

Xilinx Software Development Kit (SDK) User Guide

System Performance Analysis

UG1145 (v2019.1) May 22, 2019

Revision History

05/22/2019: Released with Vivado® Design Suite 2019.1 without changes from 2018.3.

Section	Revision
12/05/2018 Version 2018.3	
General updates	Minor editorial changes.
06/14/2016 Version 2016.2	
General updates	Minor editorial changes.

Table of Contents

Revision History	2
Chapter 1: Introduction	
SPA Toolbox	5
Performance Analysis Flow	7
Requirements	9
Additional Resources	9
Chapter 2: Background	
Chapter 3: System Performance Modeling Project	
SPM Software	15
SPM Hardware	18
Chapter 4: Monitor Framework	
PL Profile Counters	22
PS Profile Counters	23
Host-Target Communication	24
Chapter 5: Getting Started with SPM	
ATG Configuration	26
Performance Analysis Perspective	27
Chapter 6: Evaluating Software Performance	
Performance Monitoring	30
Visualizing Performance Improvements	33
Chapter 7: Evaluating High-Performance Ports	
HD Video Traffic	37
High-Bandwidth Traffic	39
Chapter 8: Evaluating DDR Controller Settings	
Default DDRC Settings	46

Modified DDRC Settings	47
Utilizing On-Chip Memory.....	49
Chapter 9: Evaluating Memory Hierarchy and the ACP	
Assess Memory Performance	51
Data Size and Locality	54
Shared L2 Cache.....	55
Chapter 10: Working with a Custom Target	
Instrumenting Software	60
Instrumenting Hardware.....	61
Monitoring a Custom Target.....	62
Chapter 11: End-To-End Performance Analysis	
Assess Requirements.....	65
Model Design	66
Performance Validation.....	69
In-Depth Analysis.....	72
Appendix A: Performance Checklist	
Appendix B: Additional Resources and Legal Notices	
Xilinx Resources	77
Solution Centers.....	77
Documentation Navigator and Design Hubs	77
References	78
Training Resources.....	79
Please Read: Important Legal Notices	79

Introduction

The Xilinx® Zynq®-7000 SoC device family integrates a dual-core Arm® Cortex™-A9 MPCore™ Processing System (PS) with Xilinx 7 series Programmable Logic (PL) in 28nm technology. The PS and PL are connected through standard Arm AMBA™ AXI interfaces designed for performance and system integration. This style of SoC is new to the industry, and therefore requires novel performance analysis and benchmarking techniques to provide you with a clear understanding of system performance. It is critical to understand the Zynq-7000 SoC architecture so that you can utilize its full potential and differentiate your products in the marketplace.

SPA Toolbox

To address the need for performance analysis and benchmarking techniques, the Xilinx Software Development Kit (SDK) has been enhanced with a System Performance Analysis (SPA) toolbox to provide early exploration of hardware and software systems. Specifically, a Zynq-7000 SoC designer is presented with insights into both the PS and the PL to understand the interactions across such a complex, heterogeneous system. You can observe system performance at critical stages of a design flow, enabling you to refine the performance of your system.

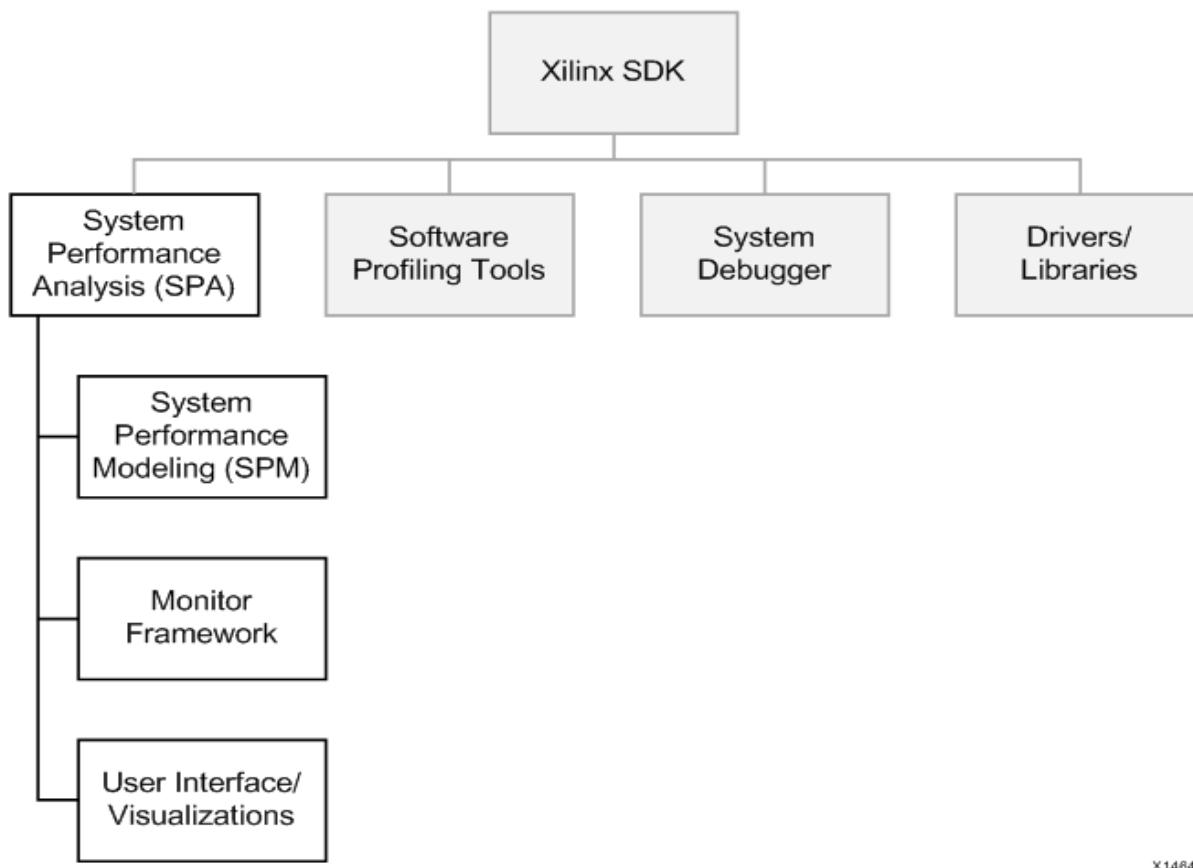


Figure 1-1: Xilinx SDK Features Including the System Performance Analysis Toolbox

Figure 1-1 shows how the SPA toolbox fits into the feature set of SDK. Other important features of SDK include software profiling tools, a system debugger, and supporting drivers and libraries. The SPA toolbox contains a monitor framework, user interface, and visualizations that are common for two important use models: an early exploration environment called System Performance Modeling (SPM) and the monitoring and analysis of your own designs. This common toolbox can be used for performance validation to ensure consistent and expected performance throughout the design process.

SPM is a unique feature of SDK that enables complex performance modeling even before design work has started. SPM is executed in actual target hardware and includes a highly configurable, fixed bitstream containing five AXI traffic generators. These traffic generators are configurable cores used by SDK to model PL traffic activity. Software applications can also be run simultaneously in the processing system, and you can specify system configuration parameters.

Performance Analysis Flow

Using the SPM design, SDK enables you to achieve an end-to-end performance analysis flow. This flow allows you to model the traffic of your estimated design and then validate the performance using your actual design.



Figure 1-2: End-to-End Performance Analysis Flow

As shown in [Figure 1-2](#), this is a four-step process that includes:

Assess Requirements – You first estimate the AXI traffic requirements of your design, including data throughputs at the PS-PL interfaces of the target system.

Model Design – Based on these traffic requirements, you then use SPM to model your design. Since actual target hardware is used, real-time clock rates are achieved. This provides improved run-times over other modeling environments, as well as increased accuracy because real system activity is being monitored.

Performance Validation – As you develop your design, you can validate the performance results by monitoring and visualizing your actual design.

In-Depth Analysis - The SDK performance tools also enable a deep-dive analysis into your design to evaluate design options and gauge the impact of design improvements.

There are multiple benefits to achieving such an end-to-end performance analysis flow, including:

- *Reduce risk for success* – Achieving the desired latencies and throughputs with the SPM-based model can provide a much stronger assurance that the final design will achieve the same performance. While certainly not a guarantee, you can have confidence that Zynq-7000 SoC can meet your performance goals.
- *Design Improvements* – Running traffic scenarios using SPM will provide valuable insights into system performance (for example, latencies) that will help you in your actual design work.
- *What-If Scenarios* – Since the SPM is a highly-configurable model, you can use it to try variations of capabilities, features, and architectures.

This guide describes the technical details of this performance analysis toolbox, as well as a methodology to appreciate its usefulness and depth. Note that this toolbox informs you as much about the capabilities of the target platform as it does about the particulars of a design.

Consequently, this guide is intended to highlight various features of the Zynq-7000 SoC, as well as how SDK can assist you in maximizing its capabilities. After reading this guide, you should be able to:

- Use the SPM design to analyze a software application and model hardware traffic
- Better understand the Zynq-7000 SoC platform and its proficiency
- Recognize the uses and capabilities of the PS-PL interfaces on the Zynq-7000 SoC
- Leverage the memory hierarchy (including the L1 and L2 data caches and DDR) to achieve the best system performance
- Model a design using SPM and follow up with performance validation of the actual design

Specific examples are used to provide detailed results and analysis. This guide also describes how you can obtain similar results in SDK. The goals of this guide are such that you can extrapolate these techniques to model and analyze your own designs.

The next four chapters provide an overview of the SPA toolbox:

- [Chapter 2, Background](#) outlines system performance and defines why it is important.
- [Chapter 3, System Performance Modeling Project](#) describes the contents of the SPM project.
- [Chapter 4, Monitor Framework](#) defines the monitoring infrastructure used by the SDK tool.
- [Chapter 5, Getting Started with SPM](#) provides the necessary steps to get up and running with the SPM design.

The next set of chapters provides in-depth exploration into using the SPM design:

- [Chapter 6, Evaluating Software Performance](#) begins by running a software executable that comes with the SPM project.
- [Chapter 7, Evaluating High-Performance Ports](#) then introduces traffic on the High-Performance (HP) ports while running the same software
- [Chapter 8, Evaluating DDR Controller Settings](#) describes how to change DDR controller (DDRC) settings and analyze their impact on the HP port traffic.
- [Chapter 9, Evaluating Memory Hierarchy and the ACP](#) evaluates bandwidth and latency from the memory hierarchy, and then introduces traffic on the Accelerator Coherency Port (ACP) to investigate its impact on that performance.

Additionally, two chapters are devoted to running performance analysis on your designs:

- [Chapter 10, Working with a Custom Target](#) defines some steps and requirements to instrumenting and monitoring your design.

- [Chapter 11, End-To-End Performance Analysis](#) describes the full cycle of performance analysis as described in [Figure 1-2](#).

Finally, the key performance recommendations mentioned throughout this guide are summarized in [Appendix A, Performance Checklist](#).

Requirements

If you would like to reproduce any of the results shown and discussed in this guide, the requirements include the following:

1. Software
 - a. Xilinx Software Development Kit (SDK) 2015.1
 - b. Optional: USB-UART drivers from [Silicon Labs](#)
2. Hardware
 - a. Xilinx [ZC702 evaluation board](#) with the XC7Z020 CLG484-1 part
 - b. AC power adapter (12 VDC)
 - c. Xilinx programming cable; either platform cable or Digilent USB cable
 - d. *Optional:* USB Type-A to USB Mini-B cable (for UART communications)
 - e. *Optional:* Zynq-7000 SoC ZC702 Base Targeted Reference Design (UG925) [\[Ref 3\]](#).

Additional Resources

For a description of SDK and links to other guides, refer to the Xilinx SDK home page:

<https://www.xilinx.com/tools/sdk.htm>

Zynq-7000 SoC documentation, including the *Zynq-7000 SoC Technical Reference Manual* (UG585) [\[Ref 1\]](#), can be found here:

<https://www.xilinx.com/products/silicon-devices/soc/zynq-7000/index.htm>

Background

Targeting a design to the Zynq®-7000 SoC can be facilitated if you are aware of the numerous system resources and their capabilities. After you become acquainted with the device, then you can decide how to map your functionality onto those resources and optimize the performance of your design.

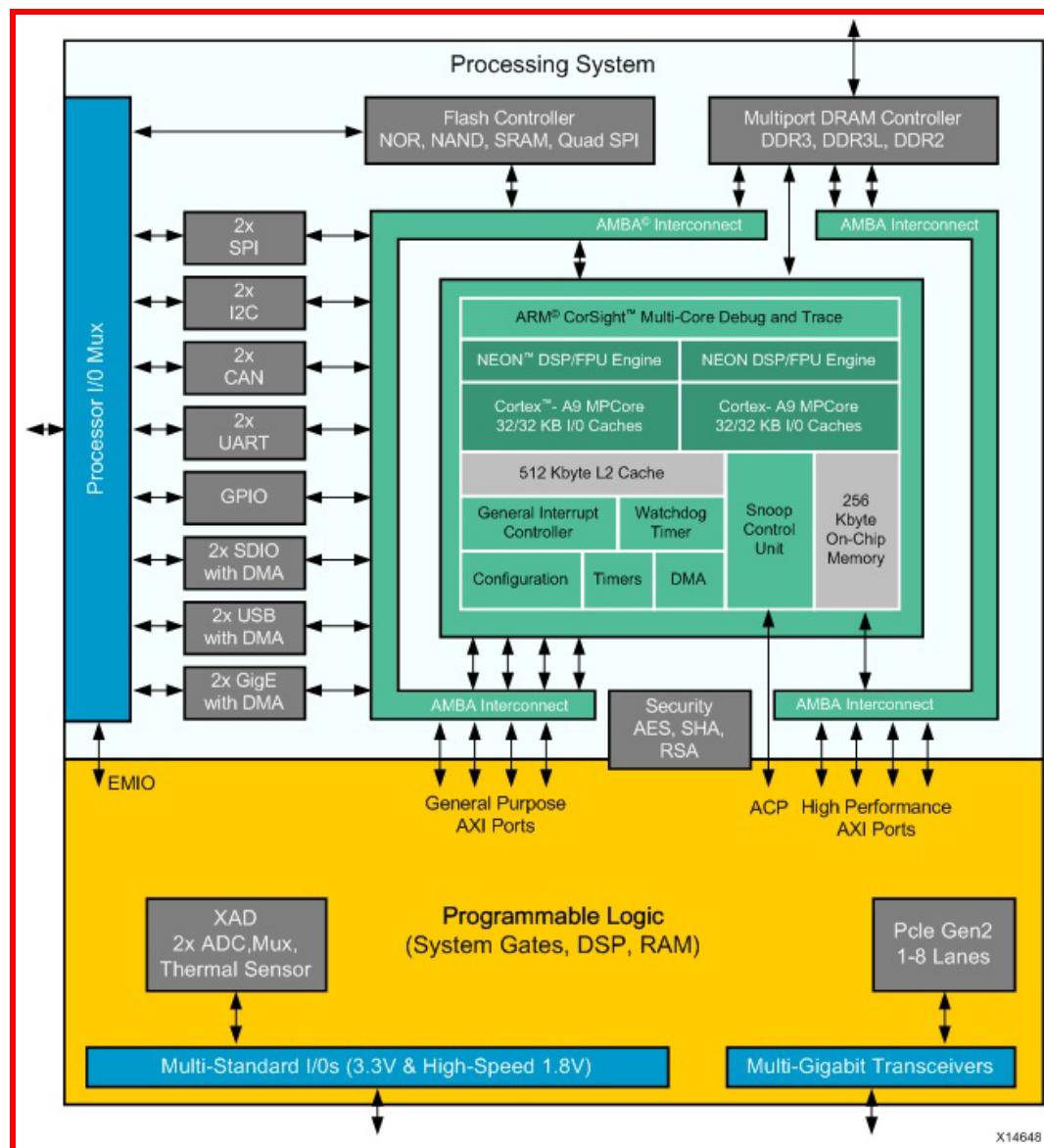


Figure 2-1: Block Diagram of Zynq-7000 SoC

Figure 2-1 shows the block diagram of the Zynq-7000 SoC. The Processing System (PS) includes two Arm Cortex-A9 cores, a 512 KB L2 cache, a 256 KB on-chip memory, and a number of peripherals and controllers.

The PL includes configurable system gates, DSPs, and block memories. There are three types of interfaces between the PS and PL:

- General Purpose (GP) AXI Ports: control ports connected directly to the slave interconnect
- Accelerator Coherency Port (ACP): low-latency access to PL masters, with optional coherency with L1 and L2 cache
- High-Performance (HP) AXI Ports: PL bus masters with high bandwidth datapaths to the DDR and OCM memories

These interfaces serve a variety of purposes for communication between the PS, the Programmable Logic (PL), and any external memory (for example, DDR SDRAM or simply DDR).

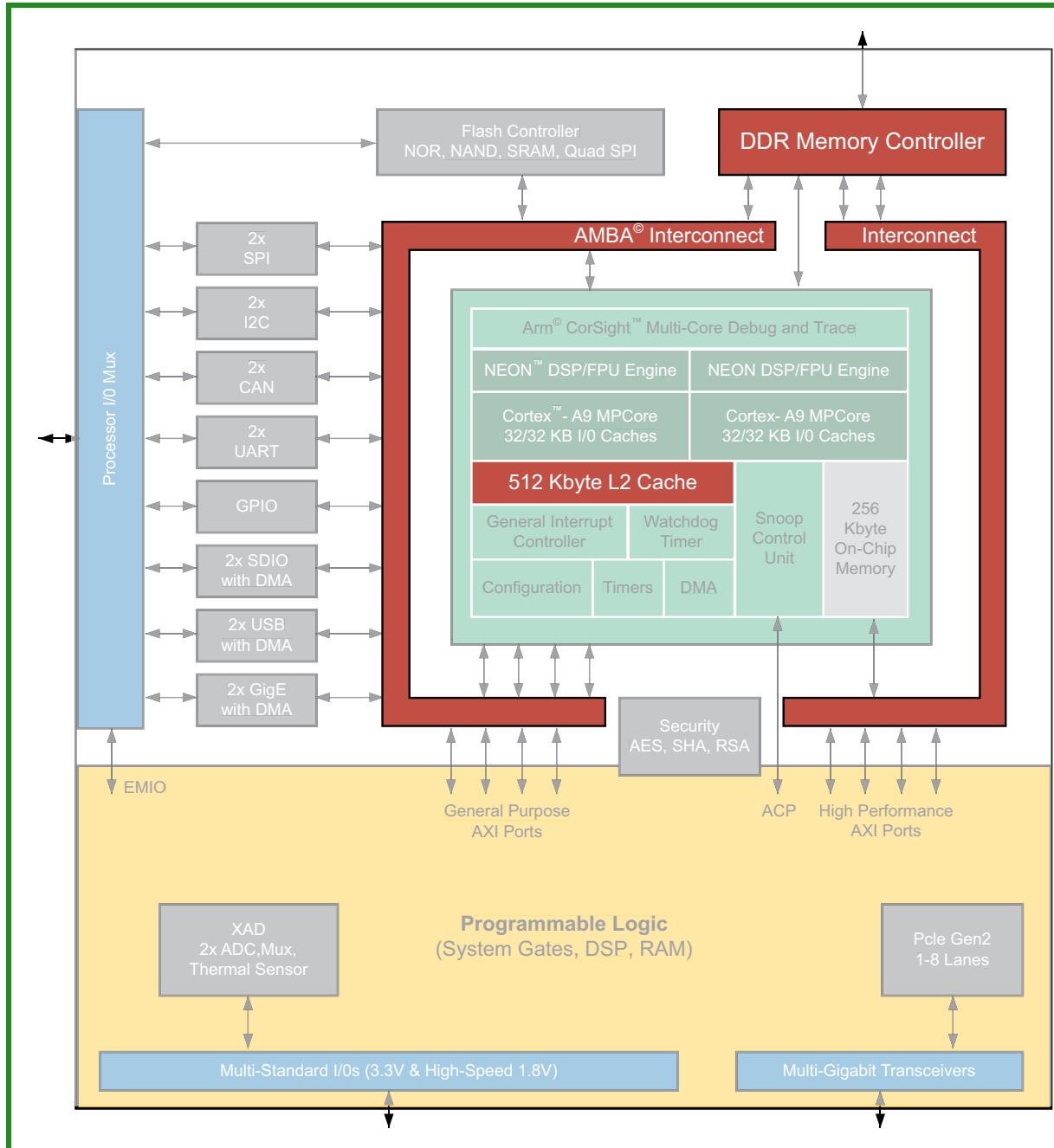


Figure 2-2: Zynq-7000 SoC Block Diagram with Shared Resources Highlighted

Figure 2-2 highlights the shared resources in the Zynq-7000 SoC block diagram. A design that fully utilizes all of the capabilities of this SoC requires multiple channels of high-bandwidth communication. All of this traffic in the system inevitably involves shared resources. In a poorly designed system this could lead to multiple points of contention across the L2 cache, the DDR memory controller, and the high-speed interconnects. To optimize the performance of a design that fully utilizes all of these resources, a designer requires observability into multiple areas of such a system. Improvements can then be made through code changes, design connectivity, or system settings.

An excellent example is the L2 cache. Because the ACP shares the L2 cache with the Cortex-A9 CPUs, high-throughput traffic from either the ACP or the processors has the potential to impact the performance of the other. While this effect can be anticipated, understanding the nature and extent of this impact can be difficult.

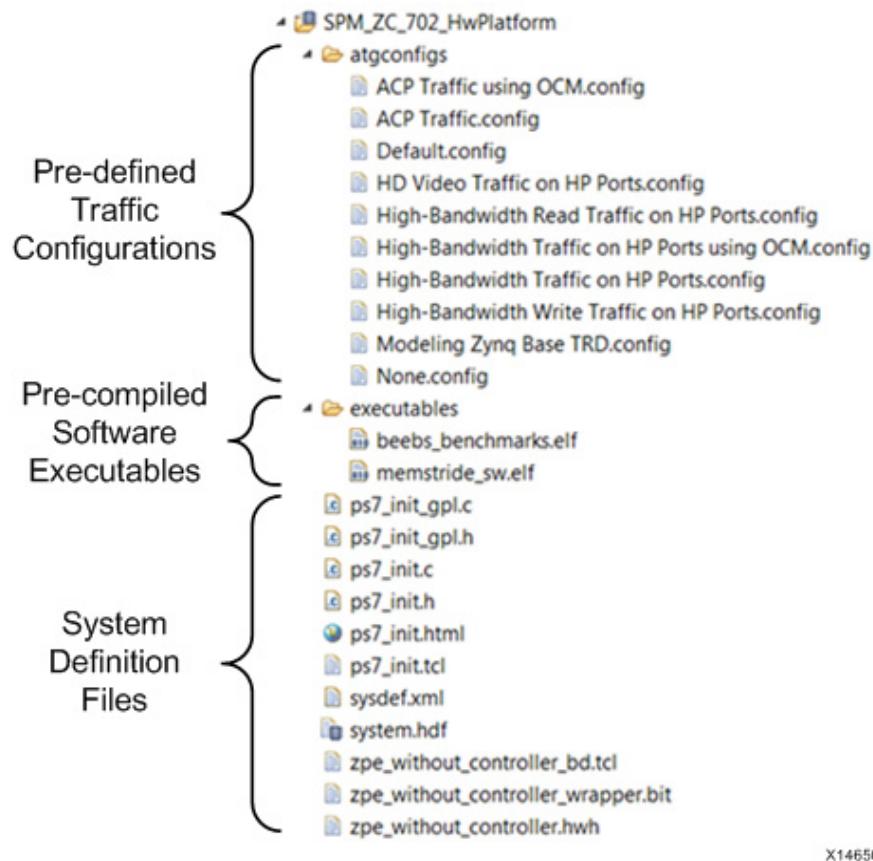
This is where the SDK performance capabilities can be exploited. SDK provides the visualizations to better understand this impact as well as SPM: an exploratory, early design environment. Tables and real-time graphs are provided to visualize PS and PL performance metrics on a common timeline. This provides much needed insight into when and where things occur to quickly isolate any potential issues. Furthermore, the same performance metrics are calculated for the SPM design as well as your own design, therefore providing early performance exploration as well as consistent results across a design flow.

Before you begin using the performance tools and analyzing performance results, it is important to understand what design is being used and what is being measured. The next chapters summarize the SPM design and the performance monitor framework used by SDK.

System Performance Modeling Project

Xilinx® Software Development Kit (SDK) is delivered with a predefined project that enables System Performance Modeling (SPM) and helps enable performance analysis at an early design stage. The SPM project contains both software executable and a post-bitstream, configurable hardware system. The SPM serves multiple purposes, including the following:

- *Target Evaluation* – Gain a better understanding of the target platform with minimal hardware design knowledge or experience. A complex SoC such as the Zynq®-7000 SoC can be characterized and evaluated, enabling critical partitioning trade-offs between the Arm Cortex-A9s and the programmable logic. Most importantly, this evaluation can be done independently of the progression or completion of a design. You can always decide to gain a better understanding of your target platform.
- *Perform Stress Tests* – Identify and exercise the various points of contention in the system. An excellent way of understanding a target platform is to evaluate the limits of its capabilities.
- *Performance Evaluation* – Model the traffic of a design. Once the target platform is understood, specifics of a design can be entered as traffic scenarios, and SPM can be used to evaluate architectural options.
- *Performance Validation* – Instrument the final design and confirm results with the initial model. The same monitoring used in the SPM design can also be added to your design (see [Instrumenting Hardware, page 61](#)), providing a validation of the modeled performance results.



X14650

Figure 3-1: SPM Project Files in SDK

Figure 3-1 lists the files contained in the predefined SPM project. It contains predefined traffic configurations, two pre-compiled software executables, and many system definition files. The system definition files include Processing System 7 (PS7) initialization files, a Vivado® Design Suite Tcl file that creates the design definition in Vivado IP integrator, and a bitstream containing a predefined Programmable Logic (PL) design.

SPM Software

The SPM project contains two pre-compiled software executables:

- A collection of benchmarks called the Bristol/Embecosm Embedded Energy Benchmark Suite (BEEBS)
- Memory stride benchmarks designed specifically to test memory bandwidth and latency

BEEBS Benchmarks

The BEEBS program comprises a sequence of eight diverse benchmarks. As shown in [Table 3-1](#), this suite contains algorithms originally contained as part of various well-known benchmark suites such as MiBench, WCET, and DSPstone. These benchmarks were chosen to test embedded systems and be portable to standalone or bare metal targets. For more information about the BEEBS benchmark suite, refer to *BEEBS: Open Benchmarks for Energy Measurements on Embedded Platforms* [\[Ref 9\]](#).

