

SDAccel Platform Reference Design User Guide

Developer Board for Acceleration with KU115

UG1234 (v2016.3) November 30, 2016

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
11/30/2016	2016.3	Initial Xilinx release.

Table of Contents

Revision History	2
Chapter 1: Overview	
Introduction	5
Platform Features	5
The Platform Reference Design	6
Chapter 2: Platform Characteristics	
Expanded Partial Reconfiguration	7
Use with the SDAccel Environment	9
Sparse Memory Connectivity	10
Stacked Silicon Interconnect (SSI) Technology Support	11
Chapter 3: Hardware Platform	
Host Connectivity	14
Memory Control	16
SDAccel OpenCL Programmable Region IP and the Programmable Region	18
AXI Interconnectivity	22
Application Profiling and Other Features	25
Clocking and Reset	26
Chapter 4: Software Platform	
Address map	30
Software Layers	32
Chapter 5: Implementation	
<code>create_design.tcl</code> Detail	36
Design Constraints Detail	38
Chapter 6: Install, Bring-Up, and Use	
Installation	40
Bring-Up Tests	45
Use with the SDAccel Environment	46

Appendix A: Additional Resources and Legal Notices

Xilinx Resources	49
Solution Centers.	49
Documentation Navigator and Design Hubs	49
References	50
Please Read: Important Legal Notices	50

Overview

Introduction

This document describes the platform reference design for the Xilinx Developer Board for Acceleration with KU115, 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 may also adapt the reference design to your own needs.

