region]
```

A Pblock is created, the reconfigurable expanded region logical hierarchy added to it, and then resized to include the ranges of physical resources that are used for that region. These ranges encompass the majority of the available physical resources on the device. The following code snippet is an example:

```
create_pblock pblock_expanded_region

{resize_pblock [get_pblocks pblock_expanded_region] \
-add SLICE_X123Y180:SLICE_X128Y299 SLICE_X119Y60:SLICE_X122Y299
SLICE_X118Y30:SLICE_X118Y119 SLICE_X97Y0:SLICE_X117Y119}

resize_pblock [get_pblocks pblock_expanded_region] \
-add {BITSLICE_CONTROL_X0Y8:BITSLICE_CONTROL_X1Y15}

...
```

A lower SLR Pblock is created, various IP cores that must be located in the lower SLR for reliable timing closure are added to it, and it is then resized to include those physical resources of the reconfigurable expanded region which are contained in the lower SLR of the device. Some submodules of the AXI SmartConnect IP instances are constrained to the lower SLR, as described in [Stacked Silicon Interconnect \(SSI\) Technology Support in Chapter 2](#). The following code snippet is an example:

```
create_pblock pblock_lower

add_cells_to_pblock [get_pblocks pblock_lower] [get_cells [list
xcl_design_i/expanded_region/interconnect_axilite]]

add_cells_to_pblock [get_pblocks pblock_lower] [get_cells [list
xcl_design_i/expanded_region/interconnect/interconnect_aximm_host]]

add_cells_to_pblock [get_pblocks pblock_lower] [get_cells [list
xcl_design_i/expanded_region/interconnect/interconnect_aximm_ddrmem0/inst/s00_axi2s
c]] -quiet

...

resize_pblock [get_pblocks pblock_lower] -add {SLICE_X123Y180:SLICE_X128Y299
SLICE_X119Y60:SLICE_X122Y299 SLICE_X118Y30:SLICE_X118Y119
SLICE_X97Y0:SLICE_X117Y119}

resize_pblock [get_pblocks pblock_lower] -add {DSP48E2_X22Y12:DSP48E2_X22Y47
DSP48E2_X18Y0:DSP48E2_X21Y47}
```

An upper SLR Pblock is created, various IP cores that must be located in the upper SLR for reliable timing closure are added to it, and it is then resized to include those physical resources of the reconfigurable expanded region which are contained in the upper SLR of the device.

Some submodules of some of the AXI SmartConnect IP instances are constrained to the upper SLR, as described in [Stacked Silicon Interconnect \(SSI\) Technology Support in Chapter 2](#). The following code snippet is an example:

```
create_pblock pblock_upper
```

```
add_cells_to_pblock [get_pblocks pblock_upper] [get_cells [list  
xcl_design_i/expanded_region/memc/axicc_ddrmem_2_ctrl]]  
  
add_cells_to_pblock [get_pblocks pblock_upper] [get_cells [list  
xcl_design_i/expanded_region/memc/axicc_ddrmem_3_ctrl]]  
  
add_cells_to_pblock [get_pblocks pblock_upper] [get_cells [list  
xcl_design_i/expanded_region/interconnect/interconnect_aximm_ddrmem2/inst/m00_exit_  
pipeline]] -quiet  
  
...  
  