Table 3-1: BEEBS Benchmarks Provided in Pre-Compiled Program

Benchmark	Suite	Description
Blowfish encoder	MiBench	Block cipher
Cyclic Redundancy Check (CRC)	MiBench	Error detecting code
Secure Hash Algorithm (SHA)	MiBench	NIST cryptographic hash function
Dijkstra's algorithm	MiBench	Graph search algorithm
Discrete Cosine Transform (DCT)	WCET	Transform used in MP3, JPEG
2-D FIR filter	DSPstone	Common in image filtering
Floating-point matrix multiplication	WCET	Multiplication of two square matrices
Integer matrix multiplication	WCET	Multiplication of two square matrices

The BEEBS benchmark suite was modified in four important ways:

- *Single executable* – A single executable was created in order to run all benchmarks within a single program. [Table 3-1](#) lists the benchmarks in the order that they are called in the pre-compiled program contained in the SPM project.
- *Sleep between benchmarks* – A sleep time of 1 second was inserted between each benchmark. In order to differentiate the execution of the benchmarks, this is an interrupt-based sleep where the CPU utilization is 0%.
- *Different data array sizes* – The benchmarks were modified to allow for a variety of data array sizes. The three array sizes used in the pre-compiled benchmark program are:
 - 4 KB (2-D array: 32 x 32) – fits into the 32 KB L1 data cache
 - 64 KB (2-D array: 128 x 128) – fits into the 512 KB L2 data cache
 - 1024 KB (2-D array: 512 x 512) – fits into the DDR SDRAM
- *Instrumented* – Run-times of these benchmarks were calculated based on instrumentation added to the code before and after every benchmark. See [Instrumenting Hardware, page 61](#) for more details. Note that the run-times are

reported in the transcript delivered over the UART (Terminal 1 panel in SDK) and read manually.

Memory Stride Benchmarks

The memory stride benchmarks differ from the BEEBS benchmarks in that they do minimal computation and are specifically designed to test memory bandwidth and latency. The various tests included in the software are listed in [Table 3-2](#). The five different types of tests include:

- Linear Bandwidth – Memory bandwidth tests with constant linear stride accesses. For the pre-compiled application that comes with the SPM project, the stride length is equal to a 32 byte cache line.
- Random Bandwidth (Pre-Computed) – Memory bandwidth tests of random stride accesses. The random addresses used in these tests are pre-computed.
- Random Bandwidth (Real-Time) – Memory bandwidth tests of random stride accesses. The random addresses used in these tests are computed in real time using a random number generator.
- Linear Latency – Memory latency tests with constant linear stride accesses.
- Random Latency (Pre-Computed) – Memory latency tests of random stride accesses. The random addresses used in these tests are pre-computed.

Similar to BEEBS, the memory stride benchmarks are contained in a single executable with a 1 second sleep in between benchmarks. [Table 3-2](#) shows the order of benchmarks in the program. Each benchmark operates on the same three array sizes listed above for BEEBS. The memory stride program is also instrumented; however, the transcript reports the achieved throughputs and latencies of each test.

Table 3-2: Memory Stride Benchmarks Provided in Pre-Compiled Program

Test Type	Pattern Type	Operation Type
Bandwidth	Linear	Read
		Write
		Copy
		Read/Write
	Random (Pre-Computed)	Read
		Write
		Copy
	Random (Real-Time)	Read
		Write
		Copy

Table 3-2: Memory Stride Benchmarks Provided in Pre-Compiled Program

Test Type	Pattern Type	Operation Type
Latency	Linear	Read
	Random (Pre-Computed)	Read

SPM Hardware

The SPM project contains a predefined hardware design that can be used for early performance exploration. This design is delivered in the project as a fixed bitstream to configure the Zynq-7000 SoC Programmable Logic (PL).

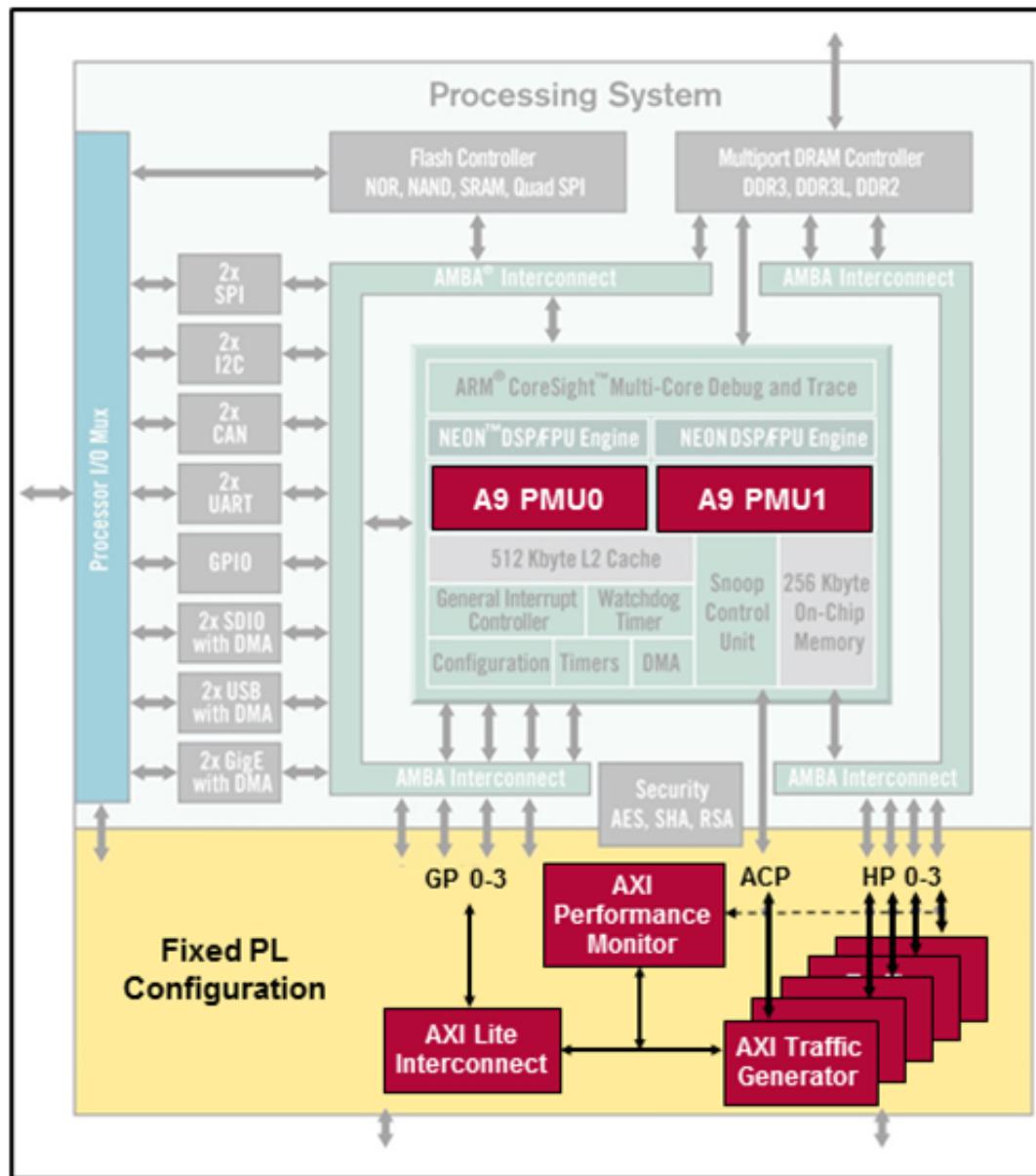


Figure 3-2: Pre-Defined SPM Design in Zynq-7000 SoC

Figure 3-2 shows a block diagram of this pre-defined SPM design targeting the Zynq-7000 SoC. It is a highly-configurable design involving five AXI traffic generator (ATGs) and one AXI performance monitor (APM). One ATG is connected to each of the four high-performance (HP) ports as well as one to the ACP. The configuration of these cores is performed via the general purpose (GP) port 0 master. CPU performance metrics are obtained using the Performance Monitor Units (PMUs).

AXI Traffic Generator

The AXI Traffic Generator (ATG) is an intelligent traffic generator configured to exhibit specific AXI traffic behavior. The command queue for each ATG is filled during initialization, and these commands are executed upon the application of a start bit. Separate queues with depth of 256 commands are included for write and read traffic. The ATG also has a loop mode where the traffic in the command queue is continuous, iterating over the traffic in the queue until a stop bit has been applied. In SDK, this has been simplified to a *Traffic Duration* (sec) value. The traffic specification for the ATGs is described in [Chapter 5, Getting Started with SPM](#).

AXI Performance Monitor

The AXI Performance Monitor (APM) is a core designed to measure the real-time performance of every connected AXI interface. In the SPM design, these interfaces include the outputs of the five ATGs. The APM is configured in *Profile* mode, providing an event count module that includes six profile counters per ATG. These six counters are designed to monitor average throughput and latency for all write and read channels on the connected AXI interfaces. See [Chapter 4, Monitor Framework](#) to understand how these metrics are calculated.

Performance Monitor Units

Each Arm Cortex-A9 CPU contains a Performance Monitor Unit (PMU). These PMUs are configured to monitor a number of different performance metrics, including CPU utilization and Instructions Per Cycle (IPC). The PMUs are accessed as part of the performance monitor framework used by SDK. See [Chapter 4, Monitor Framework](#) to understand how these PMU counters are used.

Monitor Framework

The performance analysis toolbox in the Xilinx® Software Development Kit (SDK) offers a set of system-level performance measurements. For a design that targets the Zynq®-7000 SoC, this includes performance metrics from both the Programmable Logic (PL) and the Processing System (PS).

The PL performance metrics include the following:

- *(Write/Read) Transactions* – number of AXI transactions
- *(Write/Read) Throughput* – write or read bandwidth in MB/sec
- *Average (Write/Read) Latency* – average write or read latency of AXI transactions

The PS performance metrics include the following:

- *CPU Utilization (%)* – percentage of non-idling CPU clock cycles
- *CPU Instructions Per Cycle (IPC)* – estimated number of executed instructions per cycle
- *L1 Data Cache Access and Miss Rate (%)* – number of L1 data cache accesses and the miss rate
- *CPU (Write/Read) Stall Cycles Per Instruction* – estimated number of stall cycles per instruction due to memory writes (write) and data cache refills (read)

This mix of performance metrics is gathered from different parts of the target and combined into a common timeline and display, enabling system-level performance analysis. In order to gather and display these metrics, SDK includes a monitor framework that accesses various profile counters across a target system, as well as a host-target communication framework to perform the sampling and offloading of these counters.

The metadata required by SDK to perform this monitoring is exported by Vivado® Design Suite and read by SDK when a Hardware Platform Specification project is created. While this export/import procedure has already been performed for the System Performance Modeling (SPM) project, it can also be executed on your design (see [Instrumenting Hardware, page 61](#)). In other words, all of the monitoring and analysis described herein is available and applicable to the SPM design as well as your design. This extension is addressed in [Chapter 11, End-To-End Performance Analysis](#).

PL Profile Counters

An AXI Performance Monitor (APM) inserted into a design provides the PL profile counters (see [Instrumenting Hardware, page 61](#) for information about how to do this insertion for your design). There are six counters per connected AXI interface. Table 4-1 lists how these counters are used. Note that these are incrementing up-counters that capture a running total of bytes transferred, total latency, and the number of completed AXI transactions, respectively. As shown in Table 4-1, the difference of these counters between successive samples (signified by Δ) is used in the calculations of average throughput and latency across a given Sample Interval. SDK uses a default sample interval of approximately 50 msec.

Table 4-1: Profile Counters per AXI Interface in AXI Performance Monitor

APM Counter Metric	Performance Metrics Equation
Write Byte Count	Write Throughput = $(\Delta(\text{Write Byte Count}))/(\text{Sample Interval})$
Write Latency Count	Average Write Latency = $(\Delta(\text{Write Latency Count}))/(\Delta(\text{Write Transaction Count}))$
Write Transaction Count	
Read Byte Count	Read Throughput = $(\Delta(\text{Read Byte Count}))/(\text{Sample Interval})$
Read Latency Count	Average Read Latency = $(\Delta(\text{Read Latency Count}))/(\Delta(\text{Read Transaction Count}))$
Read Transaction Count	

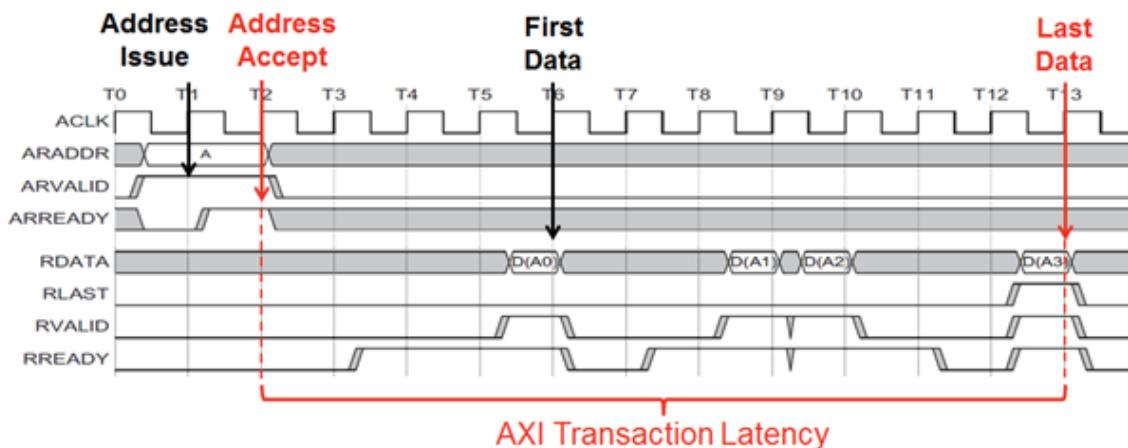


Figure 4-1: Example AXI Transaction Timeline Showing Read Latency Used in SDK

Because there are many events that constitute an AXI transaction, it is important to understand the events that SDK captures and how latency is calculated. Figure 4-1 shows a timing diagram of an example AXI read transaction, including address, data, and control signals.

The start event used by SDK is Address Accept (ARVALID=1 and ARREADY=1), while the end event is Last Data (RLAST=1 and RVALID=1 and RREADY=1). The difference between these two events within a common transaction is deemed the Transaction Latency.

Note that the APM does not support:

- more than 32 out-standing transactions. Where the transactions can be initialization of more than 32 addresses before the first data block initiation (or) transfer/acceptance of 32 data blocks before first address.
- interleaved transactions. Where the transactions can be address initialization of new data transaction before finishing the current data transaction (before getting wlast/rlast).

PS Profile Counters

The PS profile counters comprise the Arm Performance Monitor Unit (PMU) included with each Cortex-A9 CPU. [Table 4-2](#) shows how the six counters in each PMU are automatically configured by SDK, as well as the performance metric equations they provide for each CPU. As shown later in this guide, these are the metrics that are directly graphed in SDK.

Table 4-2: Profile Counters used in Arm Performance Monitor Unit

Event Name	Event	Description	Performance Metrics Equation
CCNT	N/A	Non-idle clock cycle counter	$\text{CPU Utilization}(\%) = 100 \times \Delta \text{CCNT} / (2 \times \Delta(\text{SPM reference clock}))$
INST_RENAME	0x68	Number of instructions that went through register renaming	$\text{CPU Instructions Per Cycle (IPC)} = (\Delta(\text{INST_RENAME})) / \Delta \text{CCNT}$
L1D_CACHE_REFILL	0x03	L1 data cache misses	$\text{L1 Data Cache Miss Rate (\%)} = 100 \times (\Delta(\text{L1D_CACHE_REFILL})) / (\Delta(\text{L1D_CACHE}))$
L1D_CACHE	0x04	L1 data cache misses	$\text{L1 Data Cache Miss Rate (\%)} = 100 \times (\Delta(\text{L1D_CACHE_REFILL})) / (\Delta(\text{L1D_CACHE}))$
DCACHE_DEP_STALL	0x61	Data cache dependent stall cycles with pending linefill	$\text{Read Stall Cycles Per Instruction} = (\Delta(\text{DCACHE_DEP_STALL})) / (\Delta(\text{INST_RENAME}))$
MEM_WRITE_STALL	0x81	Processor stall cycles waiting for memory write	$\text{Write Stall Cycles Per Instruction} = (\Delta(\text{MEM_WRITE_STALL})) / (\Delta(\text{INST_RENAME}))$

Host-Target Communication

Now that you appreciate the profile counters and how they are processed and displayed in the System Performance Analysis (SPA) toolbox, it is also important to understand how those counters are sampled and transferred from the target to the host machine.

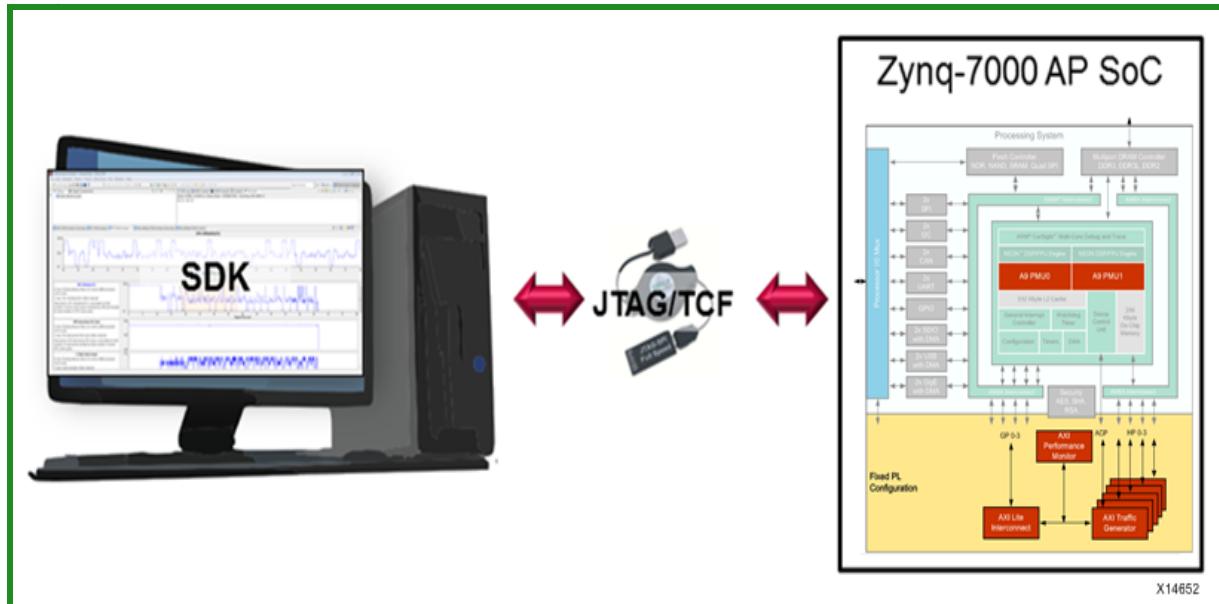


Figure 4-2: Host-Target Communication Infrastructure for SDK

Figure 4-2 shows the host-target communication infrastructure used by SDK. The SDK tool itself runs on a host machine and includes a hardware server using the Target Communication Framework (TCF). TCF can efficiently communicate with the target over the JTAG programming cable. This TCF-based hardware server controls and samples the profile counters listed in [Table 3-1, page 16](#) and [Table 3-2, page 17](#) in the least intrusive manner possible. The APM and PMU counters are read using memory-mapped access via the Zynq-7000 SoC central interconnect. The counter values are then processed as described above and displayed in SDK.

Getting Started with SPM

The pre-defined SPM project can be automatically loaded into the workspace by selecting **File > New > SPM Project** in the Xilinx® Software Development Kit (SDK).

When prompted, select the SPM target. The SPM project name is defined based on the specified target. For example, selecting a target supported by the ZC702 board will automatically assign SPM_ZC_702_HwPlatform as the name of the SPM project.

You can now edit the configuration for your test, including the target setup, software application, and traffic to use for the AXI Traffic Generators (ATGs). The target setup includes board connection information, bitstream selection, and Processing System 7 (PS7) initialization. The bitstream is specified by default to be the SPM bitstream, while the PS7 settings are defined in the SPM system definition files. If you do not have a local or direct connection to a target board from your host machine, refer to the *Vivado® Design Suite User Guide: Embedded Processor Hardware Design* (UG898) [Ref 2] for help in setting up your board connection.

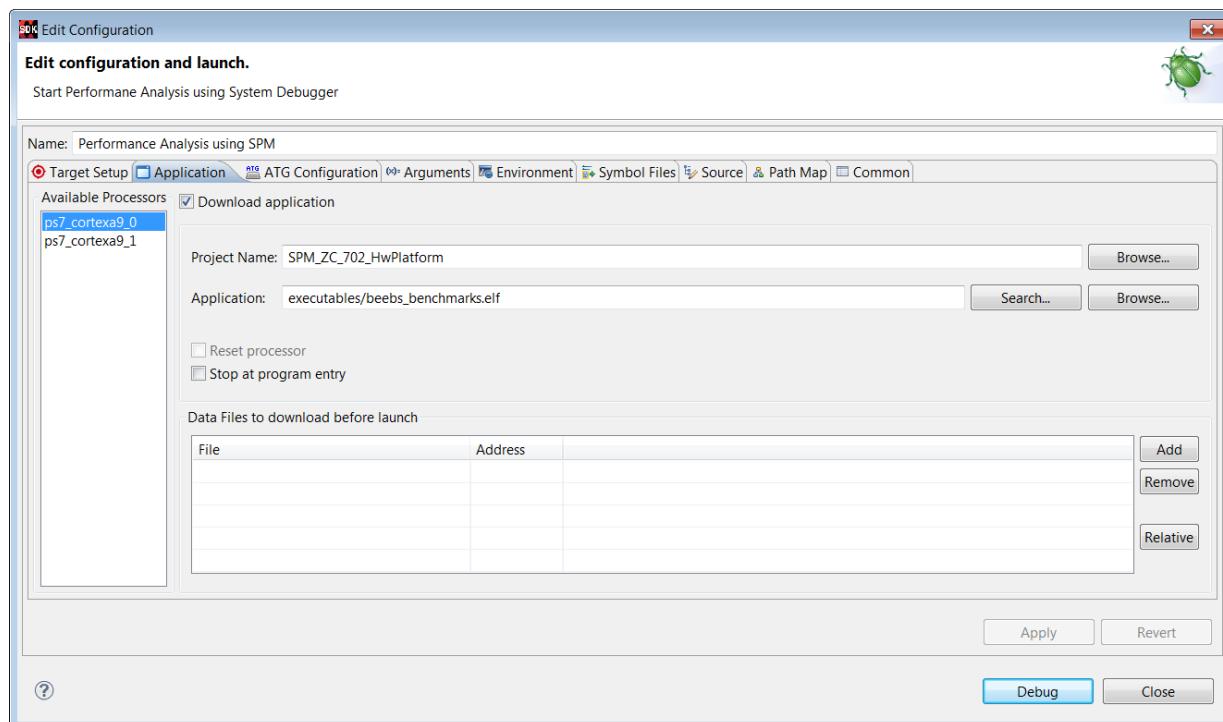


Figure 5-1: Application Setup in SDK Configuration Wizard

Figure 5-1 shows the software application setup for the SPM. By default, the software run in this configuration is the executable `beebes_benchmarks.elf`, an excellent starting application because it contains significant amounts of data processing and memory accesses (see [Chapter 3, System Performance Modeling Project](#)).

This default application was chosen to prepare your design for evaluating software performance, which is described in the next chapter. Later in this guide, this setting will be modified and memory stride – the other application that is supplied with the SPM project – will be used.

ATG Configuration

SDK enables you to specify traffic scenarios for configuring the AXI traffic generator (ATGs).

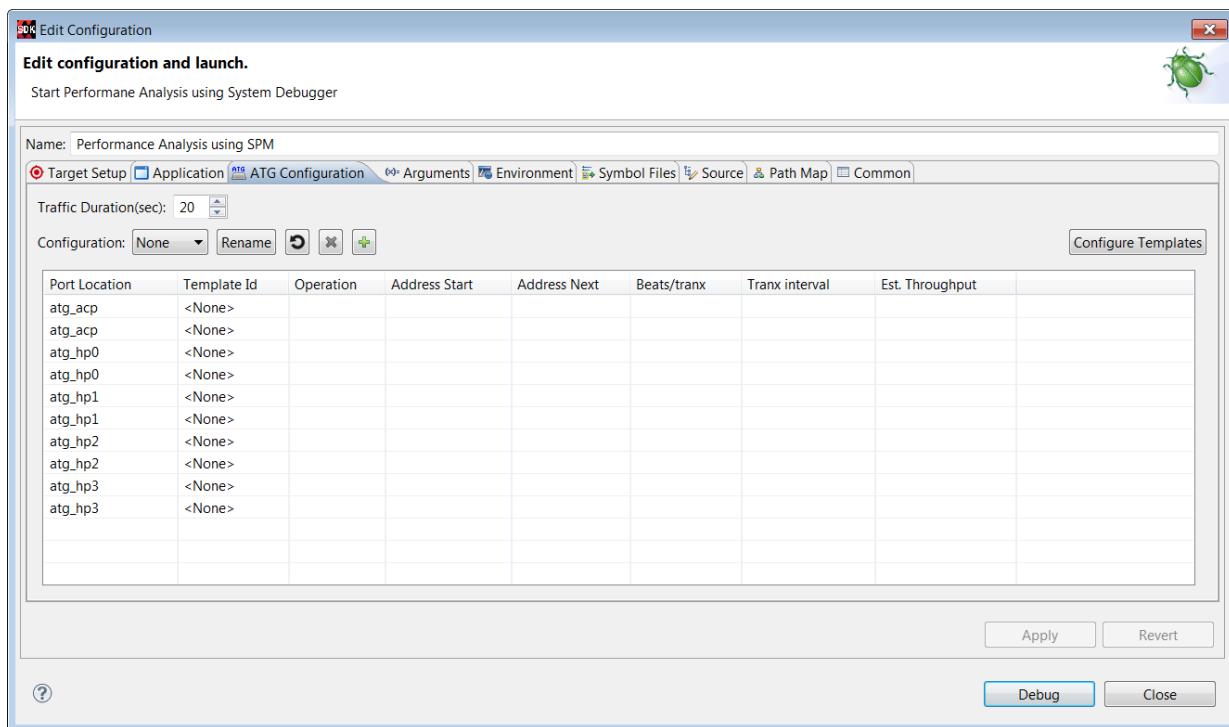


Figure 5-2: ATG Configuration Setup in SDK Configuration Wizard

Figure 5-2 shows the ATG Configuration tab, where this traffic can be specified. In the fixed bitstream of SPM (see [SPM Software, page 15](#) for more details), one ATG is attached to every High Performance (HP) port and the Accelerator Coherency Port (ACP) – all interfaces between the programmable logic (PL) and the processing system (PS). Memory access patterns are modeled by the ATGs issuing AXI transactions with various lengths, frequencies, and memory addresses onto each port. Different combinations of read/write traffic patterns across all four HP and the ACP ports can be specified. This ATG Configuration table includes the following:

- **Port Location** – the ATG that is being configured (in SPM, the names are self-explanatory)
- **Template Id** – disable/enable the traffic defined on that row. When set to “<None>” no traffic is generated.
- **Operation** – the type of AXI transactions to issue on the AXI port. Valid values include “RD” for memory read transactions (via the AXI read port) and “WR” for memory write transactions (via the AXI write port).
- **Address_Start** – the starting address of the AXI transactions. Valid values include the following preset addresses: ddr (an alias of ddr0), ddr0, ddr1, ddr2, ddr3, and ocm.
- **Address_Next** – method by which every subsequent address is generated after Address_Start (that is, incremental or random within an address range).
- **Beats/tranx** – the burst length or number of beats per AXI transaction. Valid values are between 1 and 256.
- **Tranx interval** – the number of PL clock cycles between the start of two consecutive AXI transactions. Valid values are between 5 and 1024.
- **Est. Throughput** – an estimated throughput (in MB/sec) of the current traffic settings; the throughput is calculated using: $8 \times (L_{burst}/(\text{MAX}(L_{interval}, L_{burst}))) \times f$, where L_{burst} is the Beats/tranx, $L_{interval}$ is the Tranx interval, and f is the PL clock rate.