The specific platform reference design described in this document may be referred to as the Xilinx Acceleration KU115 4DDR Expanded Partial Reconfiguration platform, because it targets the Xilinx Developer Board for Acceleration with 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 and available with SDx™ Environments 2016.3 is named `xilinx:xil-accel-rd-ku115:4ddr-xpr:3.2`, and the file is this:

```
xilinx_xil-accel-rd-ku115_4ddr-xpr_3_2.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.

Platform Features

The features of the Xilinx Acceleration KU115 4DDR Expanded Partial Reconfiguration platform reflect the capabilities and characteristics of the Xilinx Developer Board for Acceleration with KU115, and its intended use as a high-performance acceleration platform for the SDAccel Environment:

- 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 Xilinx Developer Board for Acceleration with KU115 supports Gen3 x8 connectivity.

- Four UltraScale Memory IP instances enable access to available DDR4 SDRAM. The Xilinx Developer Board for Acceleration with KU115 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 and 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.
- Support for SDx Environments application profiling via trace offload hardware infrastructure, and for flash memory programming via SPI flash IP infrastructure.

The Platform Reference Design

The platform reference design includes the necessary design sources, scripts, and instructions to build the Xilinx Acceleration KU115 4DDR Expanded Partial Reconfiguration platform. The modular scripts are used with Vivado from the SDx Environments 2016.3 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. Users who wish to modify and adapt the design would construct the IP integrator block diagram, make the desired changes, then continue with synthesis, implementation, and DSA creation. These individual steps and the scripts to run them are described in the instructions, available as part of the [Reference Design Files](#), which you can download from the Xilinx website.

Device drivers are not supplied with the reference design, but are available with the SDx Environments installation and are discussed 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) [\[Ref 2\]](#).

Platform Characteristics

The Xilinx Acceleration KU115 4DDR Expanded Partial Reconfiguration platform is a high-performance acceleration platform for the SDAccel™ Environment. Its distinguishing characteristics are as follows.

Expanded Partial Reconfiguration

Many DSAs use partial reconfiguration to enable compiled binary downloads to the accelerator device while it remains online and linked to the host's PCIe bus. The Xilinx Acceleration KU115 4DDR Expanded Partial Reconfiguration platform is also designed for partial reconfiguration, but provides a larger, "expanded region" that makes over 80% of the KU115 device's overall fabric resources available to the kernels. Only the logic necessary to 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. 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. 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.

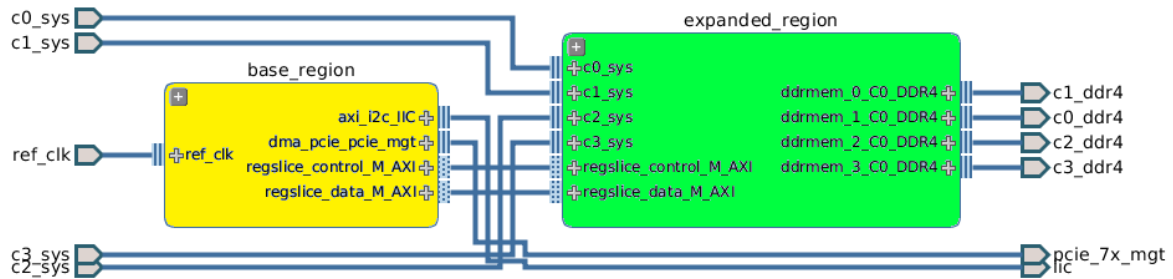


Figure 2-1: IP Integrator Top-Level View of Platform, Showing Base and Expanded Regions

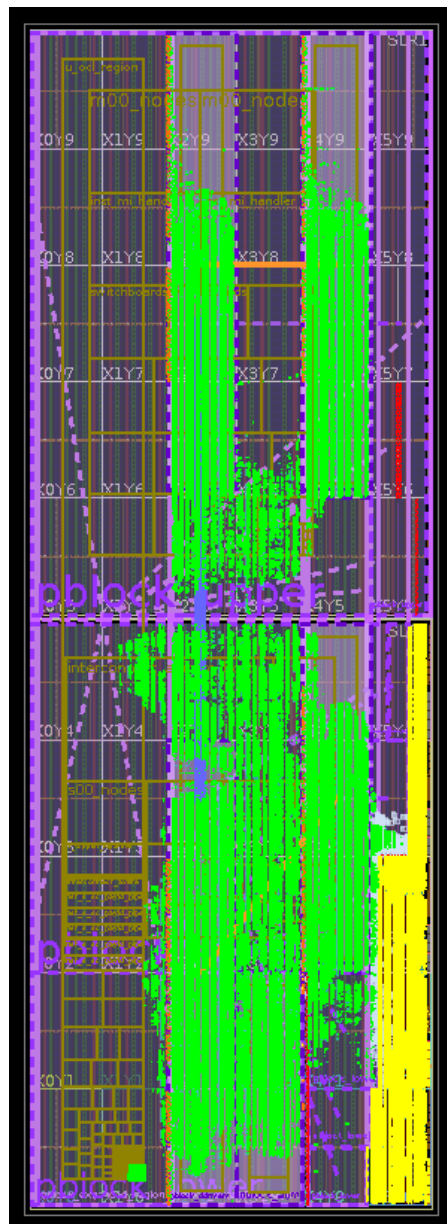


Figure 2-2: Device View of Implemented Platform, Showing Base and Expanded Regions

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 DMA driver for the Xilinx DMA Subsystem for PCI Express (herein referred to as the `xcldma` driver) is likewise provided with the SDx Environments installation.
- **Global memory access** - DDR4 SDRAM global memory is accessible to both the host and user kernels via 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 KU115 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.

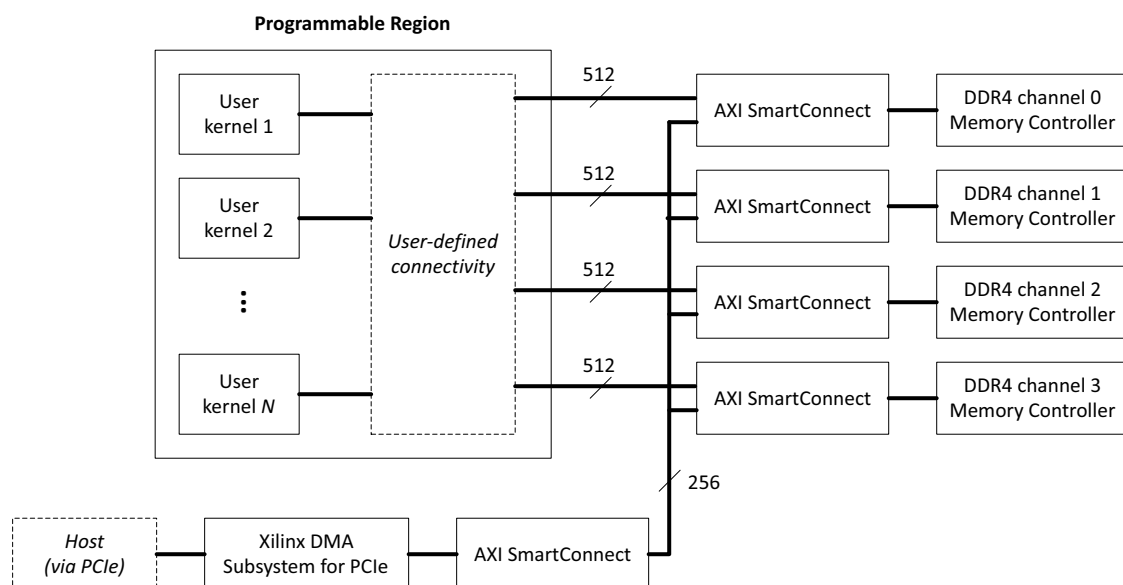


Figure 2-3: Sparse Memory Connectivity with User-Defined Kernel to Global Memory Connectivity

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 which cross from one SLR to another can be significant, and make timing closure challenging in highly connected designs such as the Xilinx Acceleration KU115 4DDR Expanded Partial Reconfiguration platform. The AXI data paths used throughout the platform are wide and generally include combinatorial paths to implement the AXI-4 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 which 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 all 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.

Figure 2-4 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 line outlines represent pipeline stages which connect to the Programmable Region and which, like the Programmable Region, are not floorplanned and may 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.

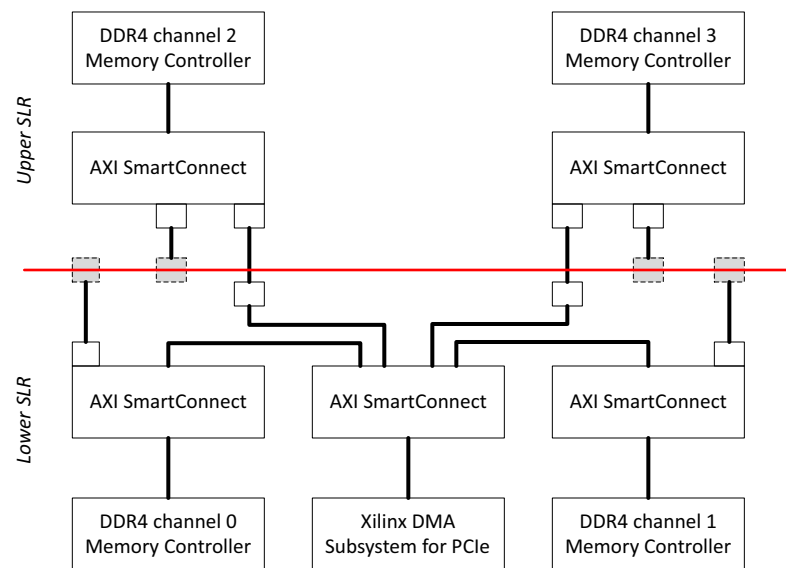


Figure 2-4: Simplified Representation of IP Partitioning for Controlled SLR Crossing

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

In this chapter, the hardware architecture of the Xilinx Acceleration KU115 4DDR Expanded Partial Reconfiguration platform is described from the perspective of its Vivado® IP Integrator block diagram representation, in six parts.

