

# System Performance Analysis of an All Programmable SoC

Author: Forrest Pickett

## Summary

This application note educates users on the evaluation, measurement, and modeling of the performance of the Zynq-7000 All Programmable (AP) SoC. The application note introduces and explains an example design that shows the different aspects of the system performance of Zynq-7000 devices. The design highlights the communication between the programmable logic (PL) and the processing system (PS) in the Zynq-7000 AP SoC meets your system performance needs.

The design that is provided with this application note extends the System Performance Modeling (SPM) project that is provided in SDK and documented in *SDK User Guide: System Performance Analysis* (UG1145) [Ref 1]. An SPM project is executed in actual target hardware and includes a fixed bitstream containing five AXI traffic generators. These traffic generators are configurable cores used by SDK to model programmable logic (PL) traffic activity. Software applications can also be run simultaneously in the processing system (PS), and you can specify system configuration parameters. After a user has run an SPM project and would like to extend it to better reflect their application, the design in this application note can be used.

## Introduction

There are three major sections of this document.

- The first section describes the system performance of a Zynq-7000 AP SoC.
- The second section shows you how to:
  - Implement a system model of a design using a PL implementation that emulates an application.
  - Use the Software Development Kit (SDK) to show system performance metrics using the model.
- The third section shows you how to:
  - Add your own software to run the model.
  - Replace items in the PL with your own IP.



#### Hardware and Tool Requirements

The hardware requirements for the reference systems are:

- ZC702 evaluation board
- AC power adaptor (12 VDC)
- USB Type-A to USB Mini-B cable (for UART communication)
- USB Type-A to USB Micro-B cable (for JTAG access)
- Vivado® Design Suite 2014.3

#### **AP SoC System Performance**

The performance of an SoC is measured at the device or system level, not at the interface or circuit block level. A primary component that determines performance is how effectively the system architecture handles shared resource contention. The connected blocks typically include the application processor unit (APU), interconnect, I/O peripherals, I/O, and memory controllers (static and dynamic) (Figure 1). Another important factor is the speed at which these blocks operate and the traffic they generate. These blocks can be grouped into subsystems such as the application processor unit, I/O subsystem, and memory subsystem. The memory subsystem, especially on an AP SoC, is the most critical shared resource subsystem that determines the SoC performance.





Figure 1: Example SoC Block Diagram

Some of the factors affecting system performance are:

- Cache sizes
- Memory speed
- Memory efficiency
- Memory data width
- CPU speed
- Interconnect
- Integration of the blocks

Some of these factors are constant and cannot be changed. Others are part dependent, some are platform dependent, and others are design dependent.

It is important to define how *efficiency* and *effectiveness* are used in this document:

**Efficiency** refers to how efficiently the memory controller uses the DRAM. An efficient memory controller uses DRAM intelligently to get more bandwidth. The bandwidth is measured in MB/s and efficiency is measured as a percentage.

**Effectiveness** refers to how effectively the DRAM controller distributes the memory bandwidth. Or, another way to look at this is the "fairness" of the DDR controller giving bandwidth to the different ports.



In other words, the efficiency is how much bandwidth the controller gets from the theoretical maximum, and the effectiveness is how that bandwidth is distributed to the contending blocks. Figure 2 shows the most important factors to consider for system performance.



Figure 2: Most Important Factors in System Performance

One of the important things an SoC needs to do is to keep the CPU running with a minimal number of data-dependent stalls. To achieve this, data used by the CPU need to be loaded to the caches, which has much lower access latency than the main memory. The speed at which data can be loaded into the cache depends on the performance of the memory subsystem. For example, if the bandwidth of the memory subsystem is deficient, caches are not filled on time and the CPU stalls waiting for the data. The performance of a memory subsystem mainly depends on:

- Speed (frequency) of the memory
- Efficiency of the DRAM controller
- Effectiveness of the DRAM controller
- Width of the memory interface (both internal and external)
  - Ports to the DDR controller (internal)
  - Interfaces to the DRAM (through the PHY external)

*Note:* The memory subsystem does not include the caches.

If the memory subsystem in an SoC does not have enough bandwidth, the performance of the whole system is negatively affected. For example, if the CPU is operating at a high frequency and is trying to fetch more data from the memory than the level-2 cache is able to deliver in a



timely manner, the CPU has to stall until the data is received. In this kind of system, having a CPU with a higher clock speed alone does not increase the performance of the system because the memory subsystem is the bottleneck. Figure 3 shows the other factors in system performance.



x1219\_03\_100314

*Figure 3:* Components for AP SoC System Performance

Typically, an SoC is a heterogeneous system and contains blocks that generate different types of traffic. All these blocks compete for the same memory resources. The memory subsystem must be intelligent enough to resolve contention of resources between different circuit blocks and handle different types of traffic efficiently and fairly. Therefore, comparing CPU clock rates without taking into account other elements in the SoC (specifically the memory subsystem and the programmable logic) is of limited value and could potentially paint a very misleading picture. Some of the different types of traffic in Zynq-7000 AP SoCs are between the following masters and the DDR:

- Traffic from the APU (memory load and store instructions from the CPU and direct memory access (DMA) requests from the central DMA)
- PS peripherals (Ethernet, USB, etc.)
- DMA requests originated from the PL
- PL peripherals (IP and accelerators). For example, the video pipeline IP.



The memory subsystem must arbitrate access requests from these different traffic sources while maximizing bandwidth utilization of the DRAM. An effective memory controller exhibits balanced and predictable behaviors. Fairness of the memory controller is modulated by the quality of service (QoS) mechanism, which optimizes total throughput while avoiding starvation of any individual traffic. An unfair implementation of a memory subsystem would be based on an absolute arbitration priority. This type of arbitration is inappropriate in an SoC where multiple data streams from different sources need to be supported at the same time and starvation at any source almost always slows down the overall system.

## An Instrumented Design for Performance Analysis

To model a system that uses the Zynq-7000 AP SoC, you need a design that can emulate the data traffic from the PL to the PS. This is done using Xilinx IP that can be configured to mimic different parts of a user design. See Reference Design to download the design files.

- 1. Unzip the files into a directory called XAPP1219 on your C drive.
- 2. Start Vivado Design Suite 2014.3.
  - a. Open the Tcl window.
  - b. Change directories (cd) to the directory where you placed the design files.
  - c. Source the project.tcl file.
- 3. View the block diagram in Vivado IP Integrator:
  - a. Select IP Integrator > Open Block Design.
- 4. Click **Generate Bitstream**. There is one supplied just in case. If you are using this bitstream, start at step 6.
- 5. Click **Export Hardware** and check the **Include bitstream** box.
- 6. Launch SDK command