Performance Analysis Perspective

After editing your configuration, you can start a performance analysis session by clicking **Debug** in the Edit Configuration dialog box (Figure 5-2, page 26). SDK opens the Performance Analysis perspective.

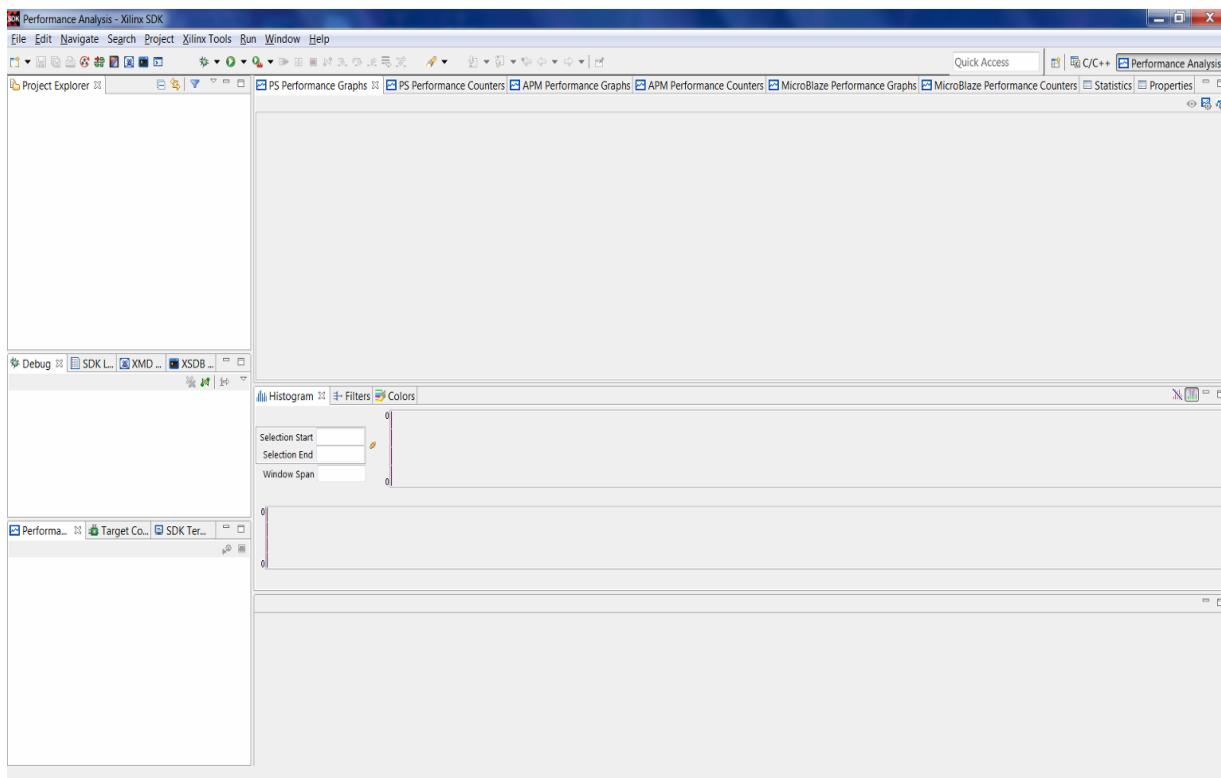


Figure 5-3: Performance Analysis Perspective in SDK

Figure 5-3, shows the Eclipse perspective, which provides a dashboard of performance results. There are five important panels to introduce:

1. PS Performance Graphs – PS (Arm®) metrics will be displayed using these graphs.
2. PS Performance Counters – PS (Arm®) metrics listed in a tabular format.
3. APM Performance Graphs – plots showing the APM performance metrics listed in [Table 4-1, page 22](#) using these graphs.
4. APM Performance counters– plots showing the APM performance metrics listed in [Table 4-1, page 22](#) in a tabular format.
5. MicroBlaze Performance Graphs – various plots showing the MicroBlaze performance metrics on a shared timeline. This is not utilized by SPM and therefore not covered in this guide.
6. MicroBlaze Performance Counters – a table summarizing MicroBlaze™ performance metrics. This is not utilized by SPM and therefore not covered in this guide.

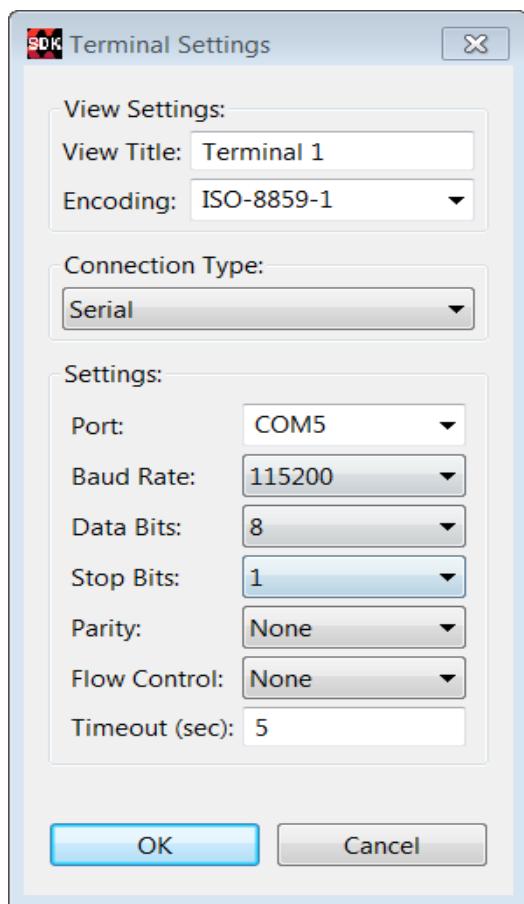


Figure 5-4: Terminal Settings to View Transcript in SDK

This guide uses various results that are output by the target applications and viewed in the Terminal 1 panel in SDK. These results include software run-times, achieved bandwidths, and average latencies.

If you would like to re-create these results and/or run your own, you must connect a USB cable (Type-A to USB Mini-B) between the UART on the ZC702 board and a USB port on your host machine.

Figure 5-4 shows the terminal settings required to view this UART transcript in SDK. To view this pop-up menu, click on the Connect button  as shown in the upper-right of Figure 5-3, page 28. Note that your port may be different than COM5, but all other settings are valid.

Evaluating Software Performance

There are a number of monitoring capabilities that evaluate software performance on a Zynq®-7000 SoC. These capabilities can inform you about the efficiency of your application and provide visualizations for you to better understand and optimize your software.

Performance Monitoring

To illustrate these performance monitoring capabilities, the Bristol/Embecosm Embedded Energy Benchmark Suite (BEEBS) benchmark program (see [Chapter 3, System Performance Modeling Project](#)) was run on a ZC702 target board and performance results were captured in the Xilinx® Software Development Kit (SDK) 2015.1. There was no traffic driven from the Programmable Logic (PL), so this is a software only test. Note that if you would like to re-create these results, the software application selection is shown in [Figure 5-1, page 25](#), and the PL traffic is shown in [Figure 5-2, page 26](#). Although the results shown here are for the BEEBS benchmarks, these exact same metrics can also be obtained for a program provided to or compiled by SDK.

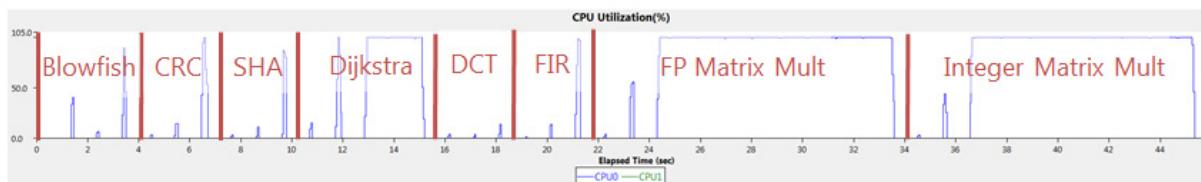


Figure 6-1: CPU Utilization Labeled with BEEBS Benchmarks

There are 24 total tests performed – each of the eight benchmarks listed in [Table 3-1, page 16](#) is run on the three different data array sizes (that is, 4 KB, 64 KB, and 1024 KB). Because a sleep time of 1 second was inserted between each test, the CPU utilization gives a clear view of when these benchmarks were run. [Figure 6-1](#) helps orient the timeline for the results. The three data sizes were run from smallest to largest within each benchmark, which can be seen in the value and length of the utilization of CPU0. It took approximately 45 seconds to run BEEBs benchmark without any traffic.

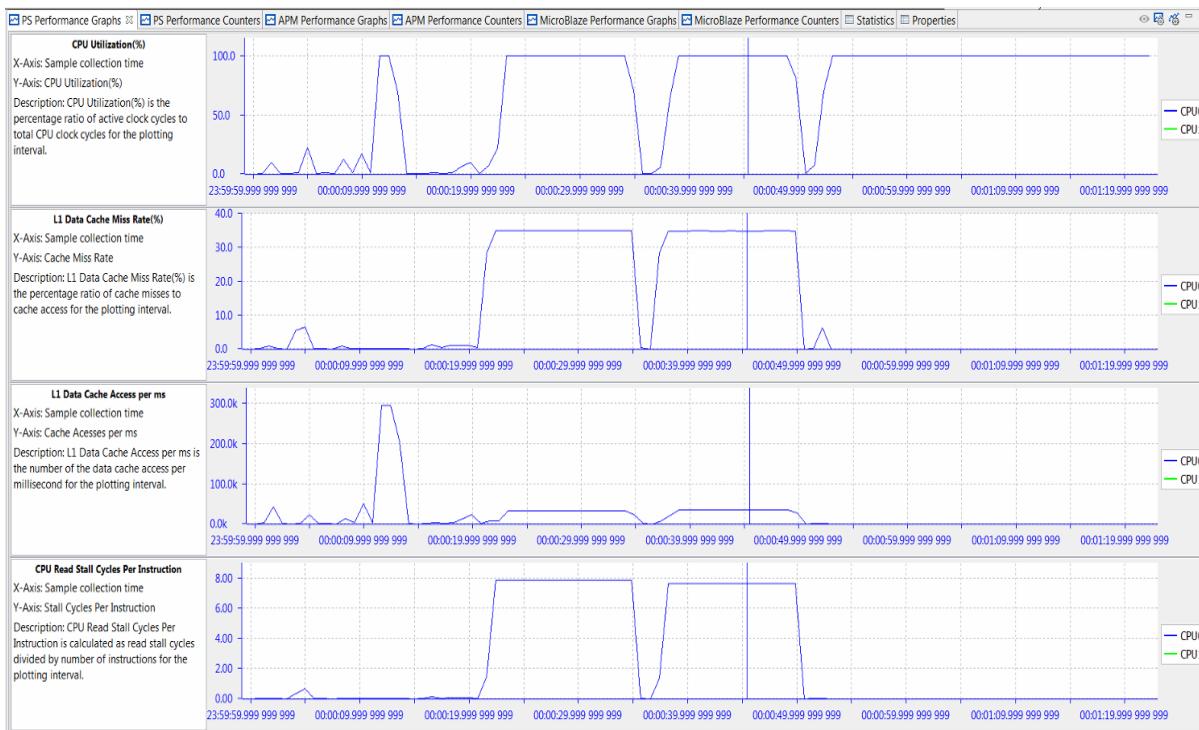


Figure 6-2: Performance Analysis of BEEBS Benchmarks - CPU Utilization, IPC, L1 Data Cache

Figure 6-2 shows four different graphs displayed by SDK in the PS Performance panel: CPU Utilization (%), CPU Instructions Per Cycle, L1 Data Cache Access, and L1 Data Cache Miss Rate (%).

While CPU Utilization helps determine when benchmarks were run, the most insightful analysis is achieved when multiple graphs are considered. For these particular tests, the significant analysis comes from the Instructions Per Cycle (IPC) and the L1 data cache graphs. IPC is a well-known metric to illustrate how software interacts with its environment, particularly the memory hierarchy. See *Computer Organization & Design: The Hardware/Software Interface* by David A. Patterson and John L. Hennessy ([Ref 10]). This fact becomes evident in Figure 6-2 as the value of IPC follows very closely the L1 data cache accesses. Also, during the memory-intensive matrix multipliers, the lowest values of IPC correspond with the highest values of L1 data cache miss rates. If IPC is a measure of software efficiency, then it is clear from Figure 6-2 that the lowest efficiency is during long periods of memory-intensive computation. While sometimes this is unavoidable, it is clear that IPC provides a good metric for this interaction with the memory hierarchy.

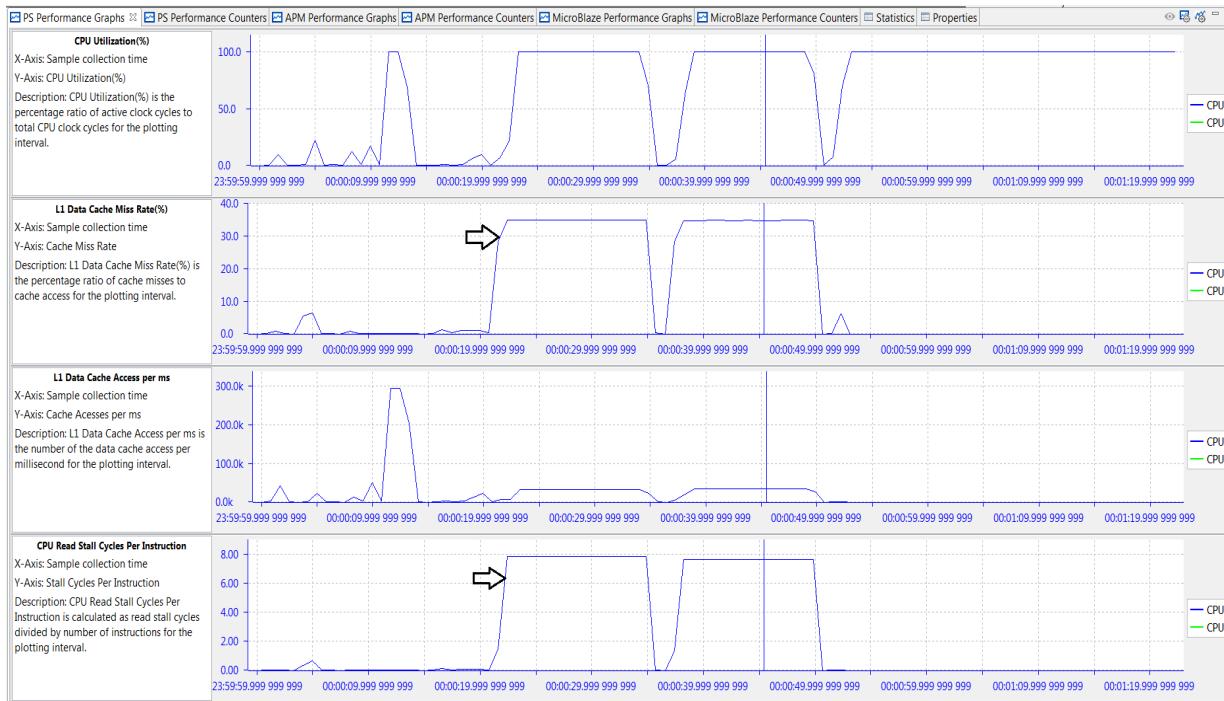


Figure 6-3: Performance Analysis of BEEBS Benchmarks - L1 Data Cache, Stall Cycles Per Instruction

Taking this analysis one step further, the graphs in [Figure 6-3](#) provide information on data locality; that is, where the processor retrieves and writes data as it performs the specified algorithms. A low L1 data cache miss rate means that a significant amount of data is stored and accessed from that respective cache. A high L1 miss rate coupled with high values in CPU write or read stall cycles per instruction means that much of the data is coming from DDR.

As expected, an understanding of the software application aids in the analysis. For example, the two long periods of high L1 miss rate and CPU read stall cycles shown in [Figure 6-3](#) are when the floating-point and integer matrix multiplier algorithms are operating on the 512 x 512 (256K words = 1024 KB) two-dimensional or 2-D data array.

To refine the performance of an application, one option to consider using is to enable/disable the L2 data cache prefetch. This is specified with bit 28 of *reg15_prefetch_ctrl* (absolute address 0xF8F02F60). If this data prefetching is enabled, then the adjacent cache line will also be automatically fetched. While the results shown in [Figure 6-2, page 31](#) and [Figure 6-3](#) were generated with prefetch enabled, the BEEBS benchmarks were also run with prefetch disabled. The integer matrix multiplication saw the largest impact on software run-time with a decrease of 9.0%. While this prefetch option can improve the performance of some applications, it can lower the performance of others. It is recommended to verify run-times of your application with and without this prefetch option.

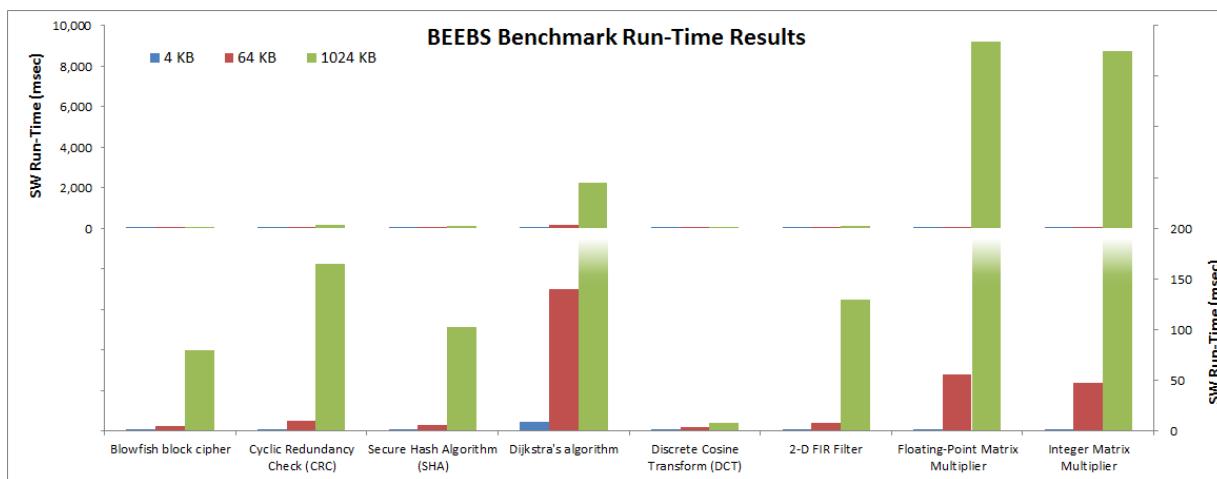


Figure 6-4: Run-Time Results of Software in BEEBS Benchmark Suite

As mentioned above, a helpful measurement of overall software performance is the run-time of different portions of the application. Figure 6-4 shows a summary of run-times for the eight BEEBS benchmarks as run on the three different data array sizes (see [Instrumenting Hardware, page 61](#) on how these run-times were calculated and captured). The Y-axis on the left side of Figure 6-4 shows the full range of values, while the Y-axis on the right side zooms in on the lowest range for clarity. As expected, the run-times increase with larger array sizes. However, the amount of increase varies across the different benchmarks tested. This is because there are a number of different factors that can impact run-time, including the amount of data, data locality, and algorithmic dependencies on data size. As far as data locality, the 4 KB, 64 KB, and 1024 KB data arrays fit into the L1 data cache, L2 data cache, and DDR, respectively.

For more information on assessing the performance of the memory hierarchy, see [Chapter 9, Evaluating Memory Hierarchy and the ACP](#)

Visualizing Performance Improvements

As shown in Figure 6-4 the longest run-times are clearly the matrix multiplications, especially for the largest array size. What if these run-times need to be improved? How would the performance analysis features in SDK help measure and visualize any code optimizations or improvements?

The sample code below shows two different C/C++ implementations for floating-point matrix multiplication. The original and traditional implementation is the `Multiply_Old()` function, whereas a newer implementation is contained in `Multiply_New()`. While the newer implementation appears to be more complicated, it takes advantage of the 32 byte cache lines. See [What Every Programmer Should Know About Memory by Ulrich Drepper](#) ([\[Ref 11\]](#)).

```

#define CLS 32
#define SM  (CLS / sizeof (float))

/* Original method of calculating floating-point matrix multiplication */
void Multiply_Old(float **A, float **B, float **Res, long dim)
{
    for (int i = 0; i < dim; i++) {
        for (int j = 0; j < dim; j++) {
            for (int Index = 0; Index < dim; Index++) {
                Res[i][j] += A[i][Index] * B[Index][j];
            }
        }
    }

/* Modified method of calculating floating-point matrix multiplication */
void Multiply_New(float **A, float **B, float **Res, long dim)
{
    for (int i = 0; i < dim; i += SM) {
        for (int j = 0; j < dim; j += SM) {
            for (int k = 0; k < dim; k += SM) {
                float *rres = &Res[i][j];
                float *rmul1 = &A[i][k];
                for (int i2 = 0; i2 < SM; ++i2, rres += dim, rmul1 += dim) {
                    float *rmul2 = &B[k][j];
                    for (int k2 = 0; k2 < SM; ++k2, rmul2 += dim) {
                        for (int j2 = 0; j2 < SM; ++j2) {
                            rres[j2] += rmul1[k2] * rmul2[j2];
                        }
                    }
                }
            }
        }
    }
}

```

Figure 6-5: Original and Modified C/C++ Software for Floating-Point Matrix Multiplication.

The BEEBS benchmark software was re-compiled and re-run using the new implementations for both floating-point and integer matrix multiplications.

Table 6-1, page 34 lists a summary of the performance metrics reported for CPU0 in the APU Performance Summary panel in SDK. Even though only 2 of the 8 benchmarks were modified, there was a noticeable difference in the reported metrics. The average IPC value increased by 34.9%, while there was a substantial drop in L1 data cache miss rate. The read stall cycles also decreased dramatically, revealing the reduction in clock cycles when the CPU is waiting on a data cache refill.

Table 6-1: CPU0 Performance Summary with Original and Modified Matrix Multipliers

Performance Metric	CPU0 Performance Summary		
	Original	Modified	Changes
CPU Utilization (%)	100.00	100.00	≈
CPU Instructions Per Cycle	0.43	0.58	↑
L1 Data Cache Miss Rate (%)	8.24	0.64	↓↓
L1 Data Cache Accesses	3484.33M	3653.48M	≈

Table 6-1: CPU0 Performance Summary with Original and Modified Matrix Multipliers

Performance Metric	CPU0 Performance Summary		
	Original	Modified	Changes
CPU Write Stall Cycles Per Instruction	0.00	0.00	≈
CPU Read Stall Cycles Per Instruction	0.79	0.05	↓↓

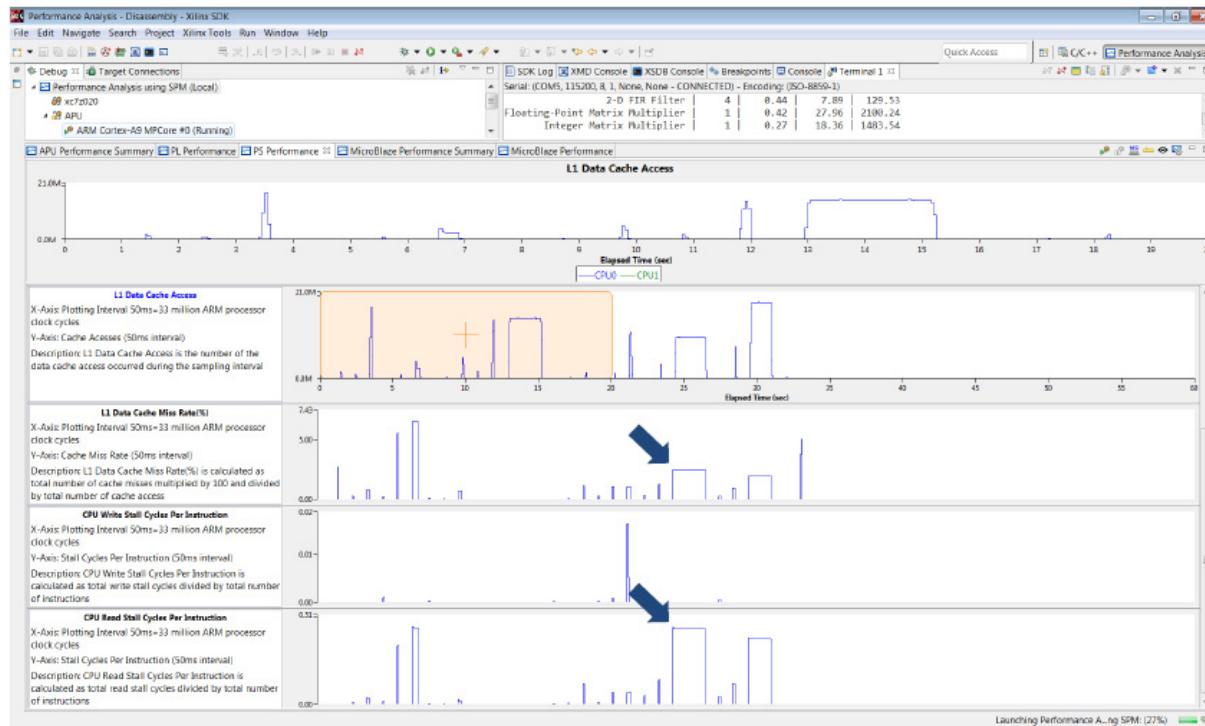


Figure 6-6: Performance Analysis of BEEBS Benchmarks with Improved Matrix Multipliers

Figure 6-6 shows the L1 data cache graphs and CPU stall cycles per instruction reported in the PS Performance panel. In contrast to Figure 6-3, page 32, it becomes clear that the bottlenecks of the code – the floating-point and integer matrix multiplications – have been dramatically improved. The long periods of low IPC, high L1 data cache miss rate, and high CPU read stall cycles toward the end of the capture have been shortened and improved considerably.