- [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. The two views are shown below in Figures [Figure 3-1](#) and [Figure 3-2](#), respectively. The remainder of this chapter makes use of both views as appropriate.

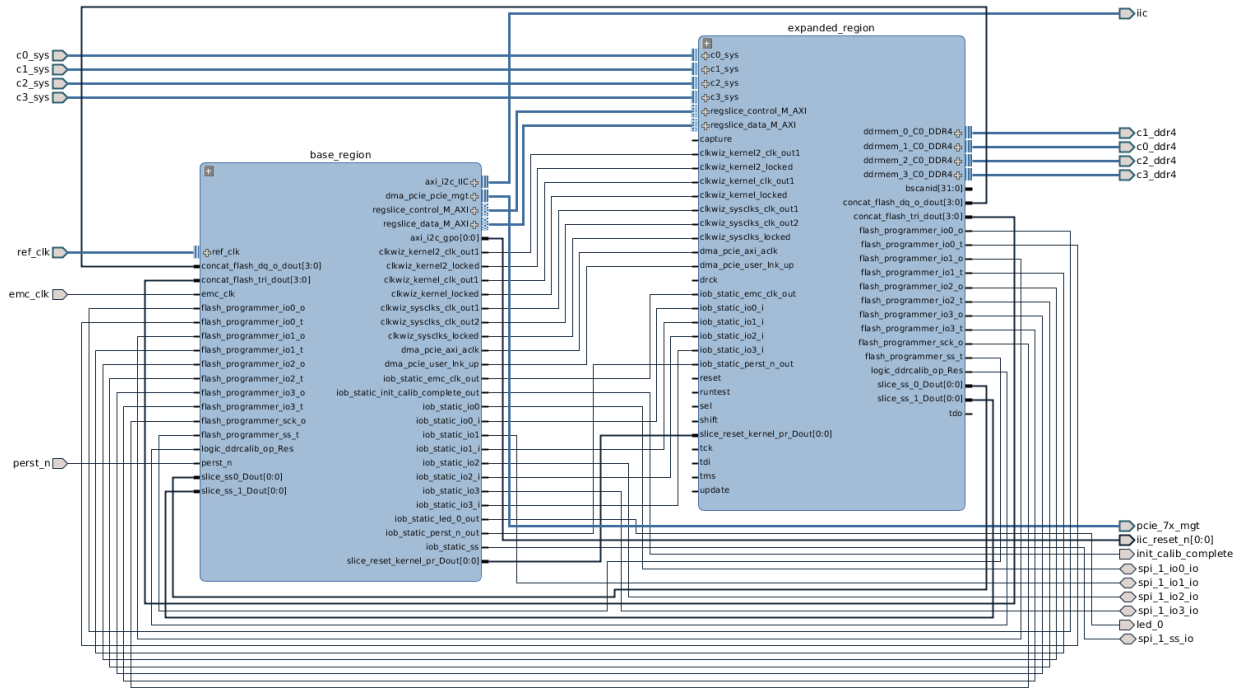


Figure 3-1: Top-Level IP Integrator Block Diagram - Default View with Interfaces and Wires

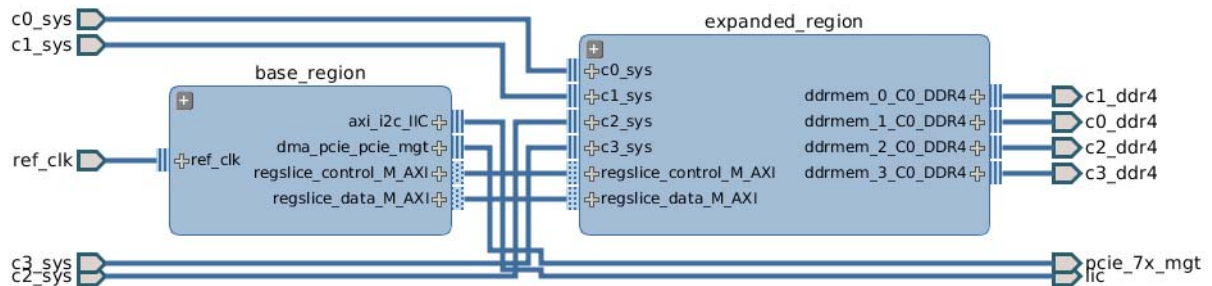


Figure 3-2: Top-level IP Integrator Block Diagram - Interface Connections Only View

Host Connectivity

The platform implements connectivity to the host computer via the Xilinx DMA Subsystem for PCI Express (PCIe) IP, which contains a Scatter Gather DMA and PCI Express 3.x Integrated Block. For best performance on the Xilinx Developer Board for Acceleration with KU115, the PCIe Integrated Block can negotiate a link up to Gen3 x8 connectivity, or 8.0 GT/s. The Xilinx Developer Board for Acceleration with KU115 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 as shown in Figure 3-3.

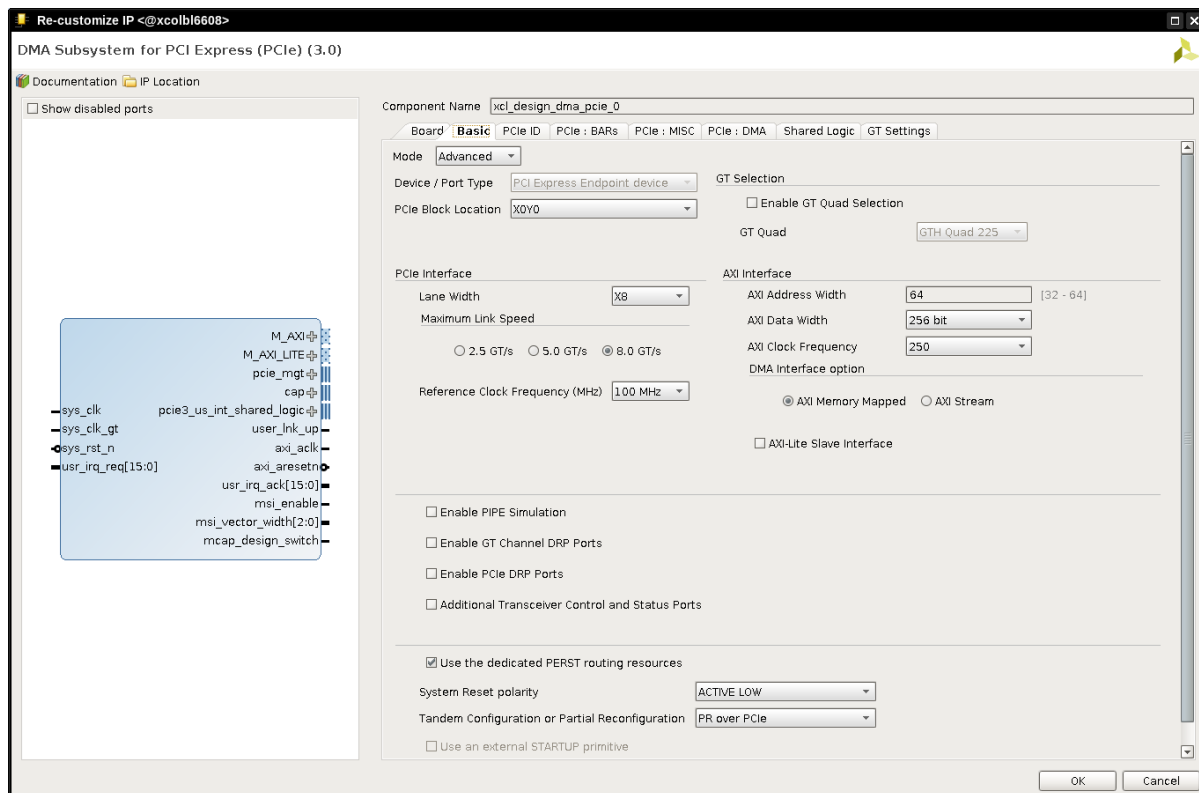


Figure 3-3: XDMA IP customization - Basic Tab

For compatibility with the provided `xcl_dma` and HAL drivers, the XDMA IP instance is customized to use Vendor ID 0x10EE (Xilinx Vendor ID), Device ID 0x8238, Subsystem Vendor ID 0x10EE, and Subsystem ID 0x4432. Derived platforms with user modifications should 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 Figure 3-4, 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.

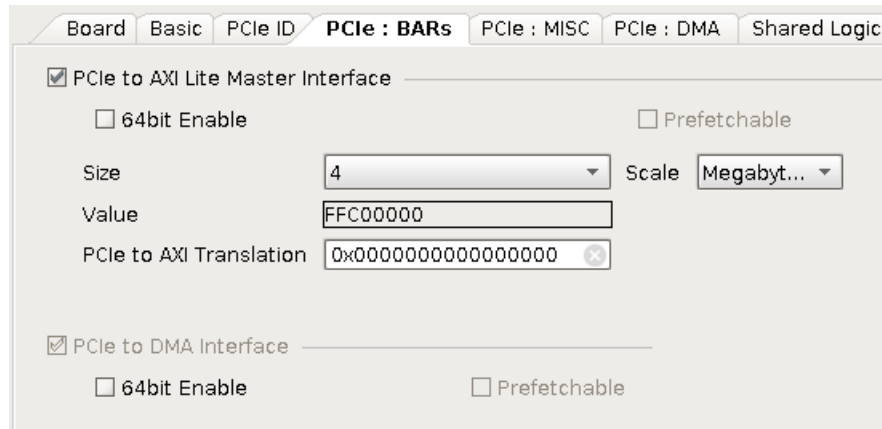


Figure 3-4: XDMA IP customization - PCIe: BARs Tab

As shown in Figure 3-5, 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.

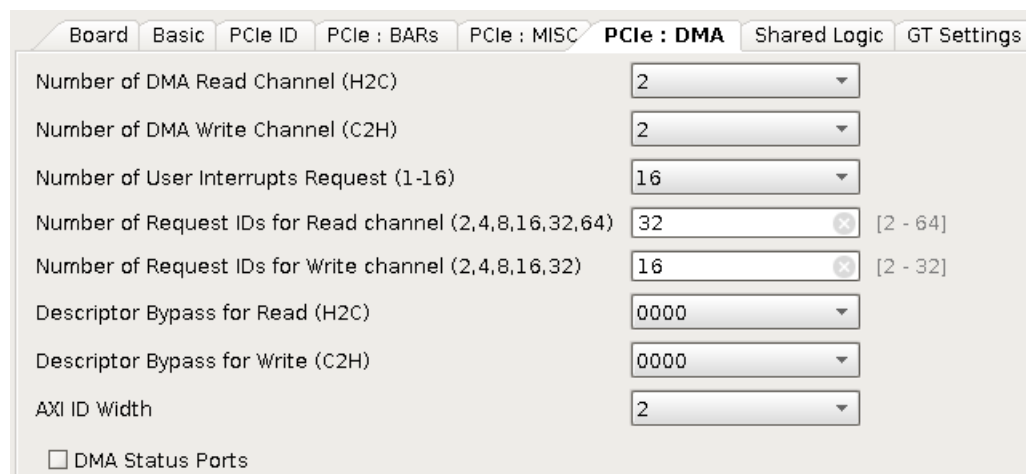


Figure 3-5: XDMA IP Customization - PCIe: DMA Tab

Refer to the XDMA instance's IP Customization GUI 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 Xilinx Developer Board for Acceleration with KU115 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 that channel's memory controller.

The DDR4 IP instances are customized to target the MT40A512M16HA-083E components on the Xilinx Developer Board for Acceleration with KU115, 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](#), 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 [Figure 3-6](#).

Note: Three of the four DDR4 IP instances are identical and enable ECC. One of the board's DDR4 channels does not support ECC.

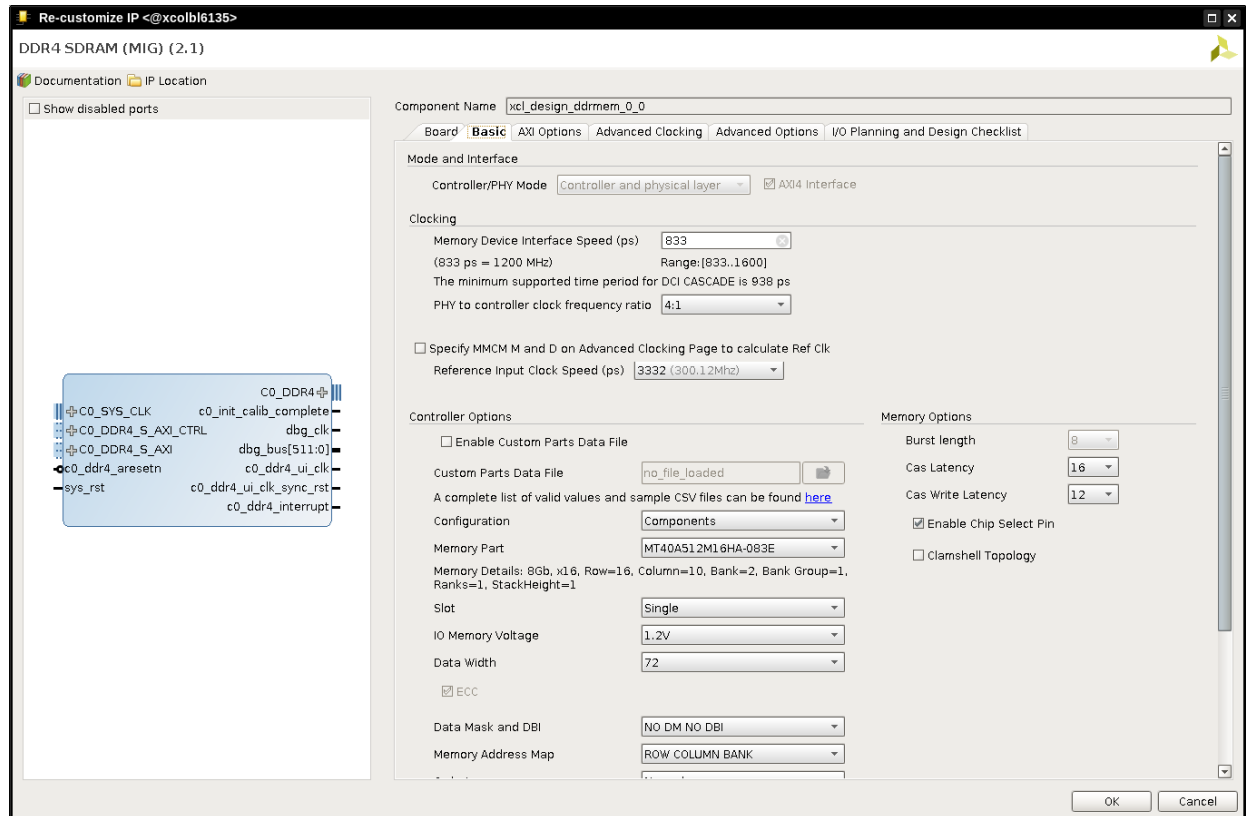


Figure 3-6: DDR4 IP Customization - Basic Tab

Contained within the memc sub-hierarchy of the reconfigurable expanded region as shown in [Figure 3-7](#), the four DDR4 IP instances are individually accessed, and individually indicate their calibration status via the `c0_init_calib_complete` output port. The logical AND of these signals is available to the device driver via memory mapped GPIO IP instance in the static base region.

As introduced in [Stacked Silicon Interconnect \(SSI\) Technology Support](#), the `ddrmem_0` and `ddrmem_1` instances are located in the KU115 device's lower SLR, while `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.

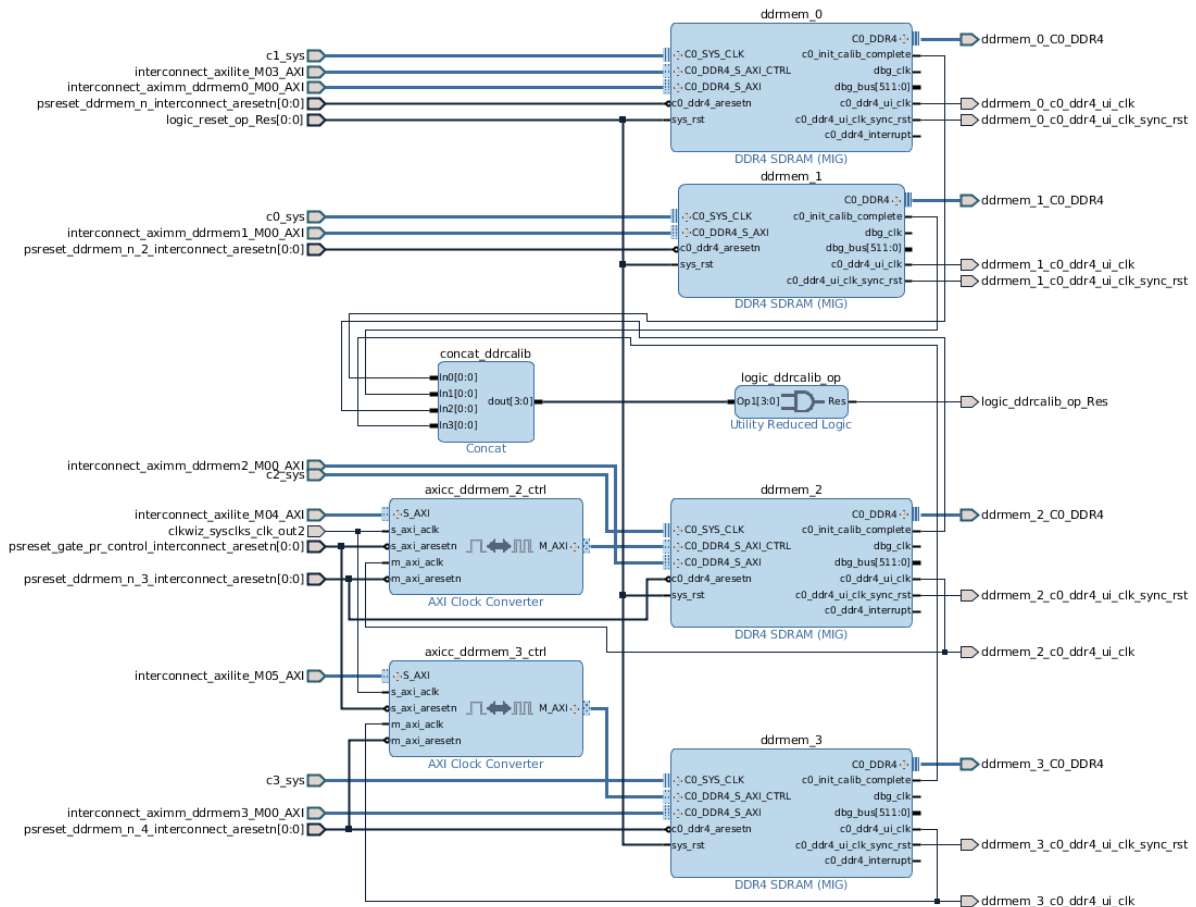


Figure 3-7: DDR4 IP Instances and Supporting IP in Reconfigurable Expanded Region memc Sub-Hierarchy

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 platform's IP Integrator block diagram 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](#), 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. Since the expanded region is a superset of the Programmable Region and other platform logic, each SDAccel System Compiler run places and routes 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.

[Figure 3-8](#) shows the SDAccel OpenCL Programmable Region IP core instance. 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 which indirectly connect to the four DDR4 IP memory controller instances as previously described in [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. However, data paths must be synchronous to the `DATA_CLK` and `DATA_RESET` domain on both the input and output interfaces of the kernel(s), making the RTL kernel developer responsible for transferring data from the `CONTROL_CLK` domain into the `KERNEL_CLK2` domain on ingress, and from the `KERNEL_CLK2` domain into the `CONTROL_CLK` domain on egress of the kernel.

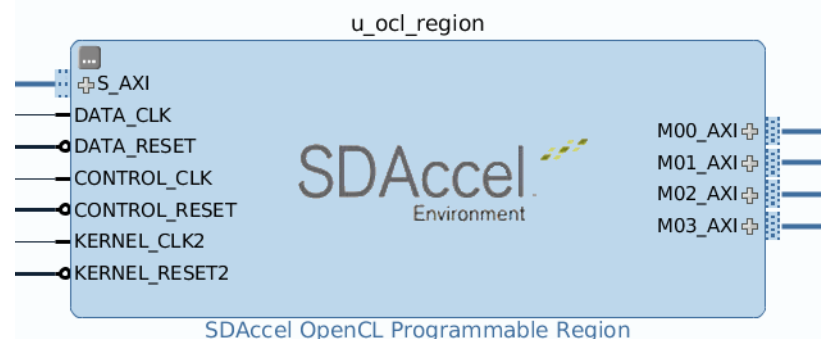


Figure 3-8: SDAccel OpenCL Programmable Region IP Core Instance and its Interfaces

The SDAccel OpenCL Programmable Region IP core is a hierarchical IP. Viewing its contents reveals the subsystem block diagram shown in [Figure 3-9](#). 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 via 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.

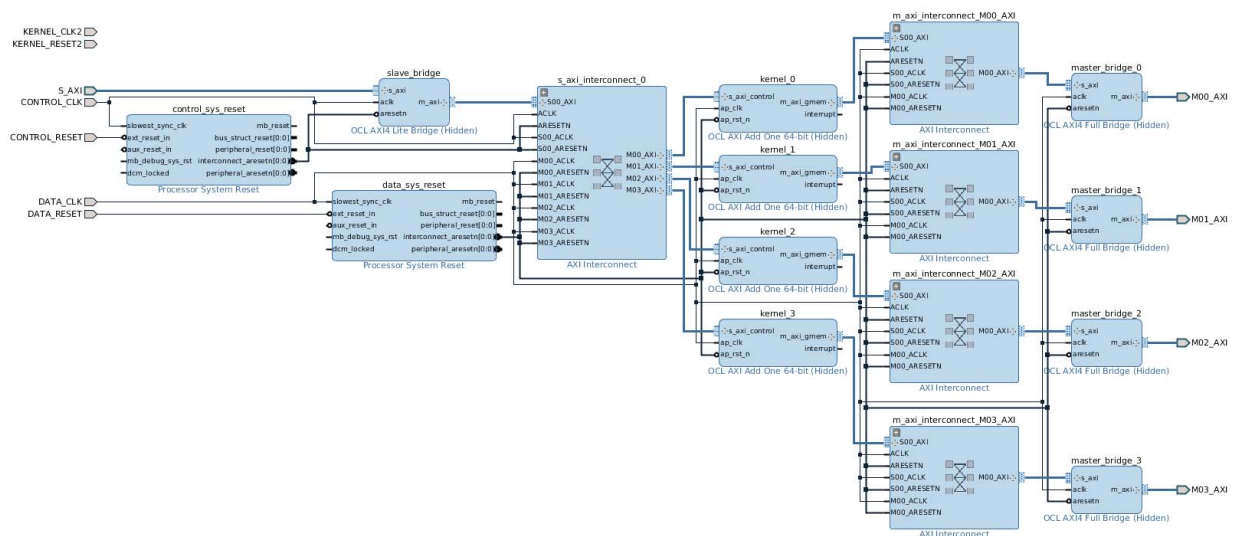


Figure 3-9: SDAccel OpenCL Programmable Region Hierarchical IP Subsystem

The SDAccel OpenCL Programmable Region IP instance customization is shown in [Figure 3-10](#) and [Figure 3-11](#), below.

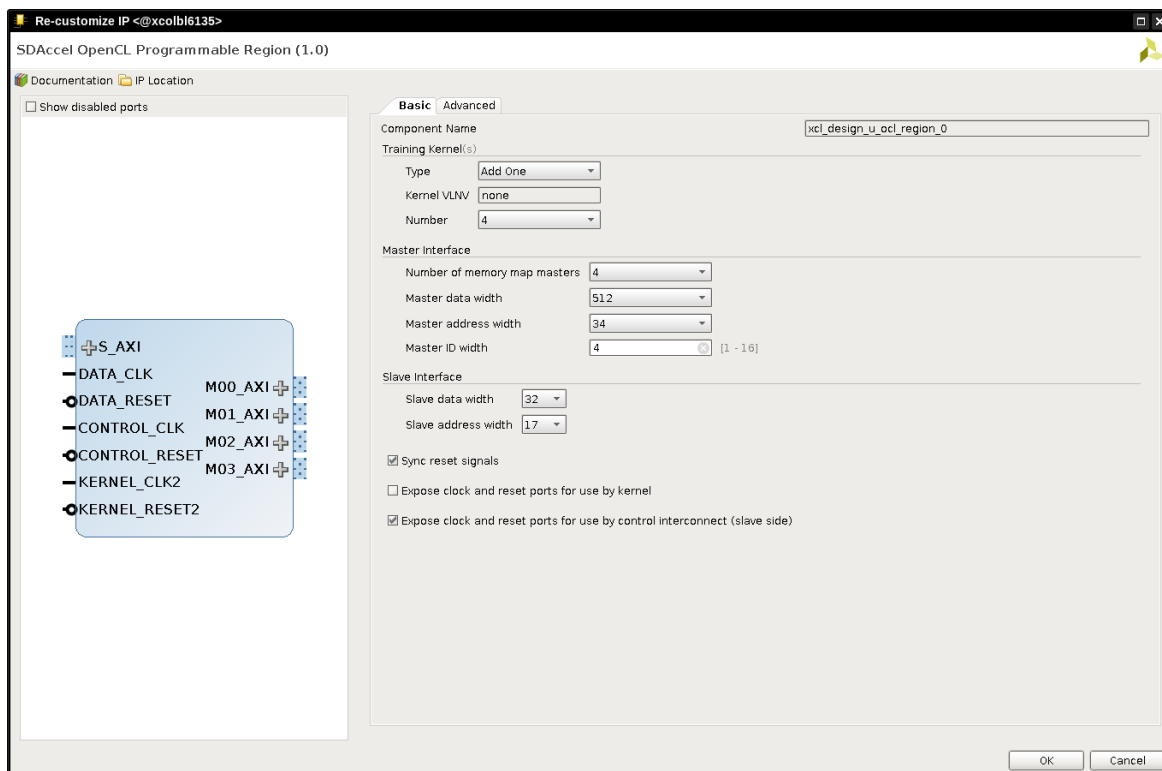


Figure 3-10: SDAccel OpenCL Programmable Region IP Customization - Basic Tab

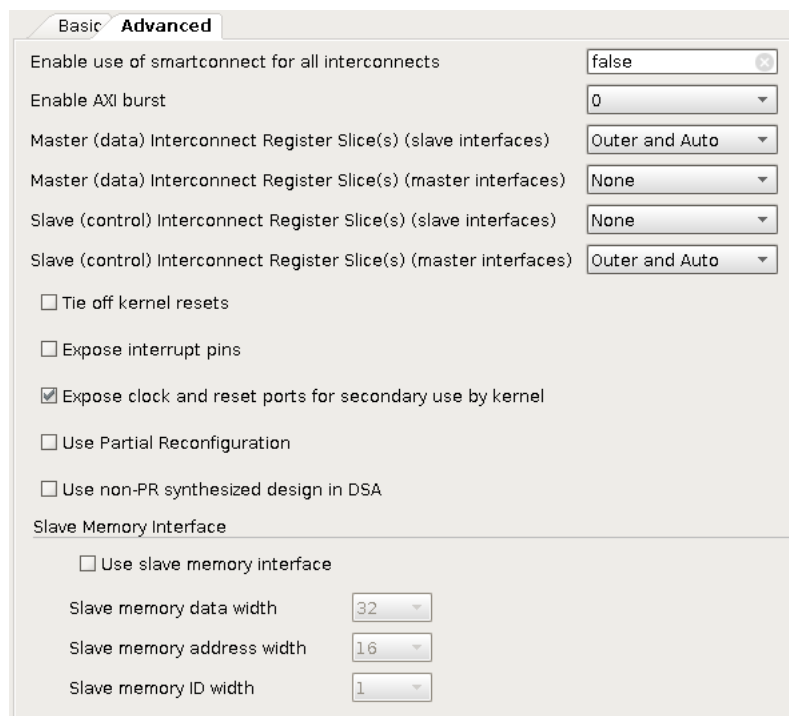


Figure 3-11: SDAccel OpenCL Programmable Region IP Customization - Advanced Tab

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 (via XDMA IP instance) to all global memory (via individual DDR4 IP instances), and from the user kernels to user-defined regions of global memory. High-performance data path connectivity is implemented via five instances of AXI SmartConnect IP, with the topology previously described in [Sparse Memory Connectivity](#).