resize_pblock [get_pblocks pblock_upper] -add {SLICE_X119Y300:SLICE_X142Y599}  
  
resize_pblock [get_pblocks pblock_upper] -add {RAMB18_X15Y120:RAMB18_X17Y239}
```

Install, Bring-Up, and Use

Introduction

When implemented with a produced DSA as described in [Chapter 5, Implementation](#), the Xilinx® Acceleration KCU1500 4DDR Expanded Partial Reconfiguration platform can be used with the SDAccel™ Environment on the Kintex® UltraScale™ KCU1500 Acceleration development board.

The following sections describe the installation, bring-up and use procedures, and assume a device support archive (DSA) created from an unmodified reference design, therefore comparable to the file in the SDx™ Environments 2017.1 installation, `xilinx_kcu1500_4ddr-xpr_4_0.dsa`.



IMPORTANT: *User modifications to the reference design can affect compatibility with the host, Kintex UltraScale KCU1500 Acceleration development board, or the SDAccel System Compiler flow, so confirming successful installation and bring-up is important.*

Installation

After the design is successfully implemented with closed timing, free of errors and critical warnings, and the `.dsa` file is written to disk, you can program the Kintex UltraScale KCU1500 Acceleration development board.

The installation steps are, as follows:

1. Install the board in a PCIe® Gen3 x8-compatible slot on the host computer.



IMPORTANT: *Beginning with the 2017.1 SDx Environments release, this reference design and the DSA it produces are compatible only with the new Kintex UltraScale KCU1500 Acceleration Development Board. They are not compatible with the previous revision of the board (termed Kintex UltraScale Developer Board for Acceleration with KU115), which has different FPGA pinouts and DDR4 memory configurations.*

2. Use the `xbinst` utility included with the SDx Environments installation to prepare board installation files, and program the configuration memory using the Vivado® Hardware Manager.
3. Install device drivers on the host computer. See the *SDx Environments Release Notes, Installation, and Licensing Guide* (UG1238) [Ref 1] for detailed instructions on the installation and setup process.

Note: When using the `xbinst` utility with a platform directory structure containing a DSA not included with the SDx Environments installation, such as one built from the reference design, the `-f` switch must be used to specify the location of the XPFM (`.xpfm`) file for that platform.

To determine the present operational status of the platform in more detail, including the HAL driver version, PCIe IDs, global memory details, FPGA temperature and voltage, compute unit status, and more, use the `xbsak` utility included with the directory that is created by running the `xbinst` utility as described above.

Note: The PCIe IDs (Vendor, Device, SDevice (Subsystem)) must match those indicated in [Host Connectivity in Chapter 3](#) for compatibility with the kernel mode drivers.

```
> ./xbinst/runtime/bin/xbsak query
INFO: Found 1 device(s)
DSA name:      xilinx:kcu1500:4ddr-xpr:4.0
HAL ver:      0.0
Vendor:       10ee
Device:       4b27
Device ver:   64
SDevice:     4340
SVendor:     10ee
DDR size:    0x10000000 KB
DDR count:   4
Data alignment: 64
DDR free size: 0x10000000 KB
Min xfer size: 64
OnChip Temp: 53 C
VCC INT:     930 mV
VCC AUX:     1783 mV
VCC BRAM:    931 mV
OCL Frequency:
    0:       300 MHz
    1:       500 MHz
PCIe:        GEN3 x 8
DMA threads: 4
MIG Calibrated: true
Firewall Last Error Status:
    0:       0x0 (GOOD)
    1:       0x0 (GOOD)
    2:       0x0 (GOOD)
CU Status:
    0:       0x4 (IDLE)
    1:       0x4 (IDLE)
    2:       0x4 (IDLE)
    3:       0x4 (IDLE)
INFO: xbsak query successful.
```

The `xbsak` utility also offers more options and is very useful in bringing up and debugging a platform.

For more details on the installation and bring-up process of an accelerator device that has been programmed using the SDx Environments, including more documentation on the `xbsak` utility, see the *SDx Environments Release Notes, Installation, and Licensing Guide* (UG1238) [Ref 1].

Bring-Up Tests

With the Kintex UltraScale KCU1500 Acceleration development board installed, DSA programmed, and the `xbsak` utility indicating compatibility and readiness of the device, it can be useful to compile and execute bring-up tests that use the SDAccel System Compiler flow. This process provides the next level of confidence that the platform is compatible and operational with partial reconfiguration, the SDAccel System Compiler flow and its implemented kernels, and the software stack.

Provided as source code with build scripts, the set of bring-up tests exercise low-level functionality of the accelerator device through their implementation as kernels in the SDAccel System Compiler flow, download as partial bitstreams to the operating platform, and communication with the host computer.

When compiled and run on the host computer containing the operational Kintex UltraScale KCU1500 Acceleration development board with programmed DSA, each test will run host executable code, communicate with the implemented kernel on the accelerator device, and report a pass/fail status.

See the bring-up test `README.txt` file, packaged with the reference design, for more information on building and running the tests.

Use with the SDAccel Environment

When the Kintex UltraScale KCU1500 Acceleration development board with programmed DSA is proven operational through successful installation and `xbsak` utility outputs, and when bring-up tests indicate compatibility with the SDAccel System Compiler, the DSA is ready to be used in the SDAccel Environment with user source code. See the *SDAccel Environment User Guide* (UG1023) [Ref 4] and the *SDAccel Environment Optimization Guide* (UG1027) [Ref 5] for in-depth documentation on using the SDAccel Environment.

The following information can also be helpful when targeting the `xilinx_kcu1500_4ddr-xpr_4_0.dsa` from the SDx Environments installation, or as built from the Xilinx Acceleration KCU1500 4DDR Expanded Partial Configuration platform reference design.

Device Identification

The vendor, build, name, version (VBNV) identifier for the DSA is `xilinx:kc1500:4ddr-xpr:4.0`, so an XOCC script (which invokes the SDAccel System Compiler) includes the following argument:

```
--xdevice xilinx:kc1500:4ddr-xpr:4.0
```

If attempting to target the DSA that is built from the platform reference design rather than one from the SDx Environments installation, it is necessary to specify the absolute path to the directory containing the built `.dsa` file, also known as the `device_repo_path`. Such `.dsa` files are used with priority over the installation `.dsa` files in the event of duplication. The XOCC script would contain:

```
--xp prop:solution.device_repo_paths=<path_to_dir_containing_dsa>
```

User-Specified Connectivity

The Programmable Region has four AXI memory-mapped master interfaces, each of which connect to one of the four DDR4 IP memory controllers. As described in [Sparse Memory Connectivity in Chapter 2](#), the kernel to global memory accessibility is based on user-defined connectivity.

The means of specifying that connectivity is through the use of the `map_connect` XOCC property, in the following format:

```
--xp
misc:map_connect=add.kernel.<kernel_name>.<kernel_MI>.core.OCL_REGION_0.<ocl_region_MI>
```

For example, to allow the AXI master interface `MY_MI_AXI` of kernel `my_kernel` access to global memory from the fourth of four DDR4 channels (`M03_AXI`, as described in [Chapter 3, Hardware Platform](#)), the XOCC script would contain:

```
--xp misc:map_connect=add.kernel.my_kernel.MY_MI_AXI.core.OCL_REGION_0.M03_AXI
```

These commands are additive, such that any master interface of any kernel can be connected to any Programmable Region master interface, in many-to-many fashion, thus allowing arbitrary kernel-to-global memory accessibility; however, more connectivity implies more resource utilization, and can challenge timing closure.



RECOMMENDED: *Users are recommended to consider this trade-off and implement only the connectivity which is necessary for their application and performance requirements.*

More information on `map_connect` can be found in the *SDAccel Environment User Guide* (UG1023) [Ref 4] and the *SDAccel Environment Optimization Guide* (UG1027) [Ref 5].

Clock Frequency

By default, the kernel clock of the platform runs at 300 MHz, and the optional kernel clock 2 runs at 500 MHz. Unless kernel clock 2 is explicitly used by an RTL kernel, all kernel data path logic is synchronous to kernel clock. See platform clocking for details.

To override the default kernel clock frequency with a different frequency specification, the XOCC script would contain the following switch format:

```
--kernel_frequency <freq_in_mhz>
```

For example, to specify that the kernel clock frequency should target 200 MHz rather than the default of 300 MHz:

```
--kernel_frequency 200
```

If kernel clock 2 is used by the RTL kernel, then the switch takes a key-value format, where kernel clock is identified by key 0 and kernel clock 2 is identified by key 1.

For example, to specify that the kernel clock frequency should target 250 MHz rather than the default of 300 MHz, and that kernel clock 2 should target 400 MHz rather than the default of 500 MHz, as follows:

```
--kernel_frequency 0:250|1:400
```

To instruct the SDAccel System Compiler to use the optional kernel clock 2, a user's RTL kernel must implement the following input port naming convention. The user is responsible for ensuring proper clock domain crossing from the kernel clock to kernel clock 2 in the ingress direction (slave interface), and from kernel clock 2 to kernel clock in the egress direction (master interface(s)).

- `ap_clk`, which the SDAccel System Compiler will drive with the kernel clock
- `ap_rst_n`, which the SDAccel System Compiler will drive with the active-low reset synchronous to kernel clock
- `ap_clk_2`, which the SDAccel System Compiler will drive with the kernel clock 2
- `ap_rst_n_2`, which the SDAccel System Compiler will drive with the active-low reset synchronous to kernel clock 2

See the *SDAccel Environment User Guide* (UG1023) [Ref 4] and the *SDAccel Environment Optimization Guide* (UG1027) [Ref 5] for kernel frequency specification, RTL kernels, and related topics.

Timing Closure

The platform contains many clock domains. With the exception of setup violations on the kernel clock domain (and if used, also kernel clock 2 domain) intra-clock paths, all paths must meet timing with each compiler run. Failure to do so results in the flow reporting a timing failure and not producing an `.xclbin` file.

In the event that only kernel clock paths do not meet their timing requirement and contain only setup violations, the flow will report a message similar to the following:

```
WARNING: [XOCC 60-732] Link warning: One or more timing paths failed timing targeting
300 MHz. This design may not work properly on the board with this target frequency.
The frequency is being automatically changed to 240 MHz
```

When the application is executed, the SDAccel runtime will automatically change the operating frequency of the kernel clock to the reported lower frequency in order to compensate for the setup violation. This automatic frequency scaling feature allows user kernels to operate in hardware, even if at a lower frequency than intended.

As described in [Expanded Partial Reconfiguration in Chapter 2](#), each compiler run places and routes user kernels together with the remainder of the logic in the reconfigurable expanded region. While expanded partial reconfiguration has clear benefits, a potential side effect of the approach is that paths in clock domains other than the kernel clock can sometimes fail their timing requirement if there are significant kernel timing violations. This behavior is simply a result of how the Vivado tools prioritize timing closure of critical paths. The likelihood of such violations, which prevent `.xclbin` generation, is reduced through the use of `HIGH_PRIORITY` clock properties as described in [Design Constraints Detail in Chapter 5](#).

To reduce the likelihood of large kernel timing violations affecting other paths and thus preventing automatic frequency scaling, users are advised to specify an achievable kernel clock frequency if all kernel logic is not likely to meet the default of 300 MHz - a process which might be iterative. For example, if a user kernel is likely to achieve timing closure at a maximum of 150 MHz, you should specify such a frequency target:

```
--kernel_frequency 150
```

In this way, kernel developers can decouple preliminary hardware operation from optimization of kernel code to achieve higher frequencies.

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

1. *SDx Environments Release Notes, Installation, and Licensing Guide* ([UG1238](#))
 2. *SDAccel Environment Platform Development Guide* ([UG1164](#))
 3. *DMA Subsystem for PCI Express Product Guide* ([PG195](#))
 4. *SDAccel Environment User Guide* ([UG1023](#))
 5. *SDAccel Environment Optimization Guide* ([UG1207](#))
 6. *SDAccel Environment Tutorial: Introduction* ([UG1021](#))
-

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.

PCI, PCIe, and PCI Express are trademarks of PCI-SIG and used under license.

Copyright 2017 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Vivado, Zynq, UltraScale, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.