This is confirmed with software run-times. A summary of the measured run-times is also listed in Table 6-2. The run-times of the floating-point and integer matrix multiplications were reduced to 22.8% and 17.0% of their original run-times, respectively.

Table 6-2: Summary of Software Run-Times for Original and Modified Matrix Multipliers

Benchmark	Software Run-Times (msec)	
	Original	Modified
Floating-point matrix multiplication	9201.70	2100.36
	100.0%	22.8%
Integer matrix multiplication	8726.67	1483.24
	100.0%	17.0%

Evaluating High-Performance Ports

A great benefit provided by System Performance Modeling (SPM) is to perform what if scenarios of Processing System (PS) and Programmable Logic (PL) activity and their interaction. This is a feature that enables you to explore performance before beginning the design phase, thus reducing the likelihood of finding performance issues late in the design. Although contention on the Zynq®-7000 SoC device is not directly visualized, its impact on system performance is displayed.

You can configure the SPM design to model specific traffic scenarios while running a software application in the PS. You can then use PL master performance metrics, such as throughput and latency, to verify that the desired system performance can be sustained in such an environment. This process can be repeated multiple times using different traffic scenarios.

One critical shared resource on the Zynq-7000 SoC device is the DDR controller. This resource is shared by both CPUs, the four High-Performance (HP) ports, the Accelerator Coherency Port (ACP), and other masters via the central interconnect (see [Chapter 8, Evaluating DDR Controller Settings](#) for more details). Because of this sharing, it is important to understand the available DDR bandwidth.

You can calculate the total theoretical bandwidth of the DDR using the following equation:

$$(533.3 \text{ Mcycles/sec} \times (2 \text{ tranx})/\text{cycle} \times (4 \text{ bytes})/\text{tranx}) = 4270 \text{ MB/sec}$$
 Equation 1

While this is the maximum bandwidth achievable by this memory, the actual DDR utilization is based on many factors, including number of requesting masters and the types of memory accesses. As shown in this chapter, requesting bandwidth that approaches or exceeds this maximum will potentially impact the achieved throughput and latency of all requesting masters. The System Performance Analysis (SPA) toolbox in the Xilinx® Software Development Kit (SDK) aids in this analysis.

HD Video Traffic

The software application used was the Bristol/Embecosm Embedded Energy Benchmark Suite (BEEBS) benchmark program described in [SPM Software, page 15](#). See [Figure 5-1, page 25](#), which shows how this can be specified in SDK. In this scenario, traffic on the four High Performance (HP) ports was injected into the system, and software and hardware performance metrics are measured in SDK. This models a system that is performing

complex algorithms in software while simultaneously processing HD video streams in the PL. Rather than design a system that implements this processing, you can instead model the performance using SPM. You can quickly verify that your desired performance is achieved even before beginning your design.

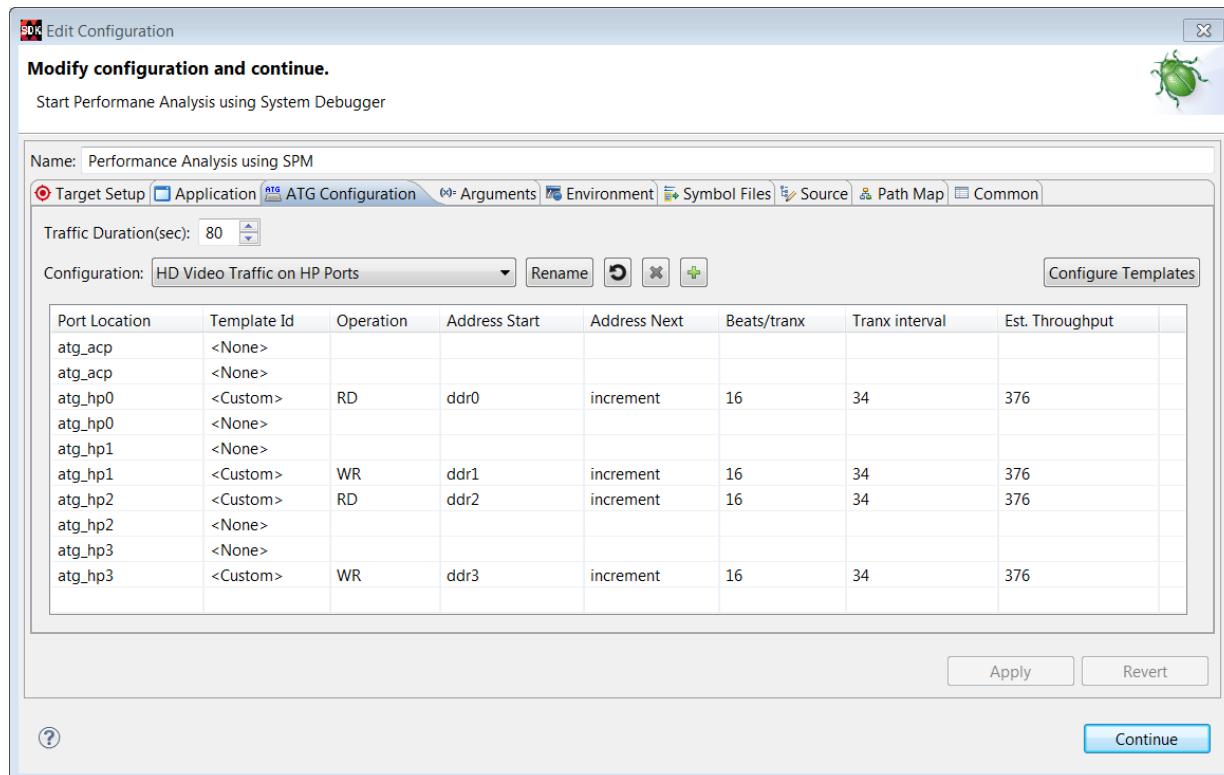
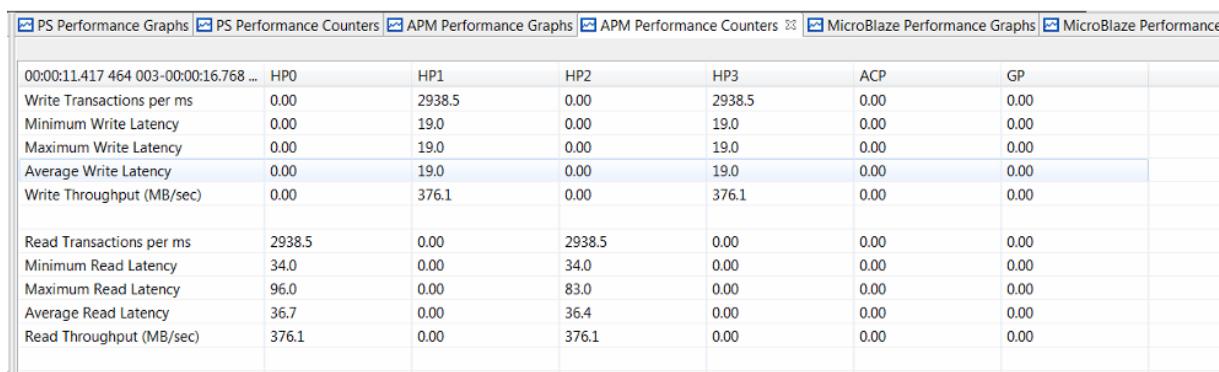


Figure 7-1: ATG Traffic Configuration Modeling HD Video Streams on HP Ports

Figure 7-1 shows the first traffic scenario used (see [Chapter 5, Getting Started with SPM](#) for a description of this traffic specification). This scenario models four uncompressed 1080p/60 (that is, 1080 lines, progressive, and 60 frames/sec) HD video streams. Two streams are being read from the DDR on ports HP0 and HP2, while two are being written on ports HP1 and HP3. For all of these modeled video streams, the Tranx Interval was chosen to request 376 MB/sec, the estimated throughput of the uncompressed RGB 4:4:4 video.



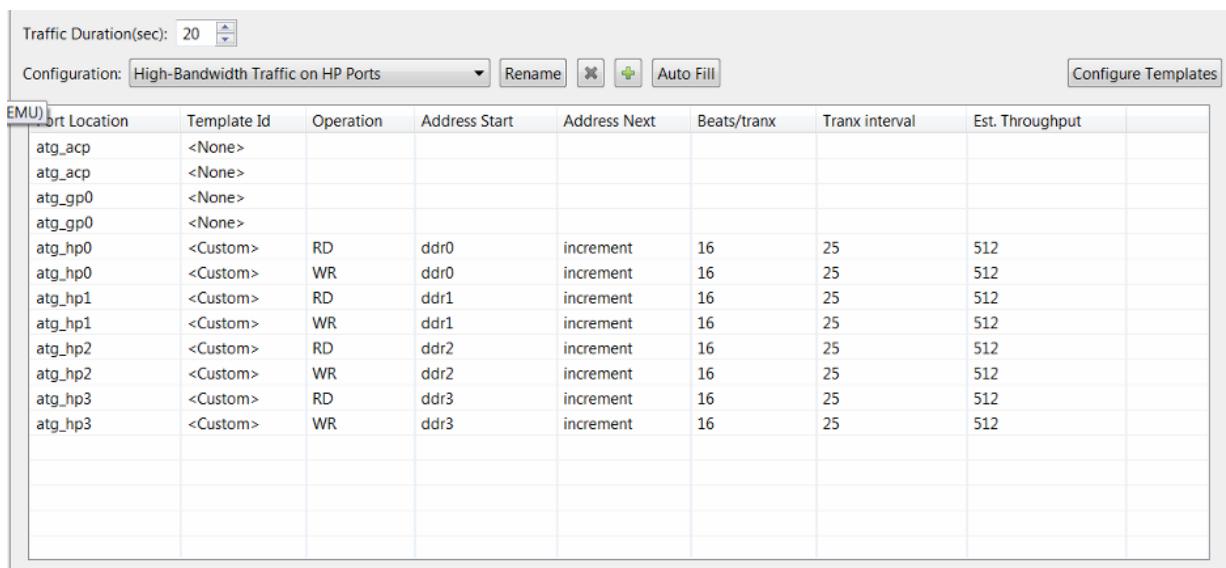
00:00:11:417 464 003-00:00:16.768 ...	HP0	HP1	HP2	HP3	ACP	GP	
Write Transactions per ms	0.00	2938.5	0.00	2938.5	0.00	0.00	
Minimum Write Latency	0.00	19.0	0.00	19.0	0.00	0.00	
Maximum Write Latency	0.00	19.0	0.00	19.0	0.00	0.00	
Average Write Latency	0.00	19.0	0.00	19.0	0.00	0.00	
Write Throughput (MB/sec)	0.00	376.1	0.00	376.1	0.00	0.00	
Read Transactions per ms	2938.5	0.00	2938.5	0.00	0.00	0.00	
Minimum Read Latency	34.0	0.00	34.0	0.00	0.00	0.00	
Maximum Read Latency	96.0	0.00	83.0	0.00	0.00	0.00	
Average Read Latency	36.7	0.00	36.4	0.00	0.00	0.00	
Read Throughput (MB/sec)	376.1	0.00	376.1	0.00	0.00	0.00	

Figure 7-2: Summary of Performance Results from Modeling HD Video Streams on HP Ports

This HD video traffic was run on a ZC702 board. As shown in Figure 7-2, all four HP ports were able to sustain the requested throughput of 376 MB/sec. These high throughputs were even achieved while the BEEBS benchmarks were running on CPU0. The total bandwidth used by the four HP ports was 1510 MB/sec, or 35% of the available bandwidth of the DDR. Thus, the Zynq-7000 SoC device was capable of achieving these throughputs. However, due to arbitration and scheduling in the DDR controller, some contention was introduced, impacting the performance of the software application (look ahead to Figure 7-7, page 43 for a summary). Most benchmarks had minimal impact on their run-time; however, the benchmarks that have a high number of memory accesses were impacted the most. The worst case was the integer matrix multiplication, which had a run-time increase of 14.0%.

High-Bandwidth Traffic

The traffic modeling concept described in [HD Video Traffic, page 37](#) can be extended. You can run the same BEEBS benchmarks but with a different traffic scenario on the HP ports.

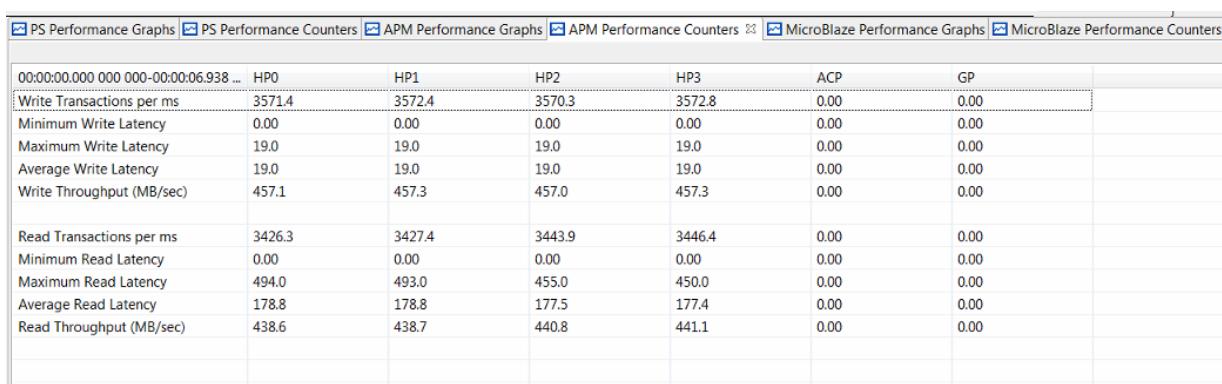


Port Location	Template Id	Operation	Address Start	Address Next	Beats/trnx	Trnx interval	Est. Throughput
atg_acp	<None>						
atg_acp	<None>						
atg_gp0	<None>						
atg_gp0	<None>						
atg_hp0	<Custom>	RD	ddr0	increment	16	25	512
atg_hp0	<Custom>	WR	ddr0	increment	16	25	512
atg_hp1	<Custom>	RD	ddr1	increment	16	25	512
atg_hp1	<Custom>	WR	ddr1	increment	16	25	512
atg_hp2	<Custom>	RD	ddr2	increment	16	25	512
atg_hp2	<Custom>	WR	ddr2	increment	16	25	512
atg_hp3	<Custom>	RD	ddr3	increment	16	25	512
atg_hp3	<Custom>	WR	ddr3	increment	16	25	512

Figure 7-3: ATG Traffic Configuration Modeling High-Bandwidth Traffic on HP Ports

Figure 7-3 shows traffic that executes a stress test to see how high-bandwidth traffic can affect performance. All four HP ports request 512 MB/sec for both read and write traffic using incremental addressing to the DDR. The total duration was set to 80 seconds to ensure that the traffic duration coincided with the entire length of the BEEBS benchmark application. [Figure 6-1, page 30](#) helps orient the timeline for the results.

The total throughput requested by HP masters is $8 * 512 = 4096$ MB/sec, or 95.9% of the theoretical maximum throughput of the DDR. Because the traffic contains both write and read requests, the DDR controller is further stressed because it needs to arbitrate between not just multiple requestors but also different types of requests. Therefore, consider this a *stress test* of the achievable DDR bandwidth. While this traffic represents a worst-case scenario, it is important to perform to visualize the *back pressure* that occurs when more bandwidth is requested than what the memory can provide.



00:00:00.000 000-00:00:06.938 ...	HP0	HP1	HP2	HP3	ACP	GP
Write Transactions per ms	3571.4	3572.4	3570.3	3572.8	0.00	0.00
Minimum Write Latency	0.00	0.00	0.00	0.00	0.00	0.00
Maximum Write Latency	19.0	19.0	19.0	19.0	0.00	0.00
Average Write Latency	19.0	19.0	19.0	19.0	0.00	0.00
Write Throughput (MB/sec)	457.1	457.3	457.0	457.3	0.00	0.00
Read Transactions per ms	3426.3	3427.4	3443.9	3446.4	0.00	0.00
Minimum Read Latency	0.00	0.00	0.00	0.00	0.00	0.00
Maximum Read Latency	494.0	493.0	455.0	450.0	0.00	0.00
Average Read Latency	178.8	178.8	177.5	177.4	0.00	0.00
Read Throughput (MB/sec)	438.6	438.7	440.8	441.1	0.00	0.00

Figure 7-4: Summary of Performance Results from Modeling High-Bandwidth Traffic on HP Ports

Using the SPM design in SDK, this type of stress test can be easily performed. The BEEBS benchmark application was run on CPU0 while the ATGs are configured by SDK to run the requested traffic shown in [Figure 7-3, page 40](#). A summary of the performance results from this traffic scenario is shown in [Figure 7-4](#). As expected, the requested throughput of 512 MB/sec per HP port is not achieved; however, a very high total throughput is indeed achieved. The total read/write bandwidth allocated to the HP ports can be calculated as follows:

$$\begin{aligned} \text{Total bandwidth} &= 457.1 + 457.3 + 457 + 457.3 + 438.6 + 438.7 + 440.8 + 441.1 \\ &= 3587.9 \text{ MB/sec} \end{aligned}$$

Equation 2

While this bandwidth is 84% of the theoretical maximum from [Equation 1](#), there is also the bandwidth allocated to CPU0 running the BEEBS benchmark suite. This stress test confirms the arbitration used by the DDR controller for write and read traffic across four HP ports. Note that exclusively write or read traffic could also be used, increasing the bandwidth performance of the DDR controller.

Since both write and read traffic is being specified, it is important to analyze the performance of both types of transactions. [Figure 7-5, page 41](#) shows the performance of the write AXI transactions in the PL Performance panel. The three graphs shown include: Write Transactions, Average Write Latency, and Write Throughput. The write throughput and latency are relatively consistent on all four HP ports across the entire test run, with a slight drop in the average value and some variance introduced at about 3.2 sec.

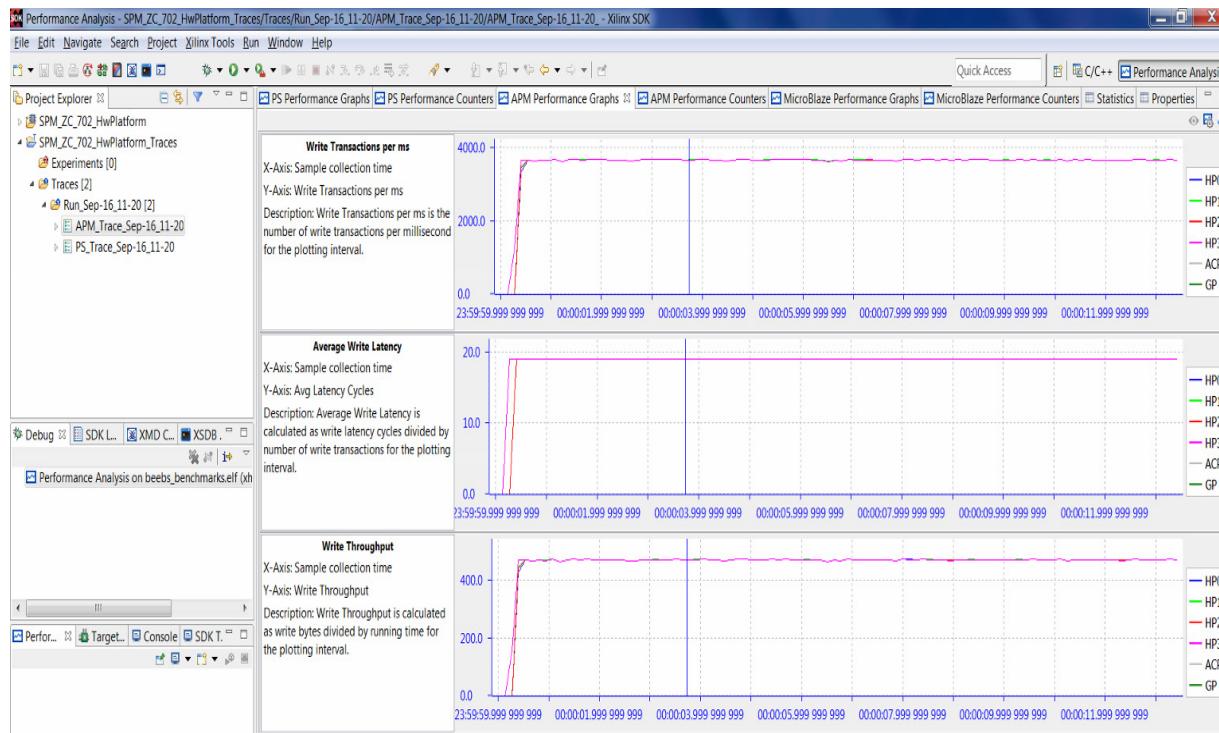


Figure 7-5: Write Performance Results of High-Bandwidth Traffic on HP Ports with BEEBS Benchmarks Running

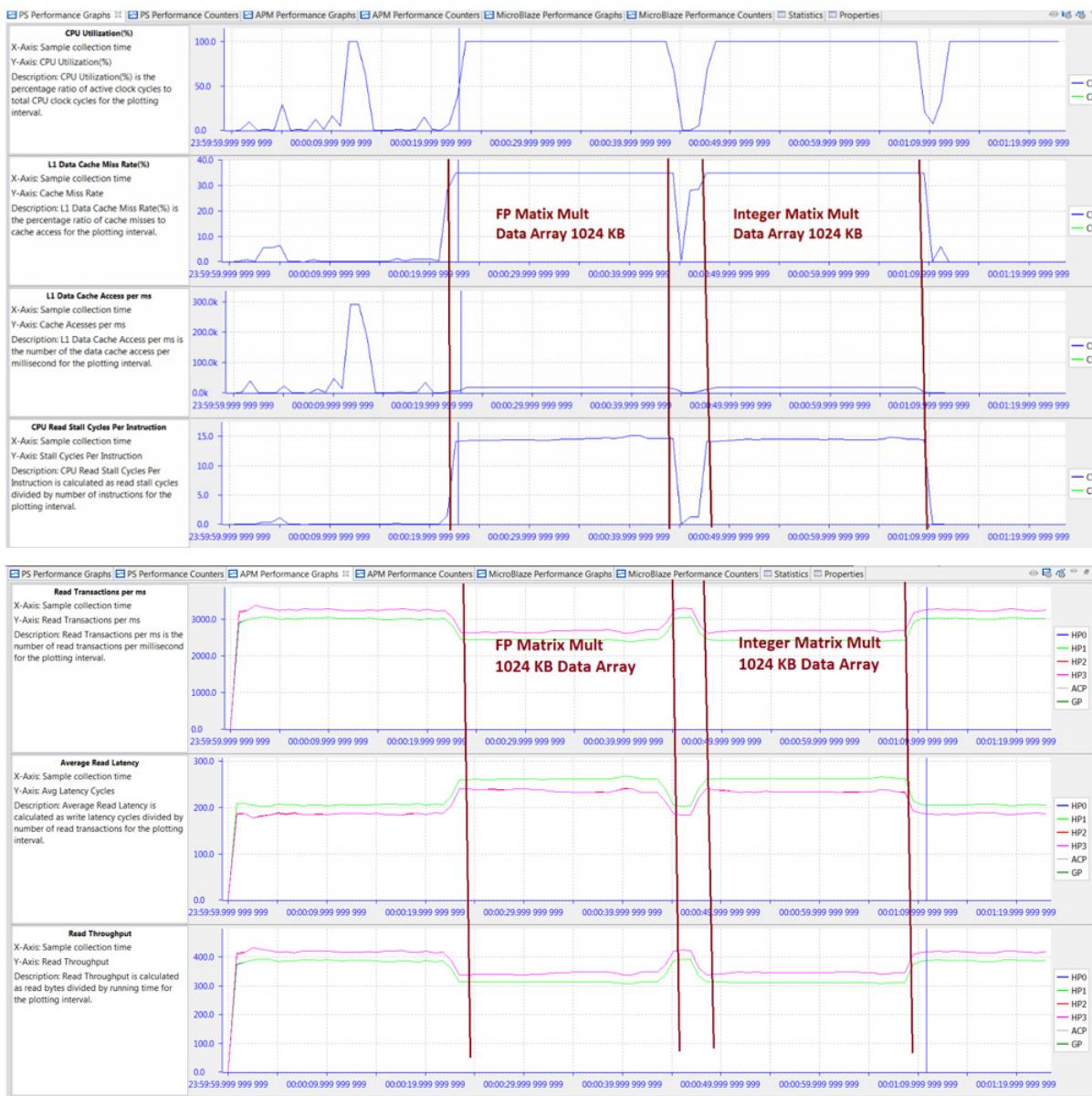


Figure 7-6: Read Performance Results of High-Bandwidth Traffic on HP Ports with BEEBS Benchmarks Running

The read throughput and latency of the HP ports show a more noticeable drop in performance during the matrix multiplications (see Figure 7-6). During all other tests, the read bandwidths are about 420 MB/sec, while during the memory-intensive matrix multipliers (using the 1024 KB data array) the throughput drops to about 340 MB/sec. Hovering the mouse over the graph will confirm the fact. At an elapsed time of 64 sec, the read throughputs of the four HP ports are all about 338 MB/sec. This decrease in throughput coincides with an increase in latency, both caused by saturation at the DDR controller.