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

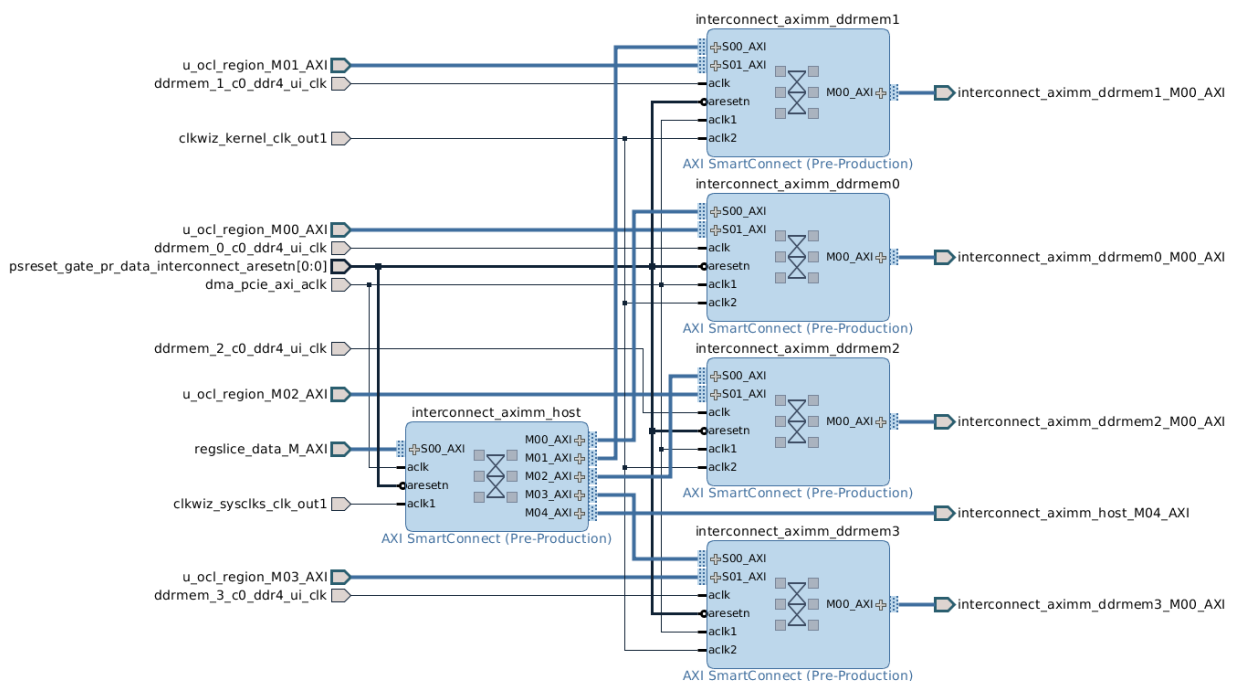


Figure 3-12: AXI SmartConnect IP Instances in Reconfigurable Expanded Region Interconnect Sub-Hierarchy

The `interconnect_aximm_host` instance of SmartConnect IP is customized to use 1 slave interface and 5 master interfaces - that is, a 1x5 customization - where each interface uses a 256-bit data path. The `S00_AXI` slave interface is connected to the XDMA IP via 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 - that is, 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 `ac1k1` 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 `ac1k2` port. The `M00_AXI` interface of each is connected to the corresponding DDR4 IP instance, and therefore the `ac1k` port of each instance is driven by the connected DDR4 IP instance's AXI interface clock. See [Sparse Memory Connectivity](#) for context.

Host access via XDMA IP is the single control path master of the platform. The low-speed, 32-bit data path AXI4-Lite control interface is partitioned into two parts, corresponding to the static base region and reconfigurable expanded region. As shown in [Figure 3-13](#), the static base region contains a 1x7 AXI Interconnect IP instance providing the XDMA IP master with memory-mapped access to 6 peripherals, and a connection (`M00_AXI`) to the reconfigurable expanded region's AXI Interconnect IP instance via AXI Register Slice IP for partial reconfiguration isolation (see [Partial Reconfiguration Isolation](#) for details).

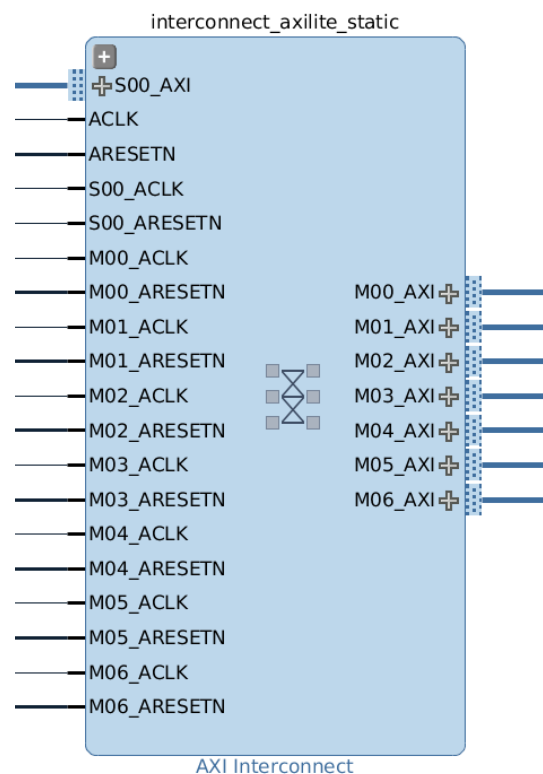


Figure 3-13: Static Base Region 1x7 AXI Interconnect Instance for AXI4-Lite Control Path

The reconfigurable expanded region also contains a 1x7 AXI Interconnect IP instance. Its `S00_AXI` slave interface is connected to the static region via partial reconfiguration isolation AXI Register Slice as described above, and its 7 master interfaces connect to a variety of

memory-mapped endpoints including the AXI Performance Monitor, DDR4 IP, and user kernels in the Programmable Region. To further ease timing closure, the AXI Interconnect instance in the reconfigurable expanded region enables the optional register slice on each interface, as shown in Figures [Figure 3-14](#) and [Figure 3-15](#).

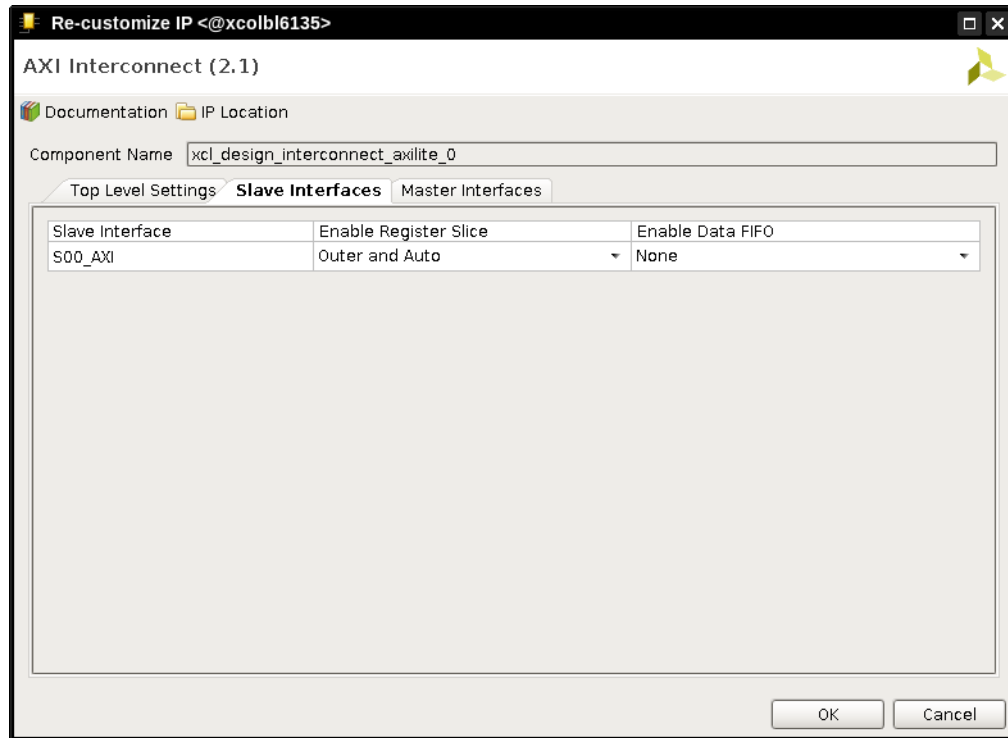


Figure 3-14: Reconfigurable Expanded Region AXI Interconnect IP Customization - Slave Interfaces Tab

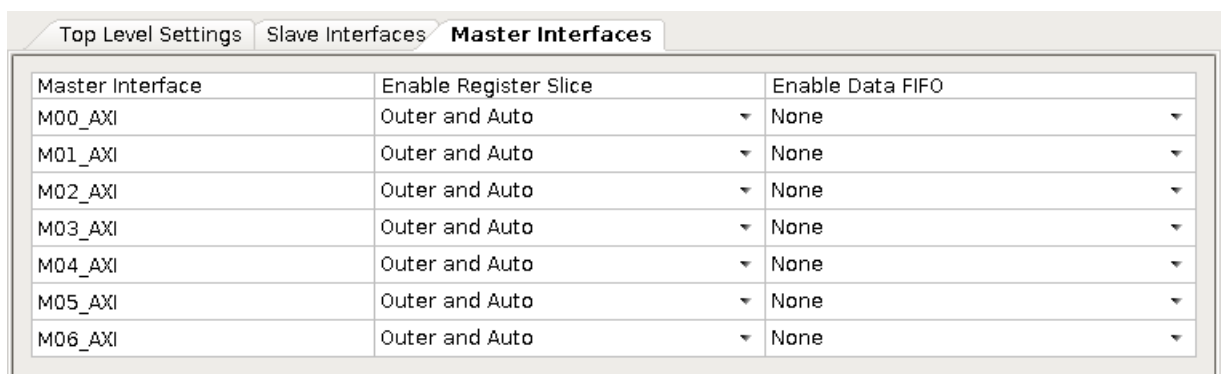


Figure 3-15: Reconfigurable Expanded Region AXI Interconnect IP Customization - Master Interfaces Tab

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:

- SDx Environments application profiling support, via trace offload hardware infrastructure
- Xilinx Developer Board for Acceleration with KU115 flash memory programming via SPI flash IP infrastructure
- Xilinx Developer Board for Acceleration with KU115 FPGA fan speed control via memory-mapped I²C controller

Contained within the apm_sys sub-hierarchy of the reconfigurable expanded region as shown in [Figure 3-16](#), 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 Programmable Region's four master interfaces (for kernel monitoring) and the XDMA IP via AXI Register Slice IP for partial reconfiguration isolation (for host monitoring). Once tabulated, key profiling data is transformed via 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.

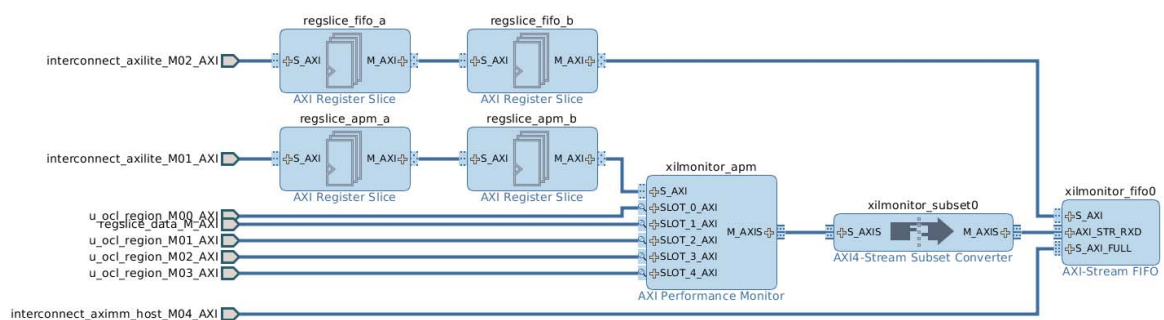


Figure 3-16: Trace Offload Hardware Infrastructure in Reconfigurable Expanded Region apm_sys Sub-Hierarchy

Clocking and Reset

The following sections describe clocking and reset on the Xilinx Developer Board for Acceleration with KU115:

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

Platform Clocking

The primary clock domains of the platform, their sources, frequency, and usage in the platform, are described in [Table 3-1](#).

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 <code>clkwiz_sysclks</code>	50 MHz	The majority of AXI4-Lite control paths throughout the platform
Kernel clock	Clocking Wizard IP instance <code>clkwiz_kernel</code>	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 <code>clkwiz_kernel2</code>	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.
AXI Performance Monitor	Clocking Wizard IP instance <code>clkwiz_sysclks</code>	300 MHz	Exclusive to trace offload hardware infrastructure
DDR4 memory controller clock (one per instance)	DDR4 IP instance <code>ddrmem_[0 1 2 3]</code>	300 MHz	Within the <code>ddrmem_[0 1 2 3]</code> 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

Xilinx Developer Board for Acceleration with KU115. Contained within the base_clocking sub-hierarchy of the static base region as shown in Figure 3-17, that reference clock 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.

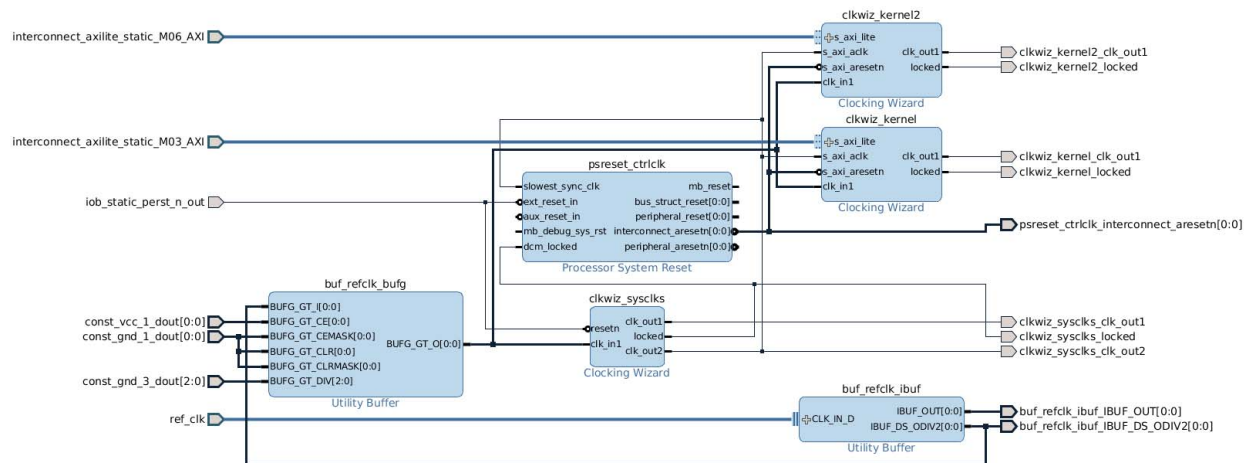


Figure 3-17: Clocking Resources in Static Base Region base_clocking Sub-Hierarchy

Although the AXI4-Lite control and AXI Performance Monitor clocks are both produced from a single Clocking Wizard IP instance utilizing a PLL, the kernel clock and kernel clock 2 are each generated from a separate, memory-mapped Clocking Wizard IP instance utilizing 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 -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](#).