You can also use the **Trace Tooltip** button to report performance metric values at a specific point in time. When the tooltip is shown, you can move it to any point in time on any graph, and it will display the specified metric values. This is very helpful in pinpointing the performance at precise points in time.

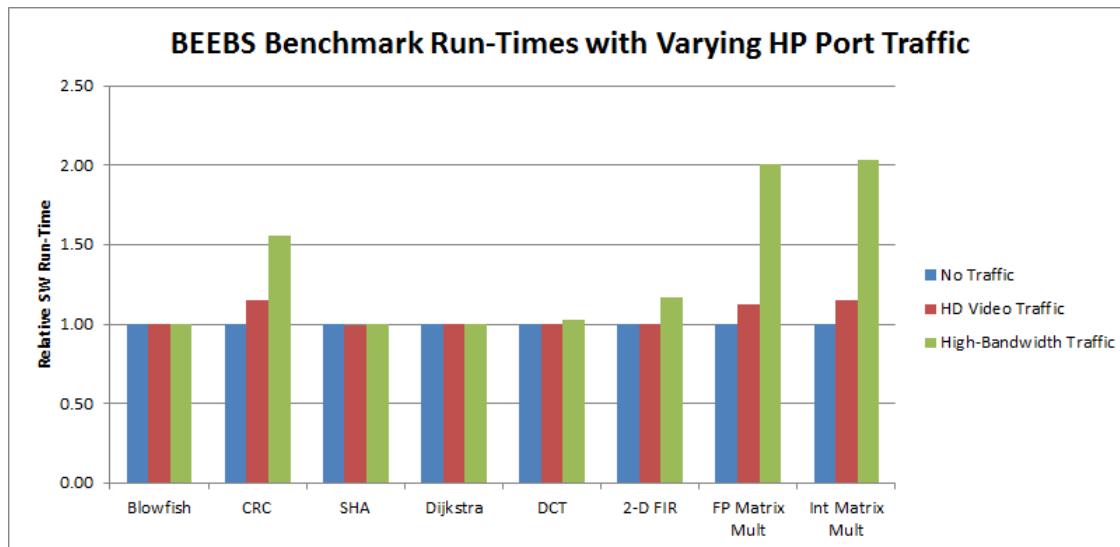


Figure 7-7: Relative Run-Times of BEEBS Benchmarks with Varying HP Port Traffic (Data Array Size: 1024 KB)

The impact of increased HP traffic can be seen in the software performance. A great overall metric of this performance is run-time. Using the instrumentation (see [Instrumenting Hardware, page 61](#)) that was added to the BEEBS software, [Figure 7-7](#) shows the relative run-times of the eight benchmarks using the 1024 KB data array size. The run-time of each benchmark with no traffic was normalized to 1.0. Because this large array requires accesses to the DDR, any increase in software run-time would be attributed to the DDR saturation. The matrix multiplications experience the largest increases in software run-times. The Cyclic Redundancy Check (CRC) algorithm also has a noticeable impact in run-time, most likely due to the increased memory reads of the data array as well as the CRC polynomial table. All other benchmarks see minimal impact on performance, even with the high-throughput traffic on all four HP ports.

Evaluating DDR Controller Settings

The Zynq-7000 SoC family of devices offers some control over the allocation of bandwidth to the DDR memory. These settings are provided in the on-chip DDR controller (DDRC). Starting in the 2014.4 version of SDK, you can modify clock frequencies and DDRC settings for the SPM design. This chapter shows that while bandwidth cannot necessarily be created, it can indeed be re-allocated.

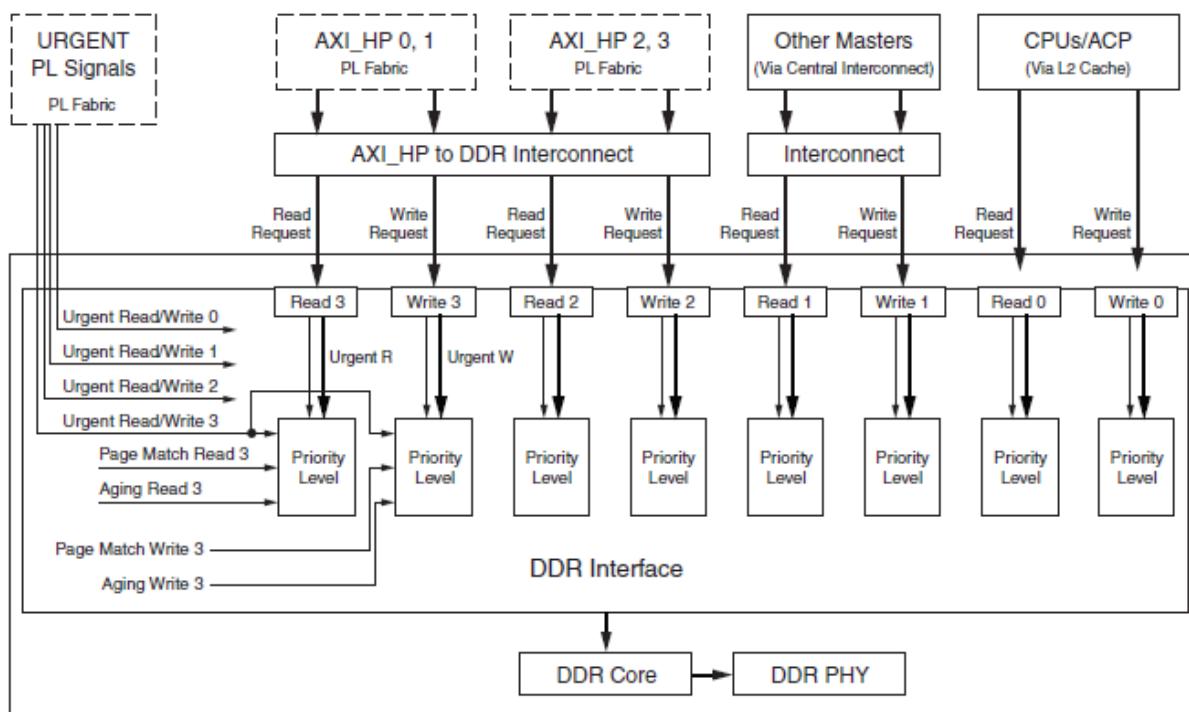


Figure 8-1: Block Diagram of Zynq-7000 SoC DDR Controller

It is first important to understand the connectivity to the DDRC. Figure 8-1 shows the block diagram of the Zynq-7000 SoC DDRC. There are four ports on the DDRC which service traffic from the CPUs and the Accelerator Coherency Port (ACP) (port 0), other masters via the central interconnect (port 1), HP ports HP2 and HP3 (port 2), and HP ports HP0 and HP1 (port 3). The settings for the DDRC are part of the PS7 or First Stage Boot Loader (FSBL) which configures various system settings early in the boot process. For more information, see the "DDR Memory Controller" chapter of the *Zynq-7000 SoC Technical Reference Manual* (UG585) [\[Ref 1\]](#).

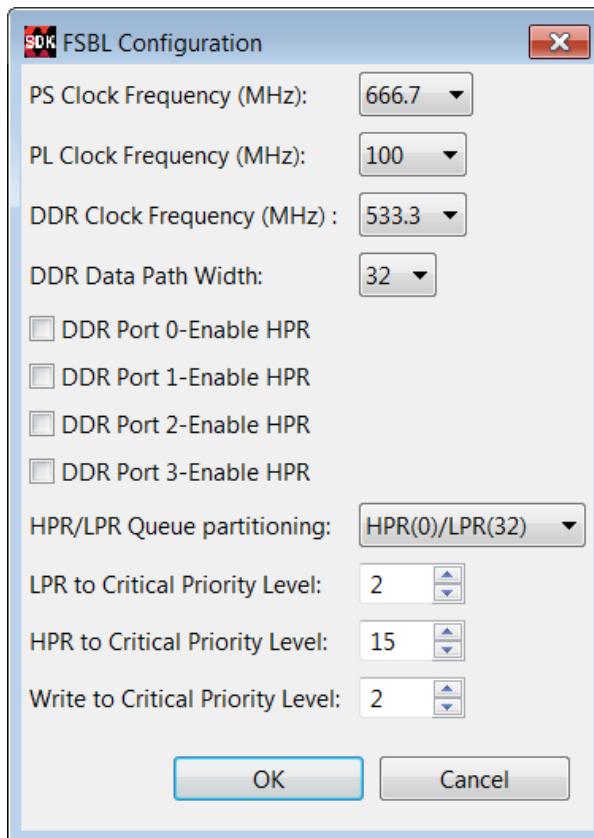


Figure 8-2: Default FSBL Configuration Settings for SPM Design

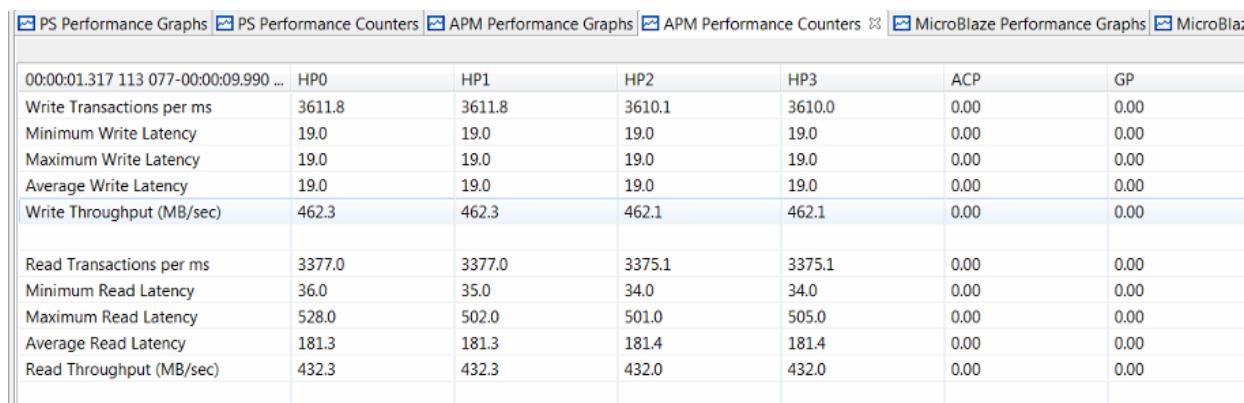
From the SDK Project Explorer, select the System Performance Modeling (SPM) project (default name is SPM_ZC_702_HwPlatform), and then right-click and select **Configure FSBL Parameters**. Figure 8-2 shows the dialog box that opens, containing the default FSBL settings for the SPM design. The settings include the following:

- **(PS/PL/DDR) Clock Frequency (MHz)** – you can specify clock frequencies for the processing system (PS), programmable logic (PL), and DDR.
- **DDR Data Path Width** – you can specify a data path width of 16 or 32 bits.
- **DDR Port (0/1/2/3)-Enable HPR** – you can enable high-performance reads (HPR) for the four DDR ports.
- **HPR/LPR Queue partitioning** – The DDRC contains a queue for read requests with a constant total depth of 32 requests. You can specify how this queue is allocated across HPR and low-priority reads (LPR).
- **(LPR/HPR/Write) to Critical Priority Level** - The last three values shown in Figure 8-2 are critical levels which specify the number of clocks (unit is 32 DDR clock cycles) that the three different queues (that is, LPR, HPR, and Write) can be starved before they go critical. Once a critical state is reached, that queue becomes top priority. That is, a smaller value means higher priority for that queue.

These DDRC settings can also be modified in the Vivado® Design Suite. In a Vivado IP integrator based design, you can double-click on the Zynq7 Processing System IP block in the block diagram. A configuration dialog box opens, in which you can re-customize the IP. The DDR Controller settings are listed under **DDR Configuration**, and the specific options of interest are under **Enable Advanced options**. When that box is checked, you can view and modify settings such as read/write priority, high-priority read (HPR), and other important settings that can impact performance. The functionality of these settings is described in depth in the *Zynq-7000 SoC Technical Reference Manual* (UG585) [Ref 1].

Default DDRC Settings

Baseline results were created by running *HP only* traffic using the default FSBL settings, as shown in [Figure 8-2, page 45](#). The PL traffic was the high-bandwidth HP traffic shown in [Figure 7-3, page 40](#); however, the traffic duration was set to 10 sec. Also, no software application was run. You can configure by modifying the Debug Configuration. Select the **Application** tab, and then uncheck the **Download Applications** check box.



00:00:01.317 113 077-00:00:09.990 ...	HP0	HP1	HP2	HP3	ACP	GP
Write Transactions per ms	3611.8	3611.8	3610.1	3610.0	0.00	0.00
Minimum Write Latency	19.0	19.0	19.0	19.0	0.00	0.00
Maximum Write Latency	19.0	19.0	19.0	19.0	0.00	0.00
Average Write Latency	19.0	19.0	19.0	19.0	0.00	0.00
Write Throughput (MB/sec)	462.3	462.3	462.1	462.1	0.00	0.00
Read Transactions per ms	3377.0	3377.0	3375.1	3375.1	0.00	0.00
Minimum Read Latency	36.0	35.0	34.0	34.0	0.00	0.00
Maximum Read Latency	528.0	502.0	501.0	505.0	0.00	0.00
Average Read Latency	181.3	181.3	181.4	181.4	0.00	0.00
Read Throughput (MB/sec)	432.3	432.3	432.0	432.0	0.00	0.00

Figure 8-3: HP Port Performance Using Default DDR Configuration Settings

This traffic scenario was run on a ZC702 board, and the results can be seen in [Figure 8-3](#). Note that default DDRC settings were used because they have not been modified yet. The HP ports sustained a higher bandwidth than the previous results shown in [Figure 7-4, page 40](#) because there is no DDR traffic from the CPUs.

The total DDR bandwidth is therefore the sum of the bandwidths measured on the HP ports, calculated as follows:

$$\begin{aligned}\text{Total bandwidth} &= 462.3 + 462.3 + 462.1 + 462.1 + 432.3 + 432.3 + 432.0 + 432.0 \\ &= 3577.4 \text{ MB/sec}\end{aligned}$$

Equation 1

This total bandwidth is 83.7% of the theoretical maximum throughput of the DDR. This is very good performance considering the mix of high-bandwidth reads and writes from four different HP ports.

Modified DDRC Settings

You can modify the DDRC settings and re-run the HP only test above to compare performance results. Note that while modifying the DDRC settings is important for this stress test, it may not be desirable for your design. In fact, most designs will achieve desired performance using the default DDRC settings.

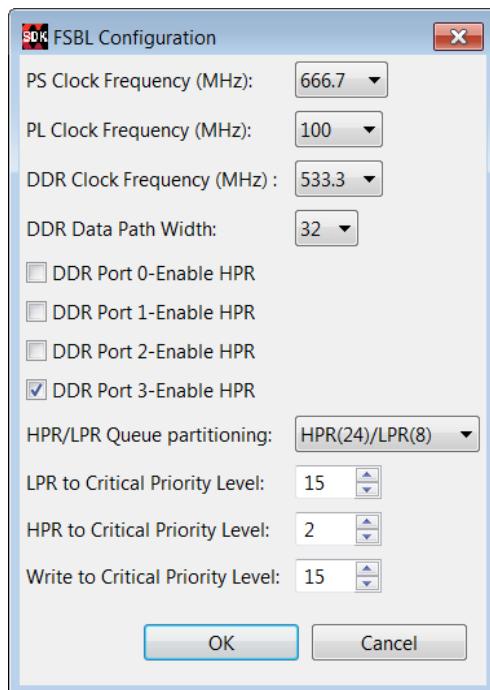


Figure 8-4: Modified FSBL Configuration Settings for SPM Design

Consider the scenario where high priority is desired for the read traffic on HP0 and HP1. In a Zynq-7000 SoC, these HP ports are serviced by DDR port 3 (see [Figure 8-1, page 44](#)). As shown in [Figure 8-4](#), this can be accomplished by adding HPR to DDR port 3. The request queue was also partitioned by assigning 24 requests for HPR and 8 requests for LPR. The HPR requests were further prioritized by decreasing its critical priority level to 2 while increasing the LPR and write critical levels to 15. This was an important step to ensure the HPRs received the desired priority.

00:00:03.624 950 326-00:00:09.176 ...						
	HP0	HP1	HP2	HP3	ACP	GP
Write Transactions per ms	2820.7	2820.7	2819.2	2819.3	0.00	0.00
Minimum Write Latency	19.0	19.0	19.0	19.0	0.00	0.00
Maximum Write Latency	19.0	19.0	19.0	19.0	0.00	0.00
Average Write Latency	19.0	19.0	19.0	19.0	0.00	0.00
Write Throughput (MB/sec)	361.0	361.0	360.9	360.9	0.00	0.00
Read Transactions per ms	3995.0	3995.0	3220.6	3220.6	0.00	0.00
Minimum Read Latency	34.0	34.0	34.0	35.0	0.00	0.00
Maximum Read Latency	261.0	258.0	738.0	742.0	0.00	0.00
Average Read Latency	47.4	48.3	168.7	168.5	0.00	0.00
Read Throughput (MB/sec)	511.4	511.4	412.2	412.2	0.00	0.00

Figure 8-5: HP Port Performance Using Modified DDR Controller Settings

Figure 8-5 shows the results in SDK after modifying the DDRC settings and re-running the same HP only traffic. HP0 and HP1 were able to achieve their requested read throughputs of 512 MB/sec due to the high priority of their transactions at the DDR controller. However, the achieved throughputs of all other traffic (read requests for HP2 and HP3 and all writes) were reduced. This was due to the low priority of those transactions – in other words, just as was specified in the DDRC settings. The total bandwidth of this test can be calculated as follows:

$$\begin{aligned} \text{Total bandwidth} &= 361 + 361 + 360.9 + 360.9 + 511.4 + 511.4 + 412.2 + 412.2 \\ &= 3291 \text{ MB/sec} \end{aligned}$$

Equation 2

Note that this total bandwidth is within 2.3% of the total bandwidth achieved with the default DDRC settings (summarized in Figure 8-3, page 46 and totaled in Equation 1, page 45). This is very important to observe: changing the DDRC settings does not create bandwidth but only re-allocates it. That is, a larger percentage of the total bandwidth issued by the DDR controller was re-allocated to the requested high priority ports: HP0 and HP1. Meanwhile, the total bandwidth stayed relatively constant.

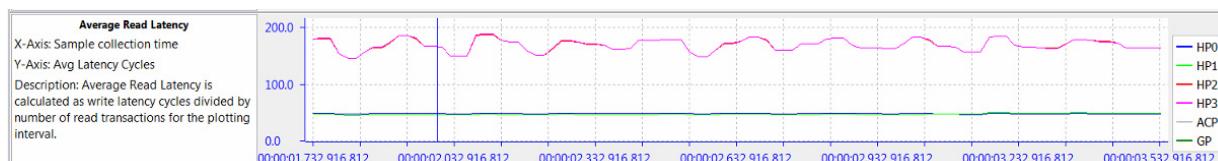


Figure 8-6: Read Latency of HP Ports Using Modified DDR Controller Settings

Another way to observe this re-allocation of priorities can be seen in the read latency graph shown in Figure 8-6. Since the DDRC is configured to provide high priority to read traffic on HP0 and HP1, the latency on those ports is relatively low. That is, read requests on those ports are being serviced by the DDRC at a higher priority. On the other hand, read requests on the other HP ports, HP2 and HP3, are being treated as low priority and therefore, their latency is much higher. The impact of this is increased throughput on HP0 and HP1 and decreased throughput on HP2 and HP3.

Utilizing On-Chip Memory

Consider the case where it is a design requirement to obtain 512 MB/sec for both write and read traffic from all HP ports. How do you leverage other memory to augment the DDRC bandwidth? This would require a total bandwidth of $8 * 512 = 4096$ MB/sec, or 96.0% of the theoretical maximum throughput of the DDR. While this is not achievable from a typical DDR controller and memory with an even mix of writes and reads, the On-Chip Memory (OCM) can be explored for additional data buffering. This is an independent 256 KB memory within the Zynq-7000 SoC PS that provides an excellent low-latency scratch pad.

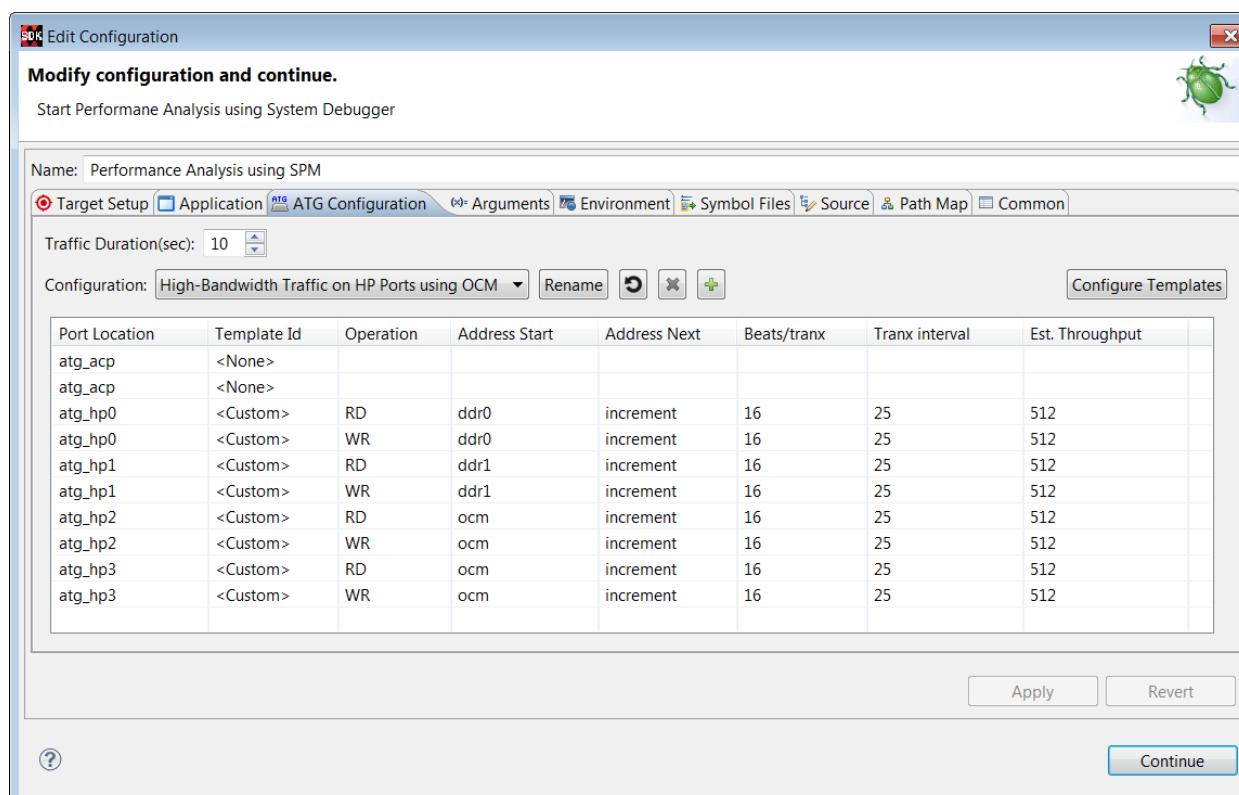


Figure 8-7: ATG Traffic Configuration Modeling High-Bandwidth Traffic on HP Ports Using Both DDR and OCM

Figure 8-7 shows one particular traffic scenario that utilizes the OCM. Compare this scenario to the original one shown in [Figure 7-3, page 40](#). While the requested throughputs are kept the same for all operations and types, the traffic on HP2 and HP3 now uses the OCM.

00:00:08.667 596 622-00:00:25.608 ...	HP0	HP1	HP2	HP3	ACP	GP
Write Transactions per ms	3995.0	3995.0	3995.0	3995.0	0.00	0.00
Minimum Write Latency	19.0	19.0	19.0	19.0	0.00	0.00
Maximum Write Latency	19.0	19.0	19.0	19.0	0.00	0.00
Average Write Latency	19.0	19.0	19.0	19.0	0.00	0.00
Write Throughput (MB/sec)	511.4	511.4	511.4	511.4	0.00	0.00
Read Transactions per ms	3995.0	3995.0	3995.0	3995.0	0.00	0.00
Minimum Read Latency	37.0	35.0	31.0	31.0	0.00	0.00
Maximum Read Latency	74.0	77.0	32.0	32.0	0.00	0.00
Average Read Latency	40.5	43.0	31.9	31.9	0.00	0.00
Read Throughput (MB/sec)	511.4	511.4	511.4	511.4	0.00	0.00

Figure 8-8: Summary of Performance from High-Bandwidth Traffic on HP Ports Using Both DDR and OCM

The results of such a traffic scenario are shown in Figure 8-8. Since the OCM was used to augment the DDR bandwidth, the desired bandwidth of 512 MB/sec was sustained for both write and read operations on all four HP ports. Also noteworthy, the average read latency from the OCM is 31-32 cycles while the average from the DDR is 40-44 cycles. Note that these latency values depend on many factors in a system, so it is important to utilize the SPM design and analyze under various conditions.

Certainly, you could exploit the bandwidth re-allocation provided by modifying DDRC settings. For example, you may have higher bandwidth or lower latency requirements for a given IP core in your programmable logic design. Once you know which HP port to connect that core to (or at least plan to connect to) you can allocate more bandwidth to that particular HP port pair. For example, in the [Chapter 11, End-To-End Performance Analysis](#), a multi-layer video display controller may have higher bandwidth requirements than other AXI masters in the system.

There are other combinations of HPRs and critical levels that could work well for other design criteria and constraints. For example, you could use these settings to improve the run-time performance of your software by increasing the priority of DDR port 0. This port is connected to the L2 cache, which in turn services the CPUs and the ACP port.

With the performance analysis capabilities provided in SDK, you now have the ability to quickly and visually verify these performance results.

Evaluating Memory Hierarchy and the ACP

The System Performance Modeling (SPM) design can be used to evaluate the performance effects of the Zynq®-7000 SoC cache. You can assess baseline performance using software only, and then you can evaluate the effects of traffic on the Accelerator Coherency Port (ACP). While running software only can produce fairly predictable memory performance, introducing other traffic activity into the system may lead to less intuitive results. Specifically, CPU and ACP traffic have the potential to impact each other at the shared L2 cache. The SPM design allows you to do this analysis very easily.

Assess Memory Performance

To begin the evaluation, the SPM design can be used to assess a baseline performance using software only. An excellent application to use for this evaluation is the memory stride benchmarks program, one of the pre-compiled executables that comes with the SPM project.

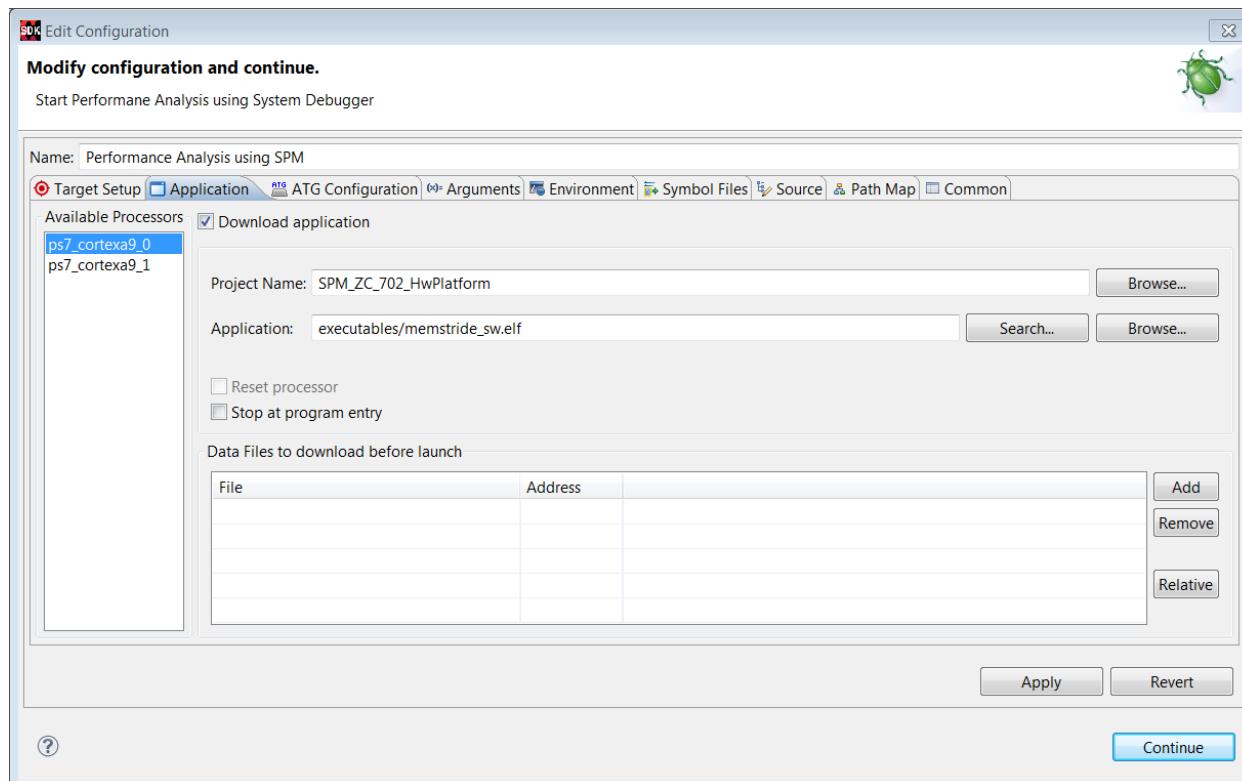


Figure 9-1: Application Setup Using Memory Stride Software

Figure 9-1 shows how this was defined in SDK using the memory stride software executable (see [SPM Software, page 15](#)). This software was instrumented to calculate bandwidth and average latency (see [Instrumenting Software, page 60](#)).

Table 9-1: Software-Only Bandwidth Results for Memory Stride Benchmarks (in MB/sec)

Pattern Type	Operation Type	Working Set Size		
		4 KB	64 KB	1024 KB
Linear	Read	2141.4	338.4	118.1
	Write	2114.8	238.4	57.6
	Copy	1248.4	126.7	32.9
	Read/Write	1194.4	265.2	64.4
Random (Pre-Computed)	Read	1019.9	325.9	92.4
	Write	532.4	285.6	64.1
	Copy	442.6	140.6	31.9
Random (Real-Time)	Read	337.2	138.7	41.5
	Write	409.8	393.6	70.0
	Copy	409.7	159.4	32.8

Table 9-2: Software-Only Latency Results for Memory Stride Benchmarks (in nsec)

Pattern Type	Operation Type	Working Set Size		
		4 KB	64 KB	1024 KB
Linear	Read	1.87	11.82	33.88
Random (Pre-Computed)	Read	1.89	16.78	69.46

Using no traffic in the Programmable Logic (PL), bandwidth results (in MB/sec) for the software-only tests are listed in [Table 9-1](#), while latency results (in nsec) are shown in [Table 9-2](#). Three different working set sizes were used to specifically test the bandwidth of different stages of the memory hierarchy: 4 KB, 64 KB, and 1024 KB. These are the three sizes used by the pre-compiled memory stride executable in the SPM project. The locality of these data arrays is known based on their size and the fact that there is little else in the heap used by the program.

Within a working set size, the highest bandwidths are achieved by the linear bandwidth tests as they are able to take advantage of the 32-byte cache lines. Operations such as copy and read/write are generally slower because they involve a mix of read and write operations. Among the three different set sizes, the 4 KB array sustains the highest bandwidth while the 1024 KB array sustains the lowest bandwidth.

There are 36 total tests performed, as the 12 benchmarks listed in [Table 9-1](#) and [Table 9-2](#) are each run on the three different data array sizes. Since a sleep time of 1 second was inserted between each test, the CPU utilization gives a clear view of when these benchmarks were run. [Figure 9-2](#) helps orient the timeline for the results. The three data sizes were run from smallest to largest within each benchmark, which can be seen in the value and length of the utilization of CPU0.

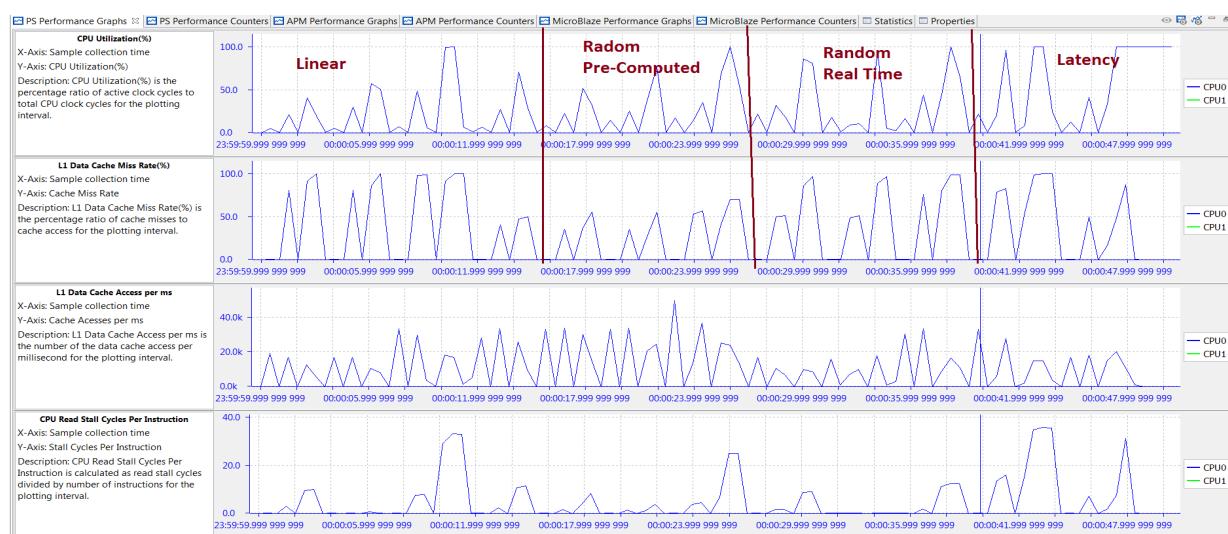


Figure 9-2: PS Performance Graphs for Memory Stride Benchmarks

Figure 9-2 also shows three other graphs reported in the PS Performance panel, including: L1 Data Cache Miss Rate, and CPU Write/Read Stall Cycles Per Instruction. While the periodic nature of the memory stride benchmarks is reflected in these graphs, these three graphs can also provide information as to where the data is being retrieved. Every third benchmark has a L1 data cache miss rate of 0% - that was when the 4 KB working set size was used. The other two set sizes do not fit in the L1 data cache and, therefore, have at or near 100% miss rates.

The software impact of these cache misses can be seen in the Stall Cycles Per Instruction. A higher number of stall cycles means the processor is spending more cycles waiting for either a memory write (write) or a data cache refill (read). This would, in turn, lower the effective bandwidth of that benchmark. An example is the *Linear Bandwidth – Copy* benchmark which was performed between approximately 10-11 sec. in Figure 9-2. The high number of CPU write/read stall cycles per instruction during this time shows why it sustains one of the lowest bandwidths of all memory stride benchmarks (33.23 MB/sec; see Table 9-1, page 52).

Data Size and Locality

To better visualize the performance of the memory hierarchy, the working set size used in the memory stride tests was swept across a range of values between 1 KB and 4096 KB. This enables a clear view of how data size and locality can affect the bandwidth and latency achieved by the CPU.

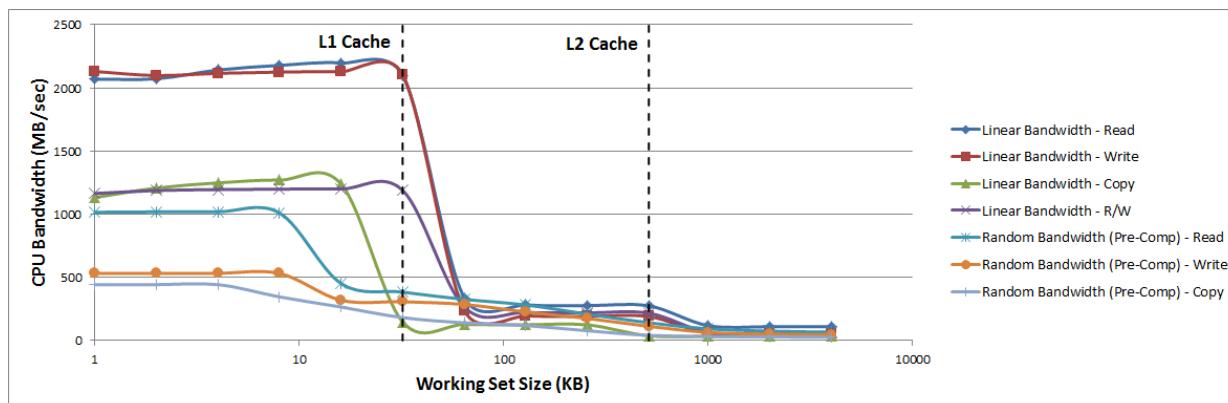


Figure 9-3: CPU Memory Bandwidth Achieved During Memory Stride Tests

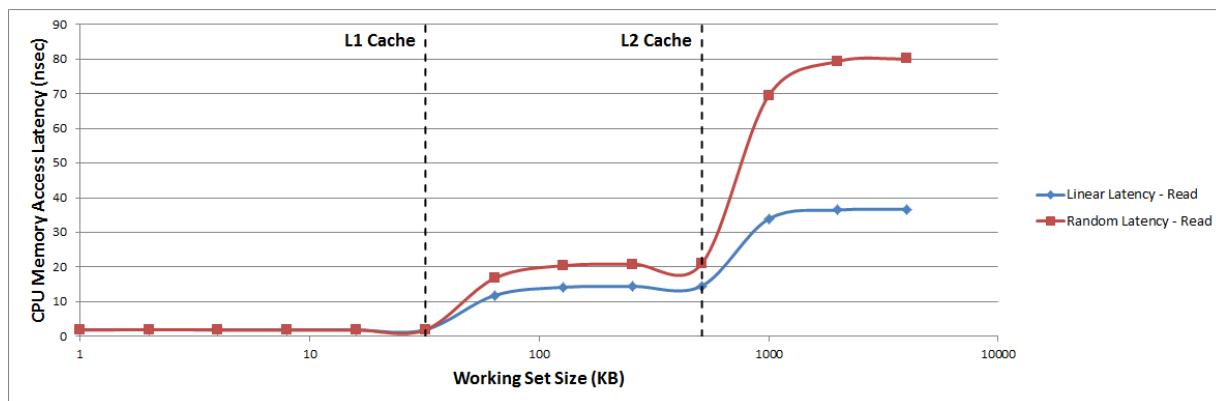


Figure 9-4: CPU Memory Access Latency Achieved During Memory Stride Tests

Figure 9-3, page 54 shows the memory bandwidth achieved by CPU0 during the memory stride tests. For clarity purposes, the random bandwidth (real-time) tests are not shown in the graph. As the working set size increases, there are noticeable transitions at the L1 and L2 data cache boundaries as the working set size no longer fits into the previous cache. Figure 9-4 shows similar transition points for the average memory access latency achieved by CPU0 during the memory stride tests. However, while the bandwidth decreases with an increasing working set size, the latency noticeably increases.

Shared L2 Cache

Now that baseline performance measurements have been made between the CPUs and the memories, traffic can be added on the Accelerator Coherency Port (ACP) to analyze its effects on system performance. The memory stride software (see [SPM Software, page 15](#)) was run again on CPU0; however, this time traffic was added to the ACP. Since the L2 cache is shared by the CPUs and the ACP, there is expected to be some contention to occur at that memory.

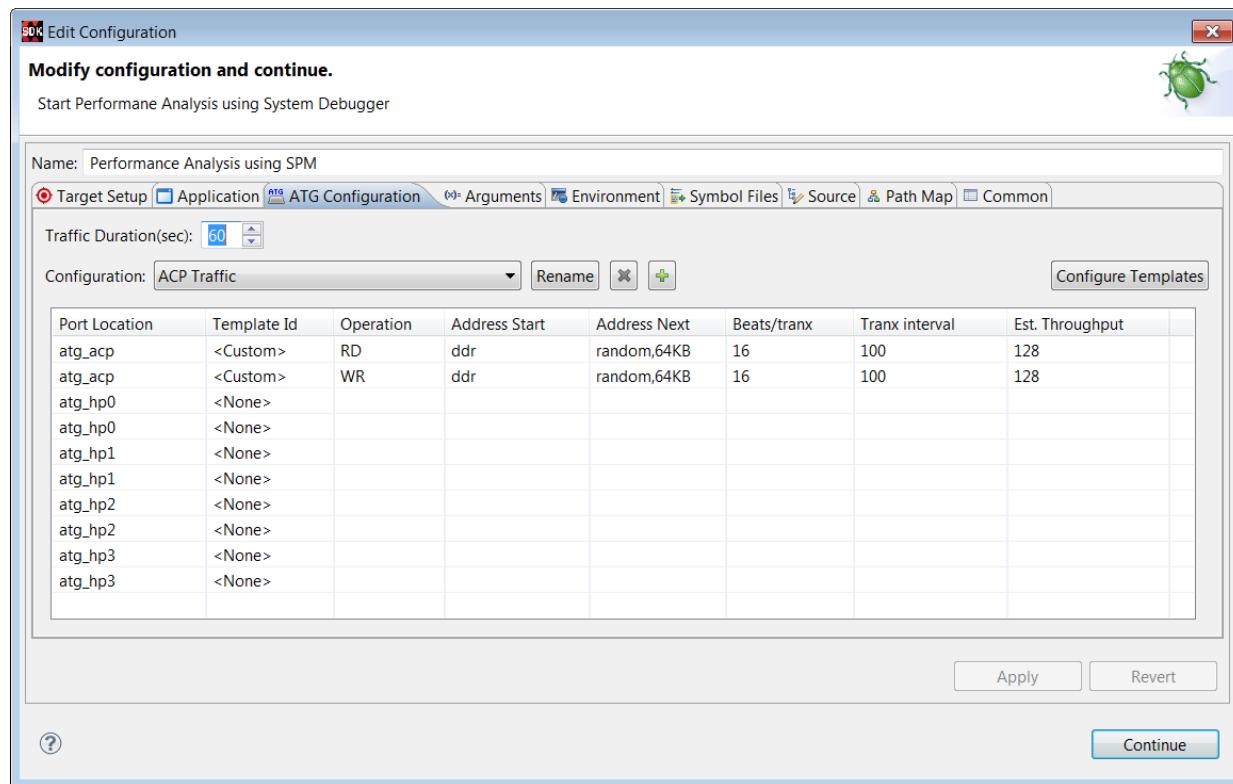


Figure 9-5: ATG Configuration Modeling Traffic on ACP

Figure 9-5 shows the AXI Traffic Generator (ATG) configuration that models the initial traffic on the ACP. It specifies random addressing within a 4 KB region and utilizes the memory path through the L2 cache to the DDR. The first value for *Tranx Interval* is 100 clock cycles, which configures the ATGs to request a throughput of 128 MB/sec at 100 MHz. Keeping every other setting constant, the values for *Tranx Interval* were decreased, thus increasing the requested throughputs. The values used for this transaction interval include: 100, 50, 30, 25, 20, and 17. These intervals lead to request throughputs of 128, 256, 426, 512, 640, and 752 MB/sec.

This traffic scenario enables an exploration of bandwidth tradeoffs between the CPUs and the ACP. Note that this models the case where the CPU and ACP are not using a shared memory space. Similarly to the DDR, the L2 data cache allocates its bandwidth based on scheduling. If the combined bandwidth is higher than what the L2 cache can support, then contention in the form of back pressure is introduced.

CPU Write Stall Cycles Per Instruction		
00:00:0.934 020 085-00:00:41.696 ...	CPU0	CPU1
CPU Utilization(%)	24.6	0.00
CPU Instructions Per Cycle	0.15	0.00
L1 Data Cache Miss Rate(%)	44.3	0.00
L1 Data Cache Access per ms	11.9k	0.00
CPU Write Stall Cycles Per Instruction	2.79	0.00
CPU Read Stall Cycles Per Instruction	4.29	0.00

Figure 9-6: Performance Summary for Memory Stride and No ACP Traffic

CPU Write Stall Cycles Per Instruction		
00:00:6.461 933 857-00:00:27.637 ...	CPU0	CPU1
CPU Utilization(%)	24.4	0.00
CPU Instructions Per Cycle	0.13	0.00
L1 Data Cache Miss Rate(%)	37.6	0.00
L1 Data Cache Access per ms	14.3k	0.00
CPU Write Stall Cycles Per Instruction	3.35	0.00
CPU Read Stall Cycles Per Instruction	5.64	0.00

Figure 9-7: Performance Summary for Memory Stride and ACP Traffic Requesting 752 MB/sec

Figure 9-6, page 57 shows the performance summary table in SDK after running memory stride with no ACP traffic, while Figure 9-7 shows the same summary table with memory stride running plus read/write traffic on the ACP requesting 752 MB/sec (Tranx Interval of 17). These are the two bounds of the test, and the differences are somewhat noticeable. While the ACP traffic is clearly evident in Figure 9-7, the CPU statistics are generally very similar.

One difference is in the *CPU Write Stall Cycles Per Instruction* and the *CPU Read Stall Cycles Per Instruction*. While these values are 2.79 and 4.29 with no ACP traffic, they increase to 3.35 and 5.64, respectively, with high-bandwidth ACP traffic. High values for these two metrics means the CPU is stalling often on memory writes and data cache refills. The reason for the increase in these values is the contention at the L2 cache, since the ACP traffic is consuming a significant amount of L2 cache bandwidth.

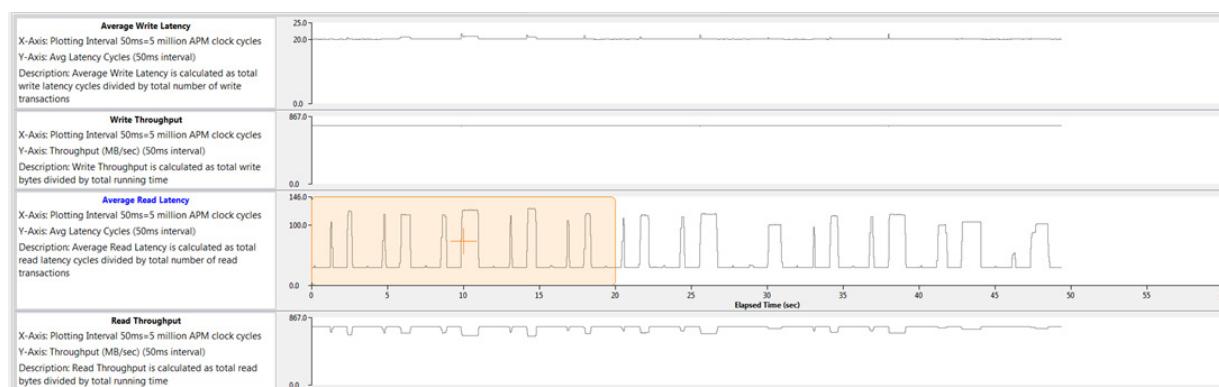


Figure 9-8: ACP Performance Results in SDK Showing Impact on Latency

Figure 9-8 shows how the write and read performance on the ACP is affected by this contention. There is minimal impact on the ACP write performance, with only slight increases in latency during the various software benchmarks. However, the average read latency is clearly affected by the memory stride software activity, showing a periodicity reflecting the sequence of benchmarks. The ACP throughput is also affected, dropping to a worst-case value of approximately 640 MB/sec. While the requested bandwidth of 752 MB/sec is perhaps unrealistically high and persistent for a typical design, it does provide a good stress test of the system.

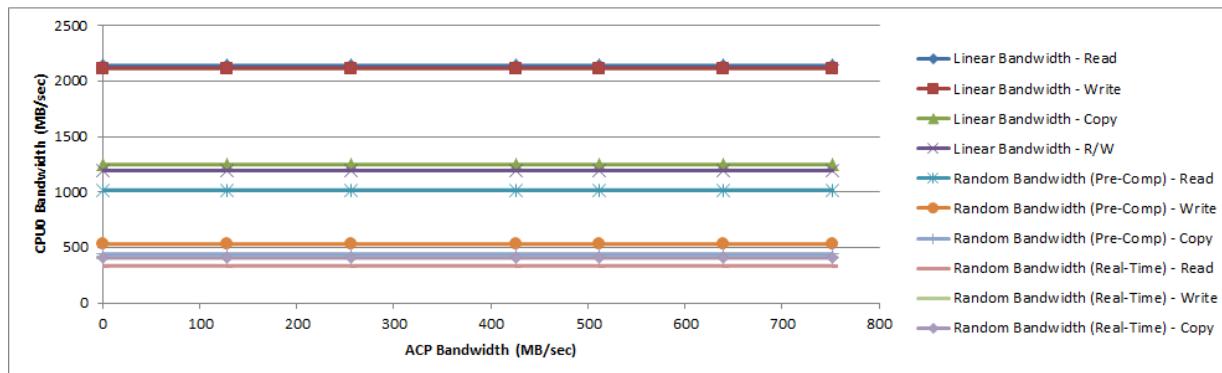


Figure 9-9: Performance Achieved by CPU0 vs. ACP Bandwidth (Set Size = 4 KB)

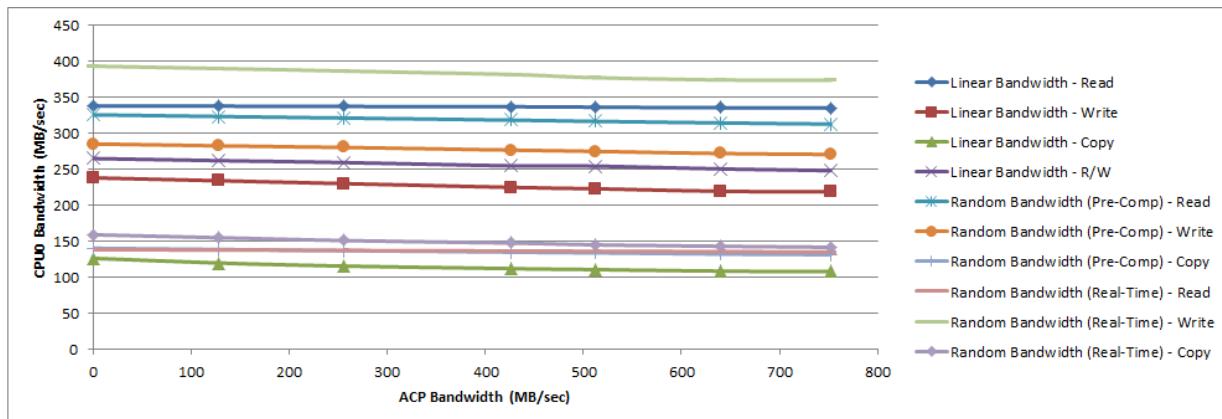


Figure 9-10: Performance Achieved by CPU0 vs. ACP Bandwidth (Set Size = 64 KB)

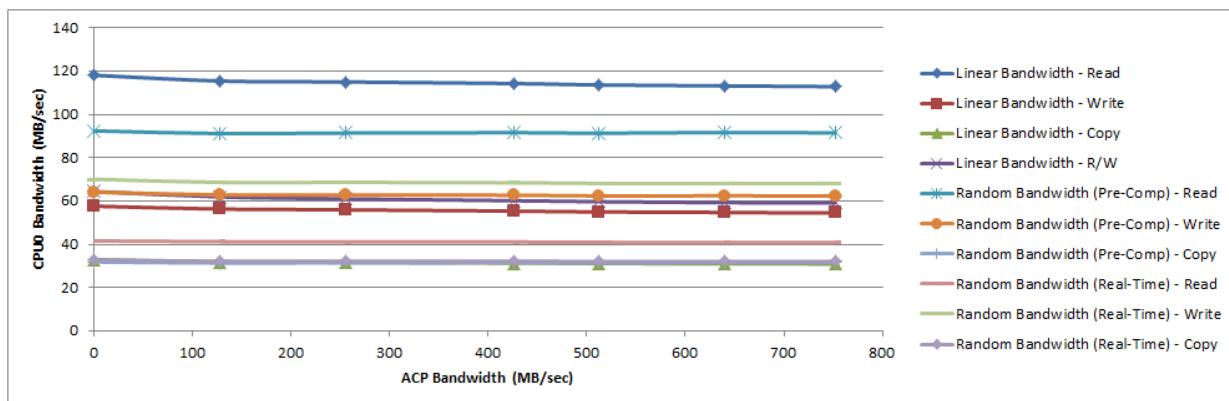


Figure 9-11: Performance Achieved by CPU0 vs. ACP Bandwidth (Set Size = 1024 KB)

As the requested ACP bandwidth was swept in value, the sustained CPU bandwidth was also recorded. [Figure 9-9, page 58](#) shows the bandwidth achieved by CPU0 with memory stride using a working set size of 4 KB. [Figure 9-10](#) shows the CPU0 bandwidth when memory stride operated on the 64 KB set size, while [Figure 9-11](#) is the bandwidth when memory stride used the 1024 KB set size. Due to their set sizes, these results are assumed to be good measurements of achieved bandwidth from the L1 data cache, the L2 data cache, and the DDR, respectively.