Platform Resets

A Processor System Reset IP core instance exists per clock domain, and drives a reset output to IP cores' reset interfaces 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 that instance's `locked` output.
- The `dcm_locked` (clock is stable) input of a reset controller operating synchronously to the XDMA clock is driven by the XDMA IP instance's `user_lnk_up` output.
- 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 that 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 now discussed.

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:

- 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 that region's reset controllers for synchronization into the appropriate clock domains.

Once 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. Figure 3-18 shows the static base region's `pr_isolation_expanded` level of sub-hierarchy.

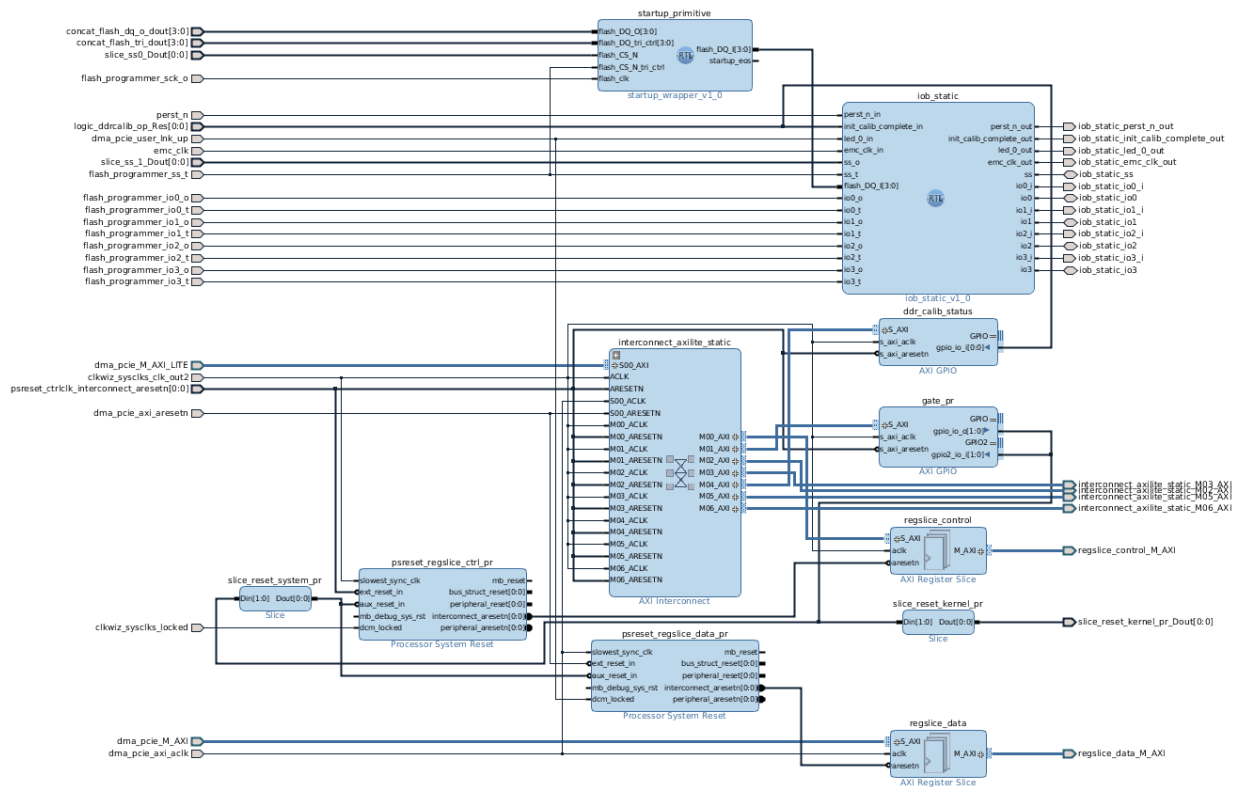


Figure 3-18: Partial Reconfiguration Support Logic in `pr_isolation_expanded` Sub-Hierarchy

Software Platform

The Xilinx Acceleration KU115 4DDR Expanded Partial Reconfiguration platform is a memory-mapped system with PCIe host connectivity supported by a kernel mode DMA driver for the XDMA IP (`xcl_dma`). A Hardware Abstraction Layer (HAL) driver isolates the SDAccel™ Software Platform runtime software from the implementation details of the `xcl_dma` driver. The `xcl_dma` driver interacts with the platform hardware based on a known address mapping.

Address map

The IP Integrator block diagram specifies the following address map for the IP cores in the reference design. The `xcl_dma` 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	M_AXI_LITE <i>AXI4-Lite control interface</i>	u_ocl_region <i>SDAccel OpenCL Programmable Region</i>	S_AXI	0x0000_0000	128K	0x0001_FFFF
		gate_pr <i>GPIO for partial reconfiguration isolation</i>	S_AXI	0x0003_0000	4K	0x0003_0FFF
		gpio_featureid <i>GPIO for feature ID code</i>	S_AXI	0x0003_1000	4K	0x0003_1FFF
		ddr_calib_status <i>GPIO for DDR4 calibration status</i>	S_AXI	0x0003_2000	4K	0x0003_2FFF
		flash_programmer <i>QSPI flash memory controller</i>	AXI_LITE	0x0004_0000	4K	0x0004_0FFF

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 (Cont'd)	M_AXI_LITE <i>AXI4-Lite control interface</i> (Cont'd)	axi_i2c <i>I2C controller for Xilinx Developer Board for Acceleration with KU115 fan</i>	S_AXI	0x0004_1000	4K	0x0004_1FFF
		clkwiz_kernel <i>Clocking Wizard kernel clock source</i>	s_axi_lite	0x0005_0000	4K	0x0005_0FFF
		clkwiz_kernel2 <i>Clocking Wizard kernel clock 2 source</i>	s_axi_lite	0x0005_1000	4K	0x0005_1FFF
		ddrmem_0 <i>DDR4 channel 0 controller</i>	C0_DDR4_S_AXI_CTRL	0x0006_0000	64K	0x0006_FFFF
		ddrmem_2 <i>DDR4 channel 2 controller</i>	C0_DDR4_S_AXI_CTRL	0x0007_0000	64K	0x0007_FFFF
		ddrmem_3 <i>DDR4 channel 3 controller</i>	C0_DDR4_S_AXI_CTRL	0x0008_0000	64K	0x0008_FFFF
		sys_mgmt_wiz <i>System Management Wizard</i>	S_AXI_LITE	0x000A_0000	64K	0x000A_FFFF
		xilmonitor_apm <i>AXI Performance Monitor for application profiling</i>	S_AXI	0x0010_0000	64K	0x0010_FFFF
		xilmonitor_fifo0 <i>Trace offload FIFO for application profiling</i>	S_AXI	0x0011_0000	4K	0x0011_0FFF
	M_AXI <i>AXI Memory Mapped data interface</i>	ddrmem_0 <i>DDR4 channel 0 controller</i>	C0_DDR4_S_AXI	0x0000_0000_0000_0000	4G	0x0000_0000_FFFF_FFFF

Table 4-1: Reference Design Address Map

Master IP Core	AXI Master Interface	Slave IP Core	AXI Slave Interface	Offset Address	Range	High Address
		ddrmem_1 <i>DDR4 channel 1 controller</i>	C0_DDR4_S_AXI	0x0000_0001_0000_0000	4G	0x0000_0001_FFFF_FFFF
XDMA (Cont'd)	M_AXI <i>AXI Memory Mapped data interface</i> (Cont'd)	ddrmem_2 <i>DDR4 channel 2 controller</i>	C0_DDR4_S_AXI	0x0000_0002_0000_0000	4G	0x0000_0002_FFFF_FFFF
		ddrmem_3 <i>DDR4 channel 3 controller</i>	C0_DDR4_S_AXI	0x0000_0003_0000_0000	4G	0x0000_0003_FFFF_FFFF
		xilmonitor_fifo0 <i>Trace offload FIFO for application profiling</i>	S_AXI_FULLL	0x0000_0020_0000_0000	4G	0x0000_0020_FFFF_FFFF
SDAccel OpenCL Programmable Region	M00_AXI <i>AXI Memory Mapped data interface</i>	ddrmem_0 <i>DDR4 channel 0 controller</i>	C0_DDR4_S_AXI	0x0_0000_0000	4G	0x0_FFFF_FFFF
	M01_AXI <i>AXI Memory Mapped data interface</i>	ddrmem_1 <i>DDR4 channel 1 controller</i>	C0_DDR4_S_AXI	0x1_0000_0000	4G	0x1_FFFF_FFFF
	M02_AXI <i>AXI Memory Mapped data interface</i>	ddrmem_2 <i>DDR4 channel 2 controller</i>	C0_DDR4_S_AXI	0x2_0000_0000	4G	0x2_FFFF_FFFF
	M03_AXI <i>AXI Memory Mapped data interface</i>	ddrmem_3 <i>DDR4 channel 3 controller</i>	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, or HAL. The HAL driver provides APIs to runtime software which abstract the `xcldma` driver details. The `xcldma` driver is a kernel mode DMA driver which interfaces to the memory-mapped platform over PCIe. Figure 4-1 shows the layers of the software platform.

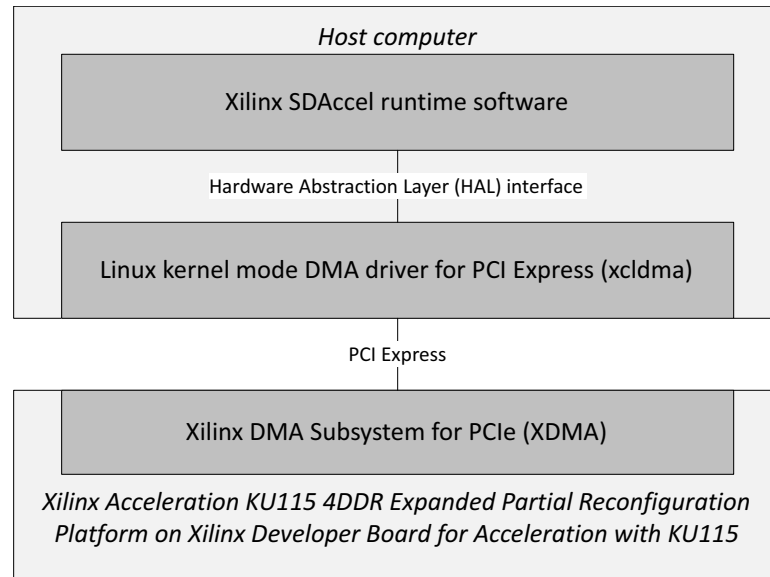


Figure 4-1: **Software Platform Layers**

Linux Kernel Mode DMA Driver for PCI Express (**xcldma**)