When running the memory stride software with a working set size of 4 KB, the CPU bandwidth was unaffected by the traffic on the ACP (see [Figure 9-9, page 58](#)). When using set sizes of 64 KB and 1024 KB, the ACP traffic does have a measurable impact on the CPU bandwidth (see [Figure 9-10](#) and [Figure 9-11](#)). The largest decrease in CPU0 bandwidth was 14.6% for the *Linear Bandwidth - Copy* benchmark operating on a set size of 64 KB.

Working with a Custom Target

To use the Xilinx® Software Development Kit (SDK) system performance analysis toolbox with your design, there are three important items to be aware of: instrumenting your software, instrumenting your hardware, and monitoring a live, custom target within SDK.

Instrumenting Software

Software run-times, bandwidth, and average latency can be calculated using straightforward instrumentation available in the standalone BSP defined in the *xtime_l.h* header. Before and after every benchmark, a call to *XTime_GetTime()* was inserted (see [Figure 10-1, page 61](#)). The difference between these two *XTime* values was converted to seconds (or msec) using the COUNTS_PER_SECOND value provided in *xtime_l.h*. Since the amount of data sent during any given benchmark is known, you can also calculate the achieved bandwidth and average latency during that particular benchmark.

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include "xtime_l.h" // XTime_GetTime()

// Macros
#define TIMEDIFF(t1,t2) (t2 - t1)
#define MILLISECONDS(t) (1000.0 * t / COUNTS_PER_SECOND)

// Globals
XTime start, end;

// Start a test
void startTest() {
    XTime_GetTime(&start);
}

// End a test
void endTest() {
    XTime_GetTime(&end);
    double time_curr = TIMEDIFF(start, end);
    double msec = MILLISECONDS(time_curr);
    printf("Run-time = %.2f msec...\n", msec);

    // Achieved Bandwidth = (total bytes transferred) / (msec)
    // Average Latency = (msec) / (total memory accesses)
}
```

```
// Dummy test
void runSomething(int time) {
    sleep(time);
}

int main(void) {
    startTest();
    runSomething(1);
    endTest();

    startTest();
    runSomething(2);
    endTest();