The **xcldma** driver is included with the SDx™ Environments installation and is 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 **xcldma** driver. Users who do not use the SDx Environments or the provided HAL driver can still interact with the platform's memory-mapped IP cores by using the provided **xcldma** driver.

As provided, the Xilinx Acceleration KU115 4DDR Expanded Partial Reconfiguration platform uses an address map compatible with other Xilinx-provided platforms, so the **xcldma** driver is not specific to this platform. For more information about the common **xcldma** driver for Hardware Platforms targeting PCIe accelerator cards with XDMA IP, see the *SDAccel Environment Platform Development Guide* (UG1164) [Ref 2]. User modifications to the address map, DMA, or host interface of the provided platform may 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 **xcldma** driver, and exposes C-style APIs to the runtime or other user of the driver.

The HAL driver is not specific to the Xilinx Acceleration KU115 4DDR Expanded Partial Reconfiguration platform. For more information about the common HAL driver for Hardware Platforms which utilize the `xcldma` driver, see the *SDAccel Environment Platform Development Guide* (UG1164) [Ref 2]. User modifications to the `xcldma` driver or replacement of the kernel mode driver may necessitate a new or modified HAL driver.

Implementation

Implementing the Xilinx Acceleration KU115 4DDR Expanded Partial Reconfiguration platform in the Vivado® Design Suite is straightforward. The platform reference design includes modular scripts which are used with Vivado from the SDx™ Environments 2016.3 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.

The individual scripts provided with the reference design are:

- `create_design.tcl`, which 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`, which synthesizes the design's IP cores, followed by the top level. This script is to be run after `create_design.tcl`.
- `run_impl.tcl`, which implements the full design, including optimization, placement, and routing. The script is to be run after `run_synth.tcl`.
- `write_dsa.tcl`, which 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`, which optionally runs all of the above scripts in sequence.

There are two usage flow options for the scripts:

1. For an automated flow that involves no user interaction, simply invoke `vivado -source run.tcl` from the Linux command line. 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 tweak the block diagram before running synthesis, to report timing between stages, etc., begin by invoking `vivado -source create_design.tcl` from the Linux command line. This will create the new project and produce 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: A user who does not wish to create a DSA for the SDAccel Environment use can simply generate a bitstream after `run_impl.tcl` completes, following option 2 above but forgoing the use of `write_dsa.tcl`.

Figure 5-1 illustrates the invocation of the two flows, their stages, as well as interim and final outputs. The more automated option 1 above is shown in blue, and the more manual option 2 is shown in red.

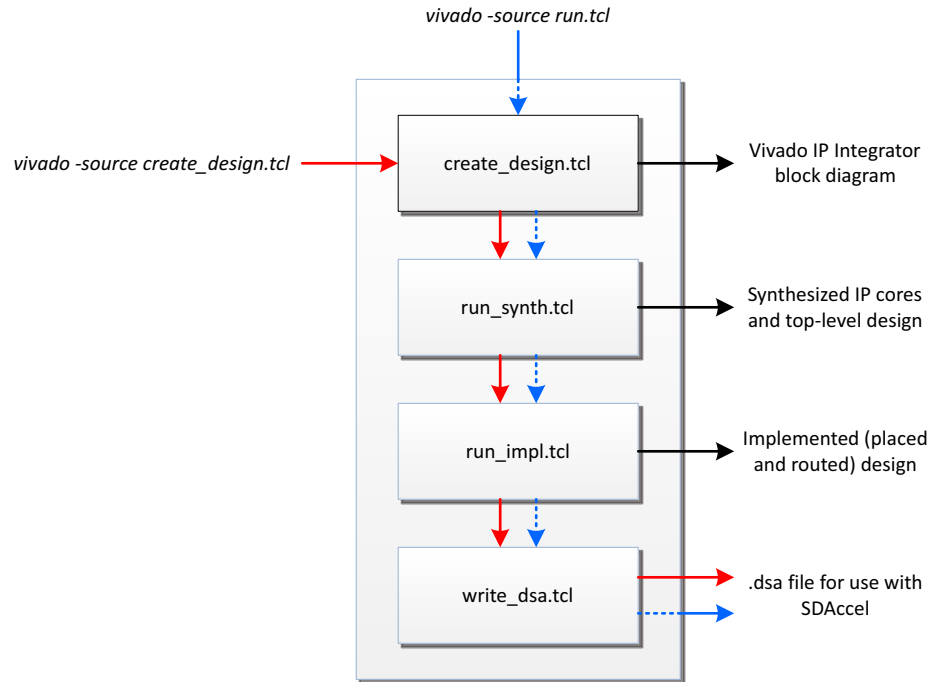


Figure 5-1: Platform Reference Design Implementation Script Usage Flow Options

create_design.tcl Detail

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

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

Command	Purpose
set ::env(OCLE_BLOCK_ADVANCED) 1	Enables advanced customization options of the SDAccel OpenCL Programmable Region IP core.
set_param dsa.expandedPRRegion 1	Required to enable the expanded partial reconfiguration flow in the DSA creation phase.
set_param chipscope.enablePRFlow true	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
<pre>set_property ip_repo_paths "\${commonDir}/iprepo/axi_fifo_mm_s_v4_1__2016_3 \${commonDir}/iprepo/axi_perf_mon_v5_0__2016_3" [current_project]</pre>	Loads locally-modified versions of the axi_fifo_mm_s_v4_1 and axi_perf_mon_v5_0 Xilinx IP cores needed in this release for platform timing closure improvements.

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

Command	Purpose
<pre>set_property dsa.vendor "xilinx" [current_project]</pre>	Sets the vendor field value in the DSA metadata.
<pre>set_property dsa.board_id "xil-accel-rd-ku115" [current_project]</pre>	Sets the board_id field value in the DSA metadata.
<pre>set_property dsa.name "4ddr-xpr" [current_project]</pre>	Sets the name field value in the DSA metadata.
<pre>set_property dsa.version "3.2" [current_project]</pre>	Sets the version field value in the DSA metadata.
<pre>set_property dsa.flash_offset_address "0x4000000" [current_project]</pre>	Sets the flash_offset_address field value in the DSA metadata.
<pre>set_property dsa.description "Xilinx XIL-ACCEL-RD-KU115 Expanded Partial Reconfiguration Quad DDR4 PCIe Gen3 XDMA" [current_project]</pre>	Sets the description field value in the DSA metadata.

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

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

Design Constraints Detail

Some aspects of the design constraints file, `xilinx_xil-accel-rd-kul15_4ddr-xpr_3_2.xdc`, benefit from further explanation. Representative commands are described below.

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, since only the latter can use automatic frequency scaling to compensate for setup violations.

```
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.

```
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 great majority of the available physical resources on the device.

```
create_pblock pblock_expanded_region

add_cells_to_pblock [get_pblocks pblock_expanded_region] [get_cells [list
xcl_design_i/expanded_region]]

resize_pblock [get_pblocks pblock_expanded_region] -add
{SLICE_X0Y300:SLICE_X142Y599 SLICE_X0Y180:SLICE_X133Y299
SLICE_X0Y120:SLICE_X122Y179 SLICE_X97Y60:SLICE_X122Y119 SLICE_X0Y0:SLICE_X95Y119
SLICE_X97Y30:SLICE_X118Y59 SLICE_X97Y0:SLICE_X117Y29}

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 device's lower SLR. Some submodules of the AXI SmartConnect IP instances are constrained to the lower SLR, as described in [Stacked Silicon Interconnect \(SSI\) Technology Support](#).

```
create_pblock pblock_lower

add_cells_to_pblock [get_pblocks pblock_lower] [get_cells [list
xcl_design_i/expanded_region/u_ocl_region]]
```

```

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_X133Y299
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 device's upper SLR. 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](#).

```

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

When implemented with produced DSA as described in [Chapter 5, Implementation](#), the Xilinx Acceleration KU115 4DDR Expanded Partial Reconfiguration platform can be used with the SDAccel™ Environment on the Xilinx Developer Board for Acceleration with KU115. The following sections assume a DSA created from an unmodified reference design, therefore comparable to the file in the SDx™ Environments 2016.3 installation, `xilinx_xil-accel-rd-ku115_4ddr-xpr_3_2.dsa`. User modifications to the reference design can affect compatibility with the host, Xilinx Developer Board for Acceleration with KU115, or the SDAccel System Compiler flow, so confirming successful installation and bring-up is important.

Installation

Once the design has been successfully implemented with closed timing and is free of errors and critical warnings, and the `.dsa` file has been written to disk, the Xilinx Developer Board for Acceleration with KU115 can be programmed.

Install the board in a PCIe Gen3 x8-compatible slot on the host computer. Then 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. Finally, 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 DSA not included with the SDx Environments installation such as one built from the reference design, the `-i` switch must be used to specify the location of that `.dsa` file.

To test for basic compatibility of the host computer with the installed board, programmed device, and installed drivers - and to confirm that DMA access to DDR4 SDRAM is enabled by the platform - use the `sdxsyschk` utility included with the SDx Environments installation. Output should be similar to the example below, which shows the following outputs of note:

- Section 2.1 shows that the PCIe device ID is the expected 0x8238, as described in Host connectivity
- Sections 2.2.1 and 2.3.1 show that the PCIe device is linked at the expected Gen3 x8 status

- Section 2.4.1 shows that device drivers have been successfully installed
- Section 2.5.1 shows that programmed DSA name and version are consistent with the built DSA
- Section 3 shows various successful DMA transfers to and from DDR4 SDRAM

```
> sudo SDx/2016.3/bin/sdxsyschk
Copyright 1986-2016 Xilinx, Inc. All Rights Reserved.
```

```
-----
| Tool Version: SDAccel System Information Checker V2016.3
| Date: 2016-10-10
| Time: 09:55:16.623237
| Host Name: xcojuliank40
-----
```

Table of Contents

```
-----
1. System and Environment Diagnosis
1.1 Linux OS System Check
1.2 64-bit Architecture Check
1.3 Environment Variables Check
1.4 Motherboard System Info

2. PCIe Diagnosis
2.1 Xilinx PCIe Device Check
2.2 Device Link Speed Check
2.3 Root Port Speed Check
2.4 Xilinx Kernel Driver Check
2.5 Xilinx DSA-Device Matching Check

3. Memory Functions (DMA) Testing
3.1 DMA Channel Check
3.2 Zero Data Pattern Transfer Test
3.3 Random Data Transfer Test
3.4 Memory Interface (MIG) Range Test

4. Summary
```

```
-----
Checking Python environment...
Version 2.7.5 (default, Jun 27 2013, 14:17:01)
[GCC 4.8.0]
```

1.1 Linux OS System Check

```
-----
Your Linux System info is:
```

```
LSB Version:
:base-4.0-amd64:base-4.0-noarch:core-4.0-amd64:core-4.0-noarch:graphics-4.0-amd64:graphics-4.0-noarch:printing-4.0-amd64:printing-4.0-noarch
Distributor ID:RedHatEnterpriseWorkstation
Description:Red Hat Enterprise Linux Workstation release 6.6 (Santiago)
Release:6.6
Codename:Santiago
```

```
PASS: OS version check okay.
```

1.2 64-bit Architecture Check

The architecture of your system processor is:
x86_64

PASS: 64-bit system requirement check okay.

1.4 Motherboard System Info

System Information
 Manufacturer: Dell Inc.
 Product Name: Precision Tower 5810
 Version: Not Specified

2.1 Xilinx PCIe Device Check

Processing PCIe info...

PASS: Found Xilinx PCIe device(s):

Device 0 -
 04:00.0 Memory controller: Xilinx Corporation Device 8238

2.2.1 Device Link Speed Check - Device 0

Device 0 -
 The link capability is: PCIe Gen3(8 GT/s) with 8 lanes
 The link status is: PCIe Gen3(8 GT/s) with 8 lanes

PASS: Your card's running transfer rate matches its capabilities.

2.3.1 Root Port Speed Check - Device 0

Root Port speed analysis on Device 0 -
 The link status is: PCIe Gen3(8 GT/s) with 8 lanes
 The link capability is: PCIe Gen3(8 GT/s) with 16 lanes

PASS: Your card's running transfer rate matches its root port's running link status rate.

2.4.1 Xilinx Kernel Driver Check - Device 0

Device 0 -
 NOTE: The device is expecting Xilinx XCLDMA driver
 PASS: The device is running on Xilinx XCLDMA kernel driver
 PASS: The kernel module is loaded successfully.
 Driver Version: 2016.3.2

2.5.1 Xilinx DSA-Device Matching Check - Device 0

Device 0 -
 PASS: The DSA version is being detected as: xilinx:xil-accel-rd-kul15:4ddr-xpr:3.2

PASS: The DSA binary is correctly installed and being detected as:
 10ee-8238-4432-0000000000000000.dsabin

3.1.1 DMA Channel Check - Device 0

```
-----
Device 0 uses xcldma
Number of card to host channels detected: 2
Number of host to card channels detected: 2
```

PASS: DMA channels are detected.

3.2.1 Zero Data Pattern Transfer Test - Device 0

```
-----
Initializing byte count and block count to the following default values for the
transfer...
```

```
Byte count = 1K
Block count = 16
Generating golden file with null data pattern...
```

```
Testing channel 0
Transferring data from host to card...
Transferring data from card back to host...
Verifying the size (bytes) of binary data output... : 16384
PASS: Output of binary data from the DMA transfer matches expected null value.
```

```
Testing channel 1
Transferring data from host to card...
Transferring data from card back to host...
Verifying the size (bytes) of binary data output... : 16384
PASS: Output of binary data from the DMA transfer matches expected null value.
```

3.3.1 Random Data Transfer Test - Device 0

```
-----
Randomizing byte count and block count used for the transfer...
Byte count = 9K
Block count = 29
```

Generating golden file with random data...

```
Testing channel 0
Transferring data from host to card...
Transferring data from card back to host...
Verifying the size (bytes) of binary data output... : 267264
PASS: Output of binary data from the DMA transfer matches expected golden value.
```

```
Testing channel 1
Transferring data from host to card...
Transferring data from card back to host...
Verifying the size (bytes) of binary data output... : 267264
PASS: Output of binary data from the DMA transfer matches expected golden value.
```

3.4.1 Memory Interface (MIG) Range Test - Device 0

```
-----
4 DDR MIG detected from the DSA, each is dedicated for 4 Gbytes of memory space.
```

```
Initializing byte count and block count used for the transfer...
Byte count = 1K
Block count = 64
```

```

Generating golden file with random data...

Testing MIG 1 with 4G of offset...
Transferring data from host to card...
Transferring data from card back to host...
Verifying the size (bytes) of binary data output... : 65536
PASS: Output of binary data from the DMA transfer matches expected golden value.

Testing MIG 2 with 8G of offset...
Transferring data from host to card...
Transferring data from card back to host...
Verifying the size (bytes) of binary data output... : 65536
PASS: Output of binary data from the DMA transfer matches expected golden value.

Testing MIG 3 with 12G of offset...
Transferring data from host to card...
Transferring data from card back to host...
Verifying the size (bytes) of binary data output... : 65536
PASS: Output of binary data from the DMA transfer matches expected golden value.

```

4 Summary

Section	Result
1.1 Linux OS System Check	PASS
1.2 64-bit Architecture Check	PASS
1.3 Environment Variables Check	NOT RUN
1.4 Motherboard System Info	DONE
2.1 Xilinx PCIe Device Check	PASS
2.2.1 Device Link Speed Check - Device 0	PASS
2.3.1 Root Port Speed Check - Device 0	PASS
2.4.1 Xilinx Kernel Driver Check - Device 0	PASS
2.5.1 Xilinx DSA-Device Matching Check - Device 0	PASS
3.1.1 DMA Channel Check - Device 0	PASS
3.2.1 Zero Data Pattern Transfer Test - Device 0	PASS
3.3.1 Random Data Transfer Test - Device 0	PASS
3.4.1 Memory Interface (MIG) Range Test - Device 0	PASS

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` query utility included with the SDx Environments installation. Note that the PCIe IDs - Vendor, Device, SDevice (Subsystem) - must match those indicated in [Host Connectivity](#) for compatibility with the `xcldma` driver.

```

> SDx/2016.3/runtime/bin/xbsak query -d 0

INFO: Found 1 device(s)
DSA name:      xilinx:xil-accel-rd-kul15:4ddr-xpr:3.2
HAL ver:      5.0
Vendor:       10ee
Device:       8238
Device ver:   50
SDevice:      4432
SVendor:      10ee
DDR size:     0x1000000 KB
DDR count:    4
Data alignment: 64

```

```

DDR free size: 0x1000000 KB
Min xfer size: 64
OnChip Temp: 55
Fan Temp: 0
VCC INT: 936 mV
VCC AUX: 1788 mV
VCC BRAM: 939 mV
OCL freq: 300 MHz
PCIE: GEN3 x 8
DMA threads: 4
Fan Speed: 0
MIG Calib: 1
CU Status:
  0: 0x4 (IDLE)
  1: 0x4 (IDLE)
  2: 0x4 (IDLE)
  3: 0x4 (IDLE)
  4: 0x4 (IDLE)
  5: 0x4 (IDLE)
  6: 0x4 (IDLE)
  7: 0x4 (IDLE)
  8: 0x4 (IDLE)
  9: 0x4 (IDLE)
 10: 0x4 (IDLE)
 11: 0x4 (IDLE)
 12: 0x4 (IDLE)
 13: 0x4 (IDLE)
 14: 0x4 (IDLE)
 15: 0x4 (IDLE)

```

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 `sdxsyschk` and `xbsak` utilities, see the *SDx Environments Release Notes, Installation, and Licensing Guide* (UG1238) [Ref 1].

Bring-Up Tests

With the Xilinx Developer Board for Acceleration with KU115 installed, DSA programmed, and SDx utilities indicating compatibility and readiness of the device, it can be useful to compile and execute bring-up tests which utilize 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 Xilinx Developer Board for Acceleration with KU115 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.

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

Use with the SDAccel Environment

When the Xilinx Developer Board for Acceleration with KU115 with programmed DSA is proven operational through successful installation and SDx 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 can also be helpful when targeting the `xilinx_xil-accel-rd-ku115_4ddr-xpr_3_2.dsa` DSA from the SDx Environments installation, or as built from the Xilinx Acceleration KU115 4DDR Expanded Partial Reconfiguration platform reference design.

Device Identification

The VBNV identifier for the DSA is `xil-accel-rd-ku115:4ddr-xpr:3.2`, so an XOCC script (which invokes the SDAccel System Compiler) will include the following argument:

```
--xdevice xilinx:xil-accel-rd-ku115:4ddr-xpr:3.2
```

If attempting to target the DSA built from the platform reference design rather than one from the SDx Environments installation, it is necessary to specify the 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](#), 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. 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 platform's kernel clock 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:

```
--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 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 will result 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 may 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 <http://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 <http://www.xilinx.com/legal.htm#tos>.

Automotive Applications Disclaimer

XILINX PRODUCTS ARE NOT DESIGNED OR INTENDED TO BE FAIL-SAFE, OR FOR USE IN ANY APPLICATION REQUIRING FAIL-SAFE PERFORMANCE, SUCH AS APPLICATIONS RELATED TO: (I) THE DEPLOYMENT OF AIRBAGS, (II) CONTROL OF A VEHICLE, UNLESS THERE IS A FAIL-SAFE OR REDUNDANCY FEATURE (WHICH DOES NOT INCLUDE USE OF SOFTWARE IN THE XILINX DEVICE TO IMPLEMENT THE REDUNDANCY) AND A WARNING SIGNAL UPON FAILURE TO THE OPERATOR, OR (III) USES THAT COULD LEAD TO DEATH OR PERSONAL INJURY. CUSTOMER ASSUMES THE SOLE RISK AND LIABILITY OF ANY USE OF XILINX PRODUCTS IN © Copyright 2016 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Vivado, Zynq, 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.