    return 0;
}
```

Figure 10-1: Sample Instrumented C/C++ Software Code

Instrumenting Hardware

If you would like to use the performance analysis features in SDK, you will need to include an AXI Performance Monitor (APM) in your programmable logic (PL) design. Note that an APM was already included in the System Performance Modeling (SPM) fixed bitstream design (see [SPM Software, page 15](#)).

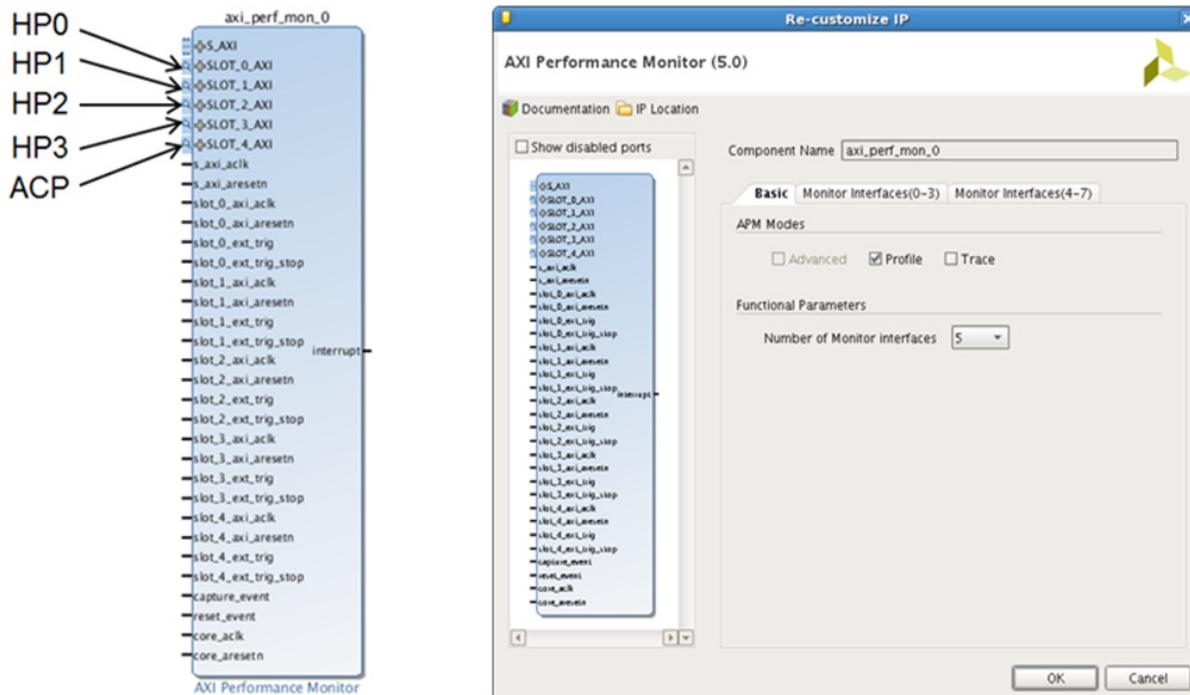


Figure 10-2: AXI Performance Monitor – Instance and Customization

Figure 10-2 shows how an instance of an APM appears in a Vivado® Design Suite IP integrator block diagram, as well as its configuration panel. To add this in Vivado IP integrator, perform the following steps to instrument your PL design with an APM:

1. In the IP integrator block diagram, click the **Add IP** button .
2. Search for “AXI Performance Monitor” and double-click it.
3. Double-click the APM to open the Re-customize IP dialog box. In this dialog box, set the following:
 - a. Check **Profile** under **APM Mode**.
 - b. Set the **Number of Monitor interfaces** to **5**.
4. Connect the APM to your design by doing the following:
 - a. Connect **S_AXI** to either GP0 or GP1 on the Zynq7 Processing System (via AXI interconnect).
 - b. Connect the **SLOT_x_AXI** ports (where x=0...4) to **HP0, HP1, HP2, HP3, and ACP**. If any of those interfaces are not used in your design, then leave the corresponding slot unconnected.
 - c. Connect **s_axi_aclk** and **s_axi_aresetn** to the clock and reset associated with GP0/GP1.
 - d. Connect all inputs **slot_x_axi_aclk** and **slot_x_axi_aresetn** to the appropriate clocks and resets.
 - e. Connect **core_aclk** and **core_aresetn** to the clock/reset with the highest clock frequency.
5. Select **File > Export > Export Hardware** to build the bitstream and export from Vivado.
6. Select **File > New > Project** and then select **Xilinx > Hardware Platform Specification** to import the generated file *<your design>.hdf* into SDK.

Monitoring a Custom Target

SDK provides the capability to monitor a running target, similar to what is described in [Chapter 11, End-To-End Performance Analysis](#). There are two techniques and both are applicable regardless of the target operating system. The difference is whether a Hardware Platform Specification is imported into SDK. This is a hardware design created by Vivado and then imported into SDK as a project.

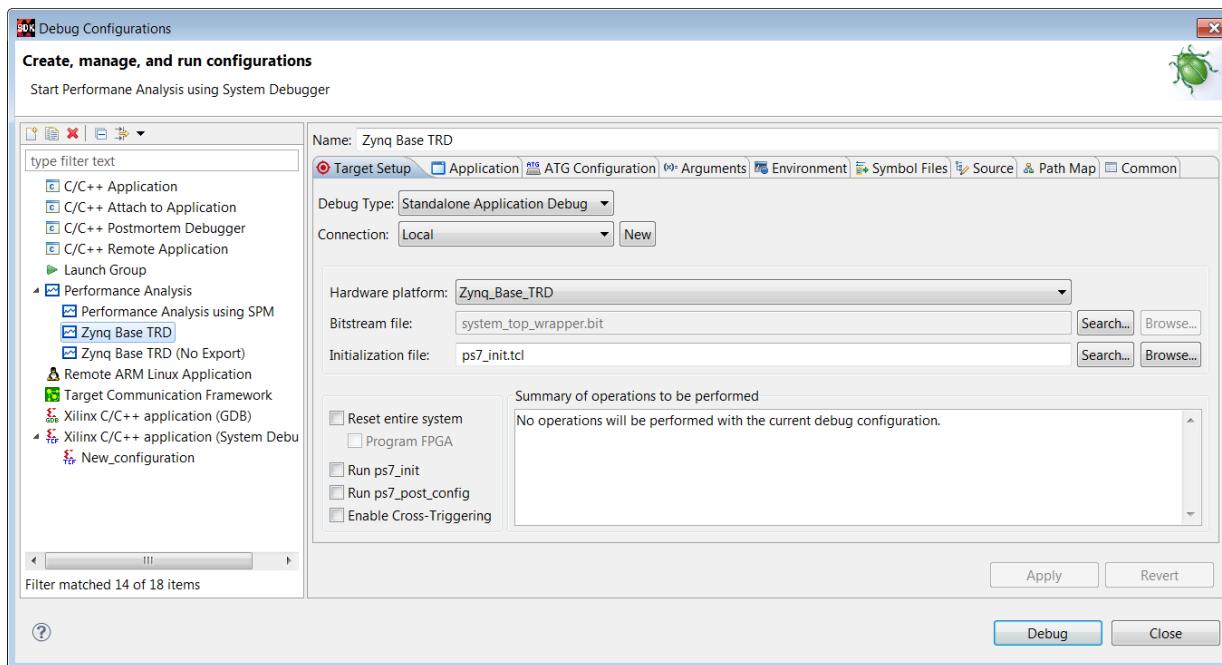


Figure 10-3: Target Setup in Debug Configuration (Using Imported Hardware Platform Specification)

If your design is defined in Vivado, then it is recommended to create a hardware platform specification based on that design. You can then do performance analysis based on that specification. The steps to use this path are as follows:

1. In Vivado, select **File > Export > Export Hardware** to build your bitstream and then export it. Select **File > New > Project** and then select **Xilinx > Hardware Platform Specification** to import the generated file `<your design>.hdf` into SDK.
2. Select **Run > Debug Configurations** to create a performance analysis configuration.
3. Do the following to configure your performance analysis:
 - a. Select **Performance Analysis** and then click **New**.
 - b. Set **Debug Type** to **Standalone Application Debug**.
 - c. Set **Hardware platform** to the hardware platform specification that you just imported. There are also options to Reset entire system, Program FPGA, Run `ps7_init`, Run `ps7_post_config`, and Enable Cross-Triggering.
 - d. Click **Debug** to open the Performance Analysis perspective.

Figure 10-3, page 63 shows the target setup used in [Chapter 11, End-To-End Performance Analysis](#) to connect to and monitor the Zynq®-7000 SoC Base Targeted Reference Design (TRD). Note that all check boxes are unchecked. Because the TRD uses an SD card boot, the system is already reset and configured.

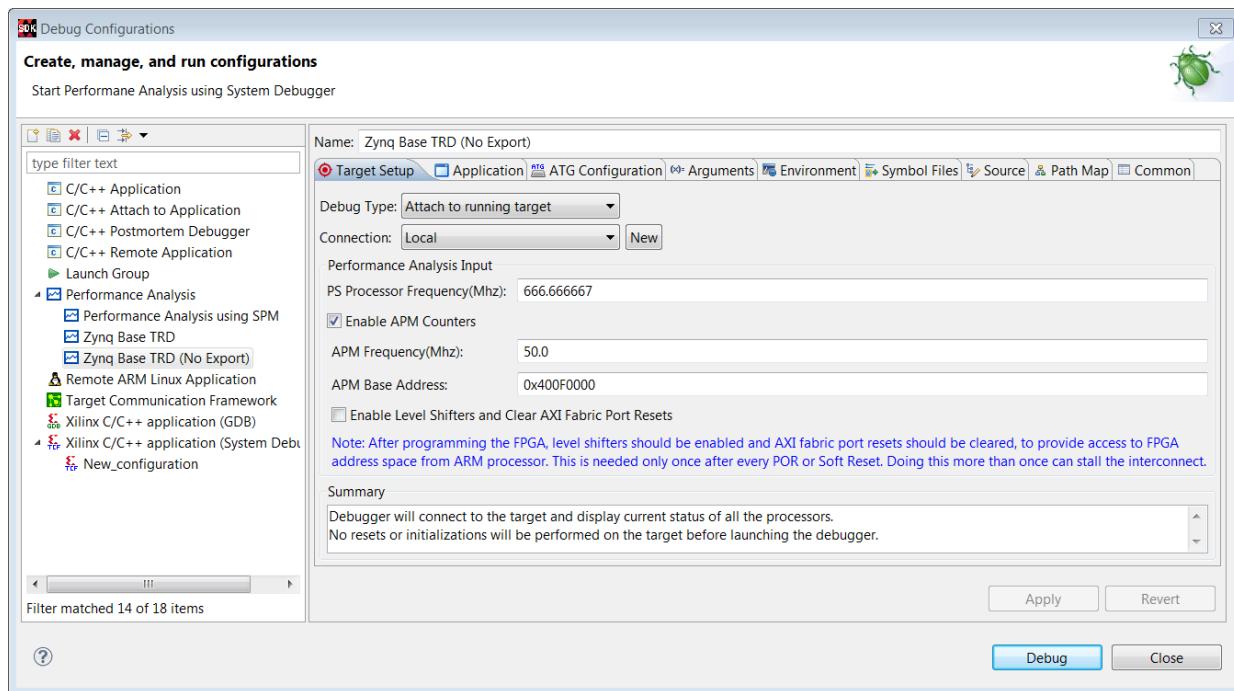


Figure 10-4: Target Setup in Debug Configuration (No Hardware Platform)

If for any reason you cannot create a hardware platform specification, you can still do performance analysis in SDK. The steps to use this path are as follows:

1. Select **Run > Debug Configurations** to create a performance analysis configuration.
2. Select **Performance Analysis** and then click **New**.
3. Set **Debug Type** to **Attach to running target**.
4. Set **PS Processor Frequency (MHz)** to the PS clock frequency used in your design.
5. If you have an APM in your design, do the following:
 - a. Select **Enable APM Counters**.
 - b. Set **APM Frequency (MHz)** to the frequency of the clock connected to **s_axi_aclk** on the APM core in your design.
 - c. Set **APM Base Address** to the base address of the APM in your design.
6. Click **Debug** to open the Performance Analysis perspective.

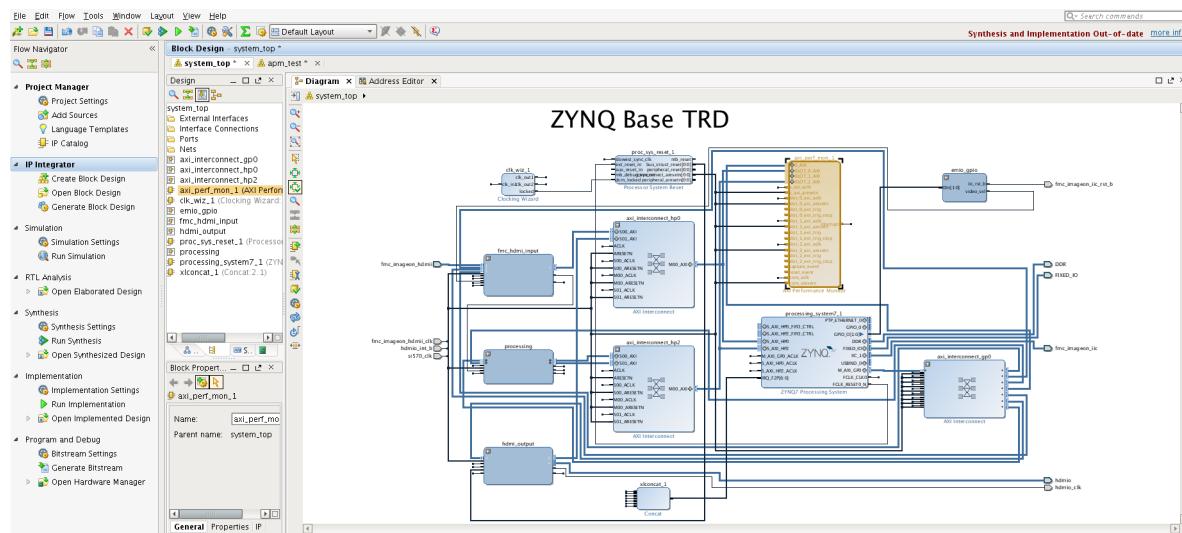
Figure 10-4, page 64 shows the target setup used in [Chapter 11, End-To-End Performance Analysis](#) to connect to and monitor the *Zynq-7000 SoC Base Targeted Reference Design (TRD)*, described in detail in the *Zynq-7000 SoC ZC702 Base Targeted Reference Design (UG925)* [Ref 3]. Note that the **APM Frequency** is 50 MHz while the **APM Base Address** is set to a hexadecimal value of 0x400F0000.

End-To-End Performance Analysis

SDK enables you to perform an end-to-end performance analysis flow (shown in [Figure 1-2, page 7](#)). This flow allows you to model the traffic of your estimated design and then validate the performance using your actual design. To better define this end-to-end flow, this chapter describes these steps for an example design. The design chosen is the Zynq®-7000 SoC Base Targeted Reference Design (TRD), described in detail in the *Zynq-7000 SoC ZC702 Base Targeted Reference Design* (UG925) [\[Ref 3\]](#).

Assess Requirements

[Figure 11-1](#) shows the block diagram of this TRD involving HDMI video input and output and processing using a Sobel filter. There are multiple video frames stored in DDR memory, creating multiple channels of high-bandwidth write and read traffic.



[Figure 11-1: Zynq-7000 SoC Base TRD Hardware Block Diagram](#)

Assume for now that this design is in the planning stages and not completed yet. How would you model this design using SPM? The first step is to assess the bandwidth requirements of the system. This involves calculating the expected bandwidth of all of the

communication channels in the design, and then deciding on the PS-PL interfaces to use. Since this is AXI traffic, the beats per transaction as well as data locality is also important.

To calculate bandwidth, you need to know the type of data that is being inputted, processed, and then output to a display. As shown in [Figure 11-1, page 65](#), the input data for the TRD is an HDMI source delivering 1080p/60 video with a data format of YCbCr 4:2:2, or 16 bits/pixel. This design also has the requirement of overlaying multiple layers of video on the output display to show the filtered video plus a GUI layer for controls and graphs. The GUI layer will be 1920 x 300 and in RGB 4:4:4 format, while all others are full frame in YCbCr 4:2:2 format. So while the clock frequencies may not be known yet, the system can still be modeled using the required bandwidths of the data traffic.

Table 11-1: Assessment of Bandwidth in Zynq-7000 SoC Base TRD

Description	Video Format	Port	Type	Bandwidth	DDR Bandwidth(%)
HDMI Video Input	YCbCr 4:2:2	HP0	Write	248.8	5.8
Processing Input	YCbCr 4:2:2	HP2	Read	248.8	5.8
Processing Output	YCbCr 4:2:2	HP2	Write	248.8	5.8
Multi-layer Video Output (Layer 1: 1920 x 300 GUI)	RGB 4:4:4	HP0	Read	138.3	3.3
Multi-layer Video Output (Layer 2: 1920 x 1080)	YCbCr 4:2:2	HP0	Read	248.8	5.8
Total Traffic	N/A	HP0/HP2	Write/Read	1133.4	26.5

[Table 11-1](#) shows a summary of the bandwidth assessment for the Zynq-7000 SoC Base TRD. The calculated bandwidth of each video channel is listed along with the percentage of DDR bandwidth. The design specification utilizes HP0 and HP2 as they use different ports on the DDR controller. Certainly, other connectivity could also be modeled.

Model Design

Now that the bandwidth requirements of the design have been identified, modeling can be performed using the SPM design. While not a necessity, a software application is

recommended to model Processing System (PS) traffic. The memory stride software application was used to model high activity in the PS. While this is obviously not the software that will be run in the actual design, it does provide a worst-case test scenario in terms of activity and memory bandwidth.

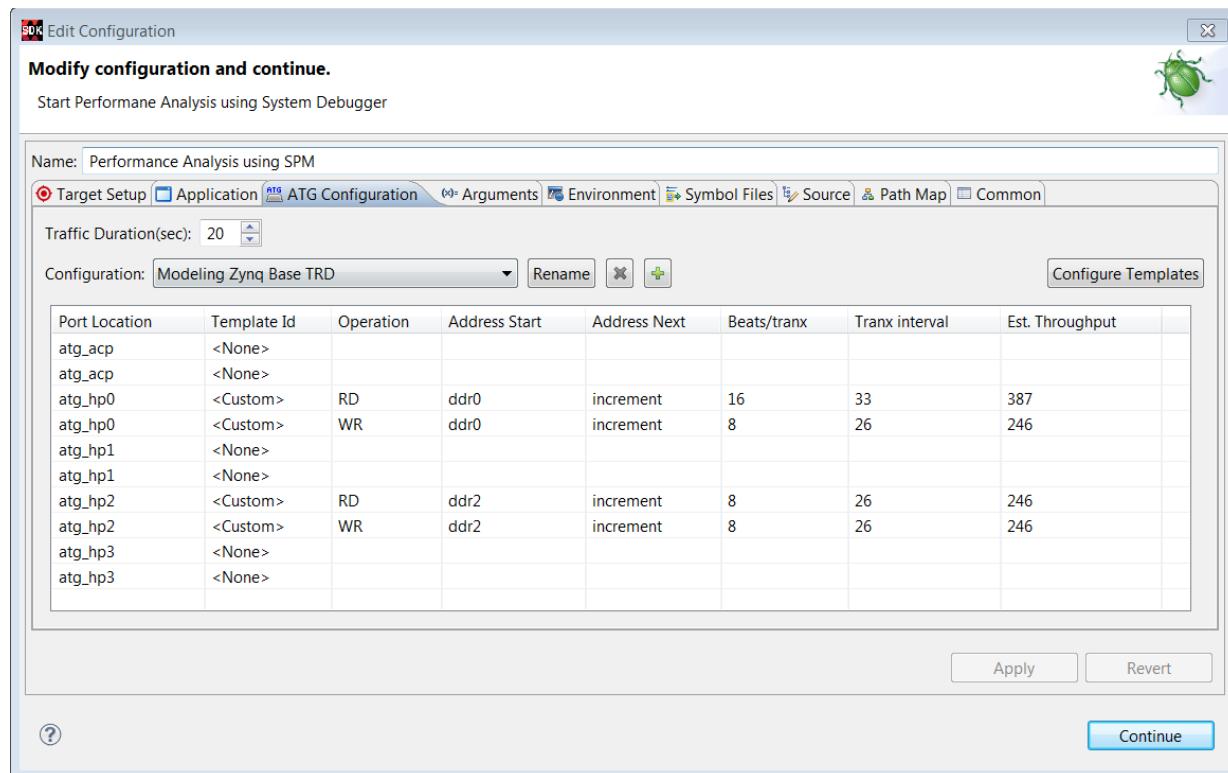


Figure 11-2: ATG Configuration to Model Traffic of Zynq-7000 SoC Base TRD

Figure 11-2 shows one way the traffic of the TRD could be modeled using System Performance Modeling (SPM). Assuming the video frames are stored in different parts of memory, the Address Start values were set to "ddr0" and "ddr2". Furthermore, video data is typically stored as successive lines of data, so the Address Next values were set to "increment". Beats/tranx values of 8 and 16 were chosen to model the video DMA to/from the Sobel filter and the multi-layer video output, respectively. The intervals were specified to match the desired throughputs for each port and type, as summarized in [Table 11-1, page 66](#).

PS Performance Graphs PS Performance Counters APM Performance Graphs APM Performance Counters MicroBlaze Performance Graphs MicroBlaze Performance Counters						
00:00:2.902 585 394-00:00:13.744 ...	HP0	HP1	HP2	HP3	ACP	GP
Write Transactions per ms	3841.5	0.00	3841.5	0.00	0.00	0.00
Minimum Write Latency	11.0	0.00	11.0	0.00	0.00	0.00
Maximum Write Latency	11.0	0.00	11.0	0.00	0.00	0.00
Average Write Latency	11.0	0.00	11.0	0.00	0.00	0.00
Write Throughput (MB/sec)	245.9	0.00	245.9	0.00	0.00	0.00
Read Transactions per ms	3027.4	0.00	3841.5	0.00	0.00	0.00
Minimum Read Latency	34.0	0.00	26.0	0.00	0.00	0.00
Maximum Read Latency	77.0	0.00	58.0	0.00	0.00	0.00
Average Read Latency	36.2	0.00	27.3	0.00	0.00	0.00
Read Throughput (MB/sec)	387.5	0.00	245.9	0.00	0.00	0.00

Figure 11-3: Summary of Performance Results for Model of Zynq-7000 SoC Base TRD

The ATGs were configured with the specified traffic and a run was performed on a ZC702 board. Figure 11-3 shows the summary table of results after this run was performed. Both HP0 and HP2 were capable of achieving the desired write and read bandwidth for this design. While the CPU performance results are for the memory stride benchmarks and not the eventual TRD application, it is still important to note the activity.



Figure 11-4: Write and Read Performance for Model of Zynq-7000 SoC Base TRD

Figure 11-4 shows the write and read performance graphs from the Xilinx® Software Development Kit (SDK), confirming the details of the achieved performance. The average write latency and write throughput remain consistent during the entire test. The average read latency is noticeably affected by the memory stride benchmarks. This is due to contention at the DDR controller. However, the read throughput remains consistent throughout the test.

As mentioned in [Chapter 1, Introduction](#), there are a number of reasons for performing this modeling, especially when the design is in its planning stages. Specific to this design, this includes the following:

- *De-risk for success* – The SPM model proved that the desired latencies and throughputs can be achieved. This provides a much stronger assurance that the Zynq-7000 SoC Base TRD will be able to achieve the same desired performance.
 - *Design Improvements* – [Figure 11-3, page 68](#) shows average read latencies of the model to be 40.13 and 32.41 for HP0 and HP2, respectively. For a video design such as this one with definitive frame rate constraints, a display controller could be designed to work within these metrics.
 - *What-If Scenarios* – The TRD could consider adding more video layers to the output display. Instead of iterating within the real system, it could be done using SPM. Other options to consider might include: adding a second video pipeline and upgrading to 4K video resolution. All of these scenarios could be modeled and analyzed before the design phase begins.
-

Performance Validation

The performance of the SPM model can be validated with the actual design. Analyze the performance of the Zynq-7000 SoC Base TRD using a properly instrumented design which include an APM (see [Instrumenting Hardware, page 61](#)). For the TRD delivered with 2015.1, this was already the case.

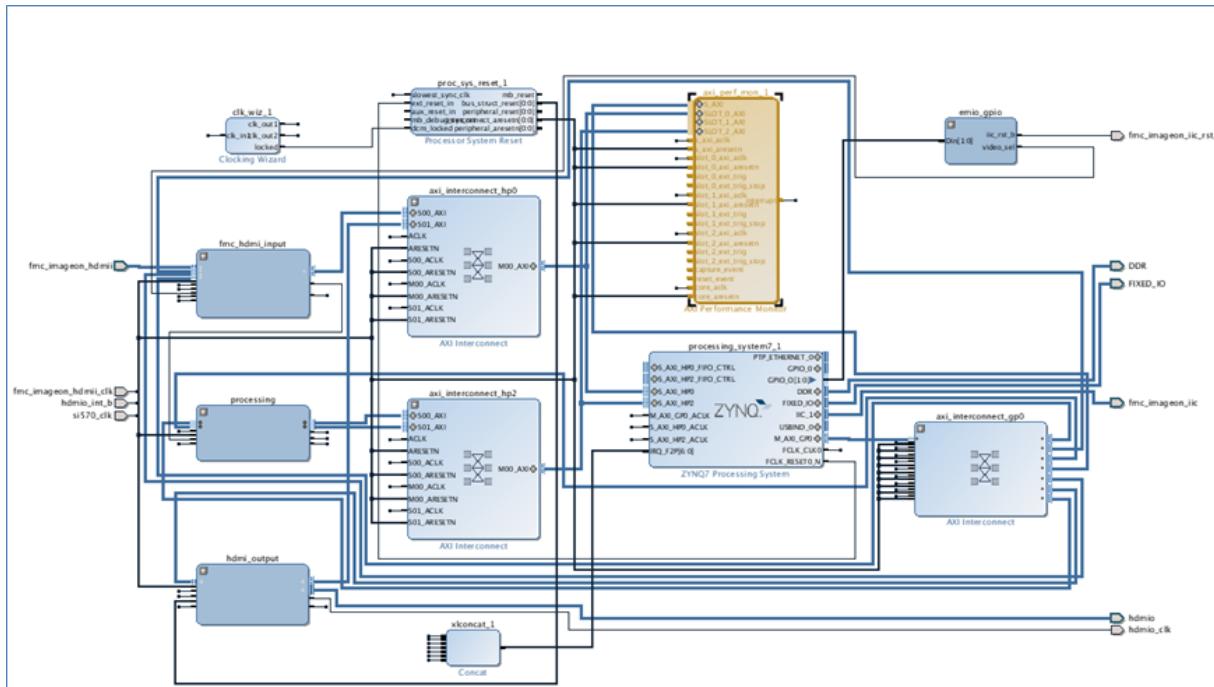


Figure 11-5: Vivado IP Integrator Block Diagram of Zynq-7000 SoC Base TRD

Figure 11-5 shows the TRD defined as a block diagram in Vivado® Design Suite IP integrator with the APM instance highlighted. The APM was properly configured in **Profile** mode to ensure that performance results would be reported correctly in SDK. HP0 was connected to slot 0 on the AXI Performance Monitor (APM) while HP2 was connected to slot 2. Slot 1 was left open (since HP1 was not used), and the **Number of Monitor interfaces** was set to **3**.

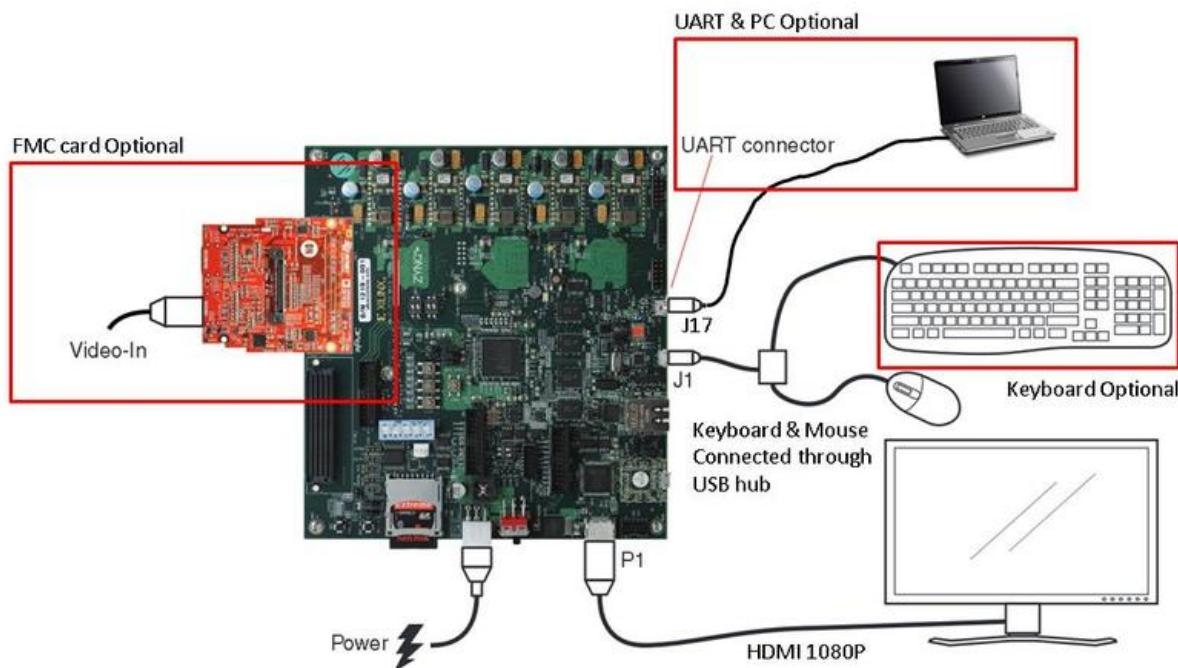


Figure 11-6: Connection Diagram of Zynq-7000 SoC Base TRD Using the ZC702 Board

Figure 11-6 shows the connection diagram to the ZC702 board in order to run the demonstration. The board was connected as shown, and the design was booted from the SD card.

The summary table of results is shown in Figure 11-7, page 71 while the corresponding graphs can be seen in Figure 11-8, page 72. The Programmable Logic (PL) performance results can be compared directly with those gathered from the SPM model in Figure 11-3, page 68. The write and read throughputs from the model are all within 2.0% of the actual results. The write latencies are within a cycle of the actual design, while the average read latencies of the model are within 16.0% of the actual design. While the actual design appears to be more bursty in nature than the SPM model, the average performance values were accurate enough to provide assurance that the TRD would meet its performance goals.

PS Performance Graphs						
PS Performance Counters						
APM Performance Graphs						
APM Performance Counters						
MicroBlaze Performance Graphs						
MicroBlaze Performance Counters						
00:00:02.902 585 394-00:00:13.744 ...	HP0	HP1	HP2	HP3	ACP	GP
Write Transactions per ms	3841.5	0.00	3841.5	0.00	0.00	0.00
Minimum Write Latency	11.0	0.00	11.0	0.00	0.00	0.00
Maximum Write Latency	11.0	0.00	11.0	0.00	0.00	0.00
Average Write Latency	11.0	0.00	11.0	0.00	0.00	0.00
Write Throughput (MB/sec)	245.9	0.00	245.9	0.00	0.00	0.00
Read Transactions per ms	3027.4	0.00	3841.5	0.00	0.00	0.00
Minimum Read Latency	34.0	0.00	26.0	0.00	0.00	0.00
Maximum Read Latency	77.0	0.00	58.0	0.00	0.00	0.00
Average Read Latency	36.2	0.00	27.3	0.00	0.00	0.00
Read Throughput (MB/sec)	387.5	0.00	245.9	0.00	0.00	0.00

Figure 11-7: Summary of Performance Results for Actual Zynq-7000 SoC Base TRD

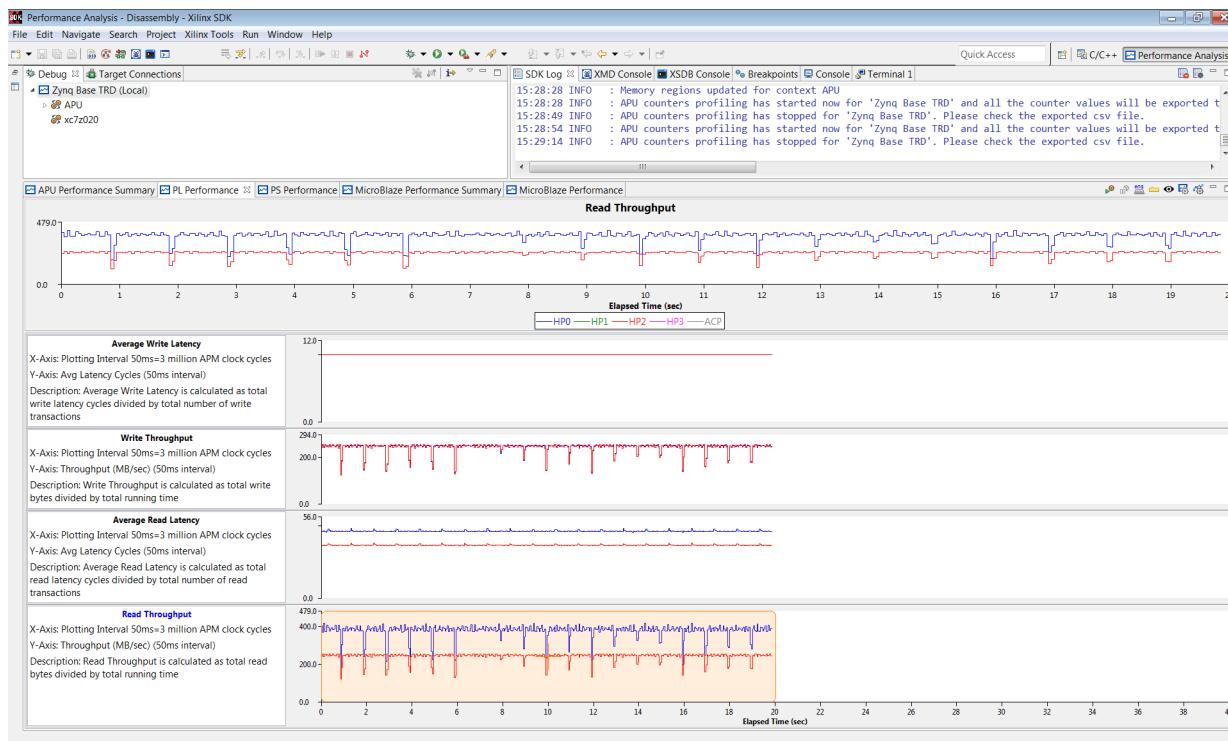


Figure 11-8: Write and Read Performance of Zynq-7000 SoC Base TRD

In-Depth Analysis

A second run of the TRD was performed to investigate the performance impacts of various run-time settings of the TRD. Table 11-2 lists all of these settings, as well as a summary of their corresponding performance results captured in SDK. The time window is approximately when those settings were used during the test run (see Figure 11-9, page 74, and Figure 11-10, page 74).

Table 11-2: Summary of Performance Metrics from Actual Zynq-7000 SoC Base TRD

Description	Time Window (sec.,approx.)	L1 Data Cache Accesses (x10 ³)		Write Throughputs (MB/sec)		Read Throughputs (MB/sec)	
		CPU0	CPU1	HPO	HP2	HPO	HP2
GUI only – minimized	0-10	100	100	0	0	37	0
GUI only – maximized	10-20	100	100	0	0	140	0
Video Enabled	20-30	100	100	246	0	382	0

Table 11-2: Summary of Performance Metrics from Actual Zynq-7000 SoC Base TRD

Description	Time Window (sec.,approx.)	L1 Data Cache Accesses (x10 ³)		Write Throughputs (MB/sec)		Read Throughputs (MB/sec)	
		CPU0	CPU1	HP0	HP2	HP0	HP2
Sobel filter on – SW mode	30-50	2800	7000	246	0	382	0
Sobel filter on – HW mode	50-70	100	200	246	246	382	246

As listed in [Table 11-2](#), the performance impacts can be directly measured in SDK. The GUI video output layer actually has two modes in the TRD: a minimized mode (size: 1920 x 80) and a maximized mode (size: 1920 x 300). Between 0-20 seconds, only that portion of the video frame was output by the hardware and sent to the video display, as confirmed by the read throughput results on HP0. After the output video was enabled, the total HP0 read bandwidth included the full video frame plus the GUI layer. The input 4:2:2 video was also written to DDR on HP0.

Turning on the Sobel filtering produced very noticeable changes in performance. There was clearly much higher CPU activity when the software (SW) mode was used, including notably L1 data cache accesses. Also, using the Sobel filter in hardware (HW) mode visibly affected the write and read traffic on HP2.

The performance results of the Zynq-7000 SoC Base TRD become especially evident in the graphs shown in [Figure 11-9, page 74](#), and [Figure 11-10, page 74](#). For the PS performance results in [Figure 11-9, page 74](#), the most activity occurs between approximately 30-50 seconds, when the Sobel SW filter mode is used. This period involves consistently high CPU utilization and notably higher values for IPC and L1 data cache accesses, among others.

For the PL performance results in [Figure 11-10, page 74](#), all time periods have very distinct characteristics. However, the most activity occurs between approximately 50-70 seconds, when the Sobel HW filter mode is used. The write throughputs on HP0 and HP2 and the read throughput on HP2 all correspond to expected values for 4:2:2 video streams. The read throughput on HP0 is higher in value as it includes the 4:2:2 full-frame video plus the partial-frame GUI layer.

In summary, instrumenting the Zynq-7000 SoC Base TRD provided two compelling benefits: 1) validating the performance results originally modeled by the SPM, and 2) providing a deeper analysis into various design settings.

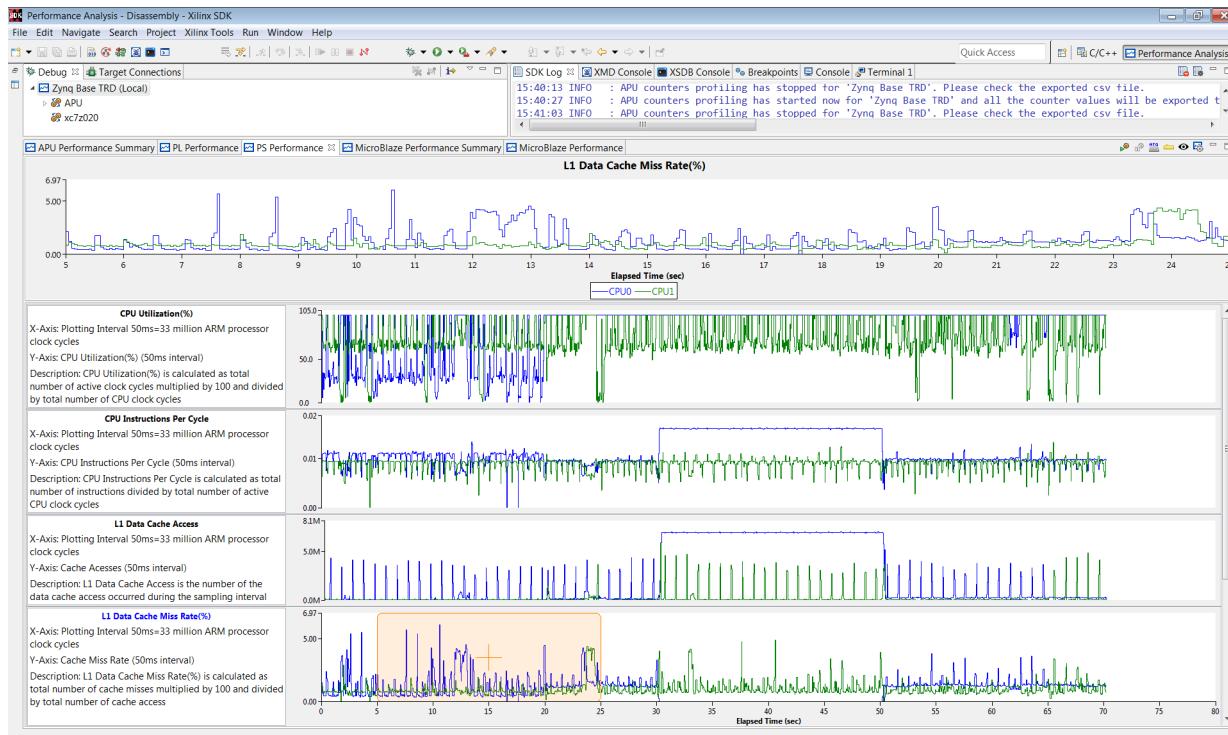


Figure 11-9: PS Performance Graphs of Zynq-7000 SoC Base TRD

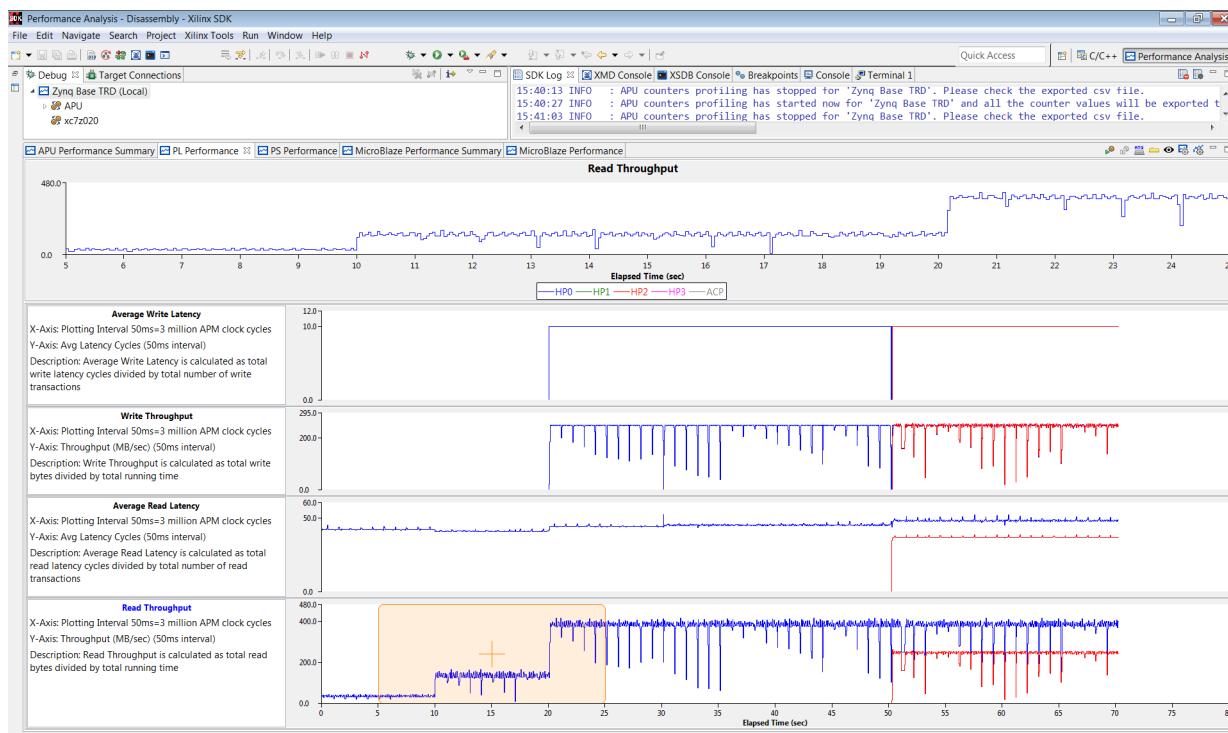


Figure 11-10: PL Performance Graphs of Zynq-7000 SoC Base TRD

Performance Checklist

The goal of this appendix is to provide you with a checklist of items to consider when evaluating the performance of your Zynq®-7000 SoC design. This should by no means be considered an exhaustive list, but instead a starting point for things to look out for and possible "gotchas" that are not easy to find. The Xilinx® Software Development Kit (SDK) can also be a helpful tool in evaluating your system performance, and some of its features and benefits are also highlighted here.

- **Use System Performance Modeling**

System Performance Modeling (SPM) is ideal for investigating system performance without any reliance on prior hardware design work or knowledge. The SPM can deliver assurance that your performance goals will be met, provide a better understanding of the achieved performance metrics of your design, and offer an environment to investigate what-if scenarios. Even if you are in the middle of creating or debugging your design, SPM can provide a low-overhead means of investigating design and architectural decisions. This guide covers a number of different examples that should help get you started.

- **Adjust clock frequencies in the SPM**

The SPM can enable investigation into adjusting the clock frequencies of the Processing System (PS), Programmable Logic (PL), and DDR. If you have high-bandwidth constraints for your system, the clock frequency would be an excellent way to increase throughputs. In an early evaluation environment, these frequencies could serve as design goals.

- **Set/Unset the L2 Cache Prefetch**

Check the instruction and data prefetch setting on the PL310 cache controller. Refer to Chapter 6: Evaluating Software Performance for information about how this is performed.

- **Compiler Optimization Settings**

Check the C/C++ build settings in SDK. For best performance, it is recommended to use either "Optimize More (O2)" or "Optimize Most (O3)".

- **Write Efficient Software Code**

This guide described how the performance of matrix multiplication was improved by almost 5x. An excellent resource on how to write high-performance, efficient software can be found in *What Every Programmer Should Know About Memory* by Ulrich Drepper (Ref [3]). Some options to consider as described in that document include:

- Bypassing the cache (for non-temporal writes)
 - Optimizing cache accesses and miss rates
 - Utilizing prefetching
 - Exploiting concurrency and multi-threading.
- **Understand Contention**

While running software-only tests can quite often produce predictable performance results, introducing traffic scenarios on either the HP ports or the ACP can lead to performance impacts that are difficult to predict. Introducing other high-bandwidth traffic into a system can inevitably lead to contention at the L2 cache, high-performance switches, or the DDR controller. Taking the time to understand the depth and impact of this contention can be very beneficial.

- **DDR Controller Settings**

These settings can either be modified in SDK (SPM design only) or in Vivado® Design Suite under the configuration of the Processing System 7 (PS7) IP block. Note that these settings can be used to re-allocate bandwidth from the DDR based on the desired needs of your system. Refer to Chapter 8: Evaluating DDR Controller Settings to see how this is done as well as how it might affect the DDR bandwidth of the CPUs and the HP ports.

- **Use the Zynq-7000 SoC On-Chip Memory**

The OCM is a 256 KB memory block accessible to both CPUs and the programmable logic from the Accelerator Coherency Port (ACP), General Purpose (GP), and High Performance (HP) ports. The OCM provides low-latency access and is an ideal component for use in synchronization or scratch pad applications.

- **Use the Accelerator Coherency Port**

The ACP is intended for cache coherency with the Arm Cortex-A9 processors. While this coherency can be very useful for some applications, it does create possible contention at the L2 Cache. If this contention becomes undesirable in your design, the OCM can be used in conjunction with the ACP to create a low-latency memory interface.

- **Use the High-Performance Ports**

The HP ports provide very high-throughput interfaces between the PS, PL, and DDR. The HP ports are recommended for any portion of a design where high bandwidth is required. While all HP ports are created equal, it is important to understand the pairing of HP ports used in the DDR controller. One DDR port is shared by HP0 and HP1, while another is shared by HP2 and HP3. These port pairings can be used together (if modifying DDRC settings) or separately (to maximize bandwidth across multiple DDRC ports).

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.

References

The references below provide additional information.

Xilinx User and Reference Guides

1. *Zynq-7000 SoC Technical Reference Manual* ([UG585](#))
2. *Vivado® Design Suite User Guide: Embedded Processor Hardware Design* ([UG898](#))
3. *Zynq-7000 SoC ZC702 Base Targeted Reference Design (ISE Design Suite 14.4) User Guide* ([UG925](#))
4. *Zynq UltraScale+ MPSoC Packaging and Pinouts Product Specification User Guide* ([UG1075](#))
5. *Zynq UltraScale+ MPSoC Technical Reference Manual* ([UG1085](#))
6. *Zynq UltraScale+ MPSoC Software Developer Guide* ([UG1137](#))
7. *Zynq UltraScale+ MPSoC Quick Emulator User Guide* ([UG1169](#))
8. Zynq UltraScale+ MPSoC OpenAMP Getting Started Guide ([UG1186](#))

Third-Party References

9. James Pallister, Simon Hollis, and Jeremy Bennett, "BEEBS: Open Benchmarks for Energy Measurements on Embedded Platforms", August 23, 2013. Available online: <http://dblp.uni-trier.de/db/journals/corr/corr1308.html#PallisterHB13>
10. David A. Patterson and John L. Hennessy, Computer Organization & Design: The Hardware/Software Interface, Morgan Kaufman Publishers, Inc., Second Edition, 1998.
11. Ulrich Drepper, "What Every Programmer Should Know About Memory", November 21, 2007. Available online: <http://www.cs.bgu.ac.il/~os142/wiki.files/drepper-2007.pdf>.

Training Resources

Xilinx provides a variety of training courses and QuickTake videos to help you learn more about the concepts presented in this document. Use these links to explore related training resources:

1. [Zynq-7000 SoC: Development Tools Overview](#)
 2. [Zynq-7000 SoC: System Performance Tools Overview](#)
 3. [Zynq-7000 SoC: Hello World in 5 Minutes](#)
 4. [Zynq-7000 SoC: Bare Metal Application Development](#)
-

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>.

© Copyright 2015-2018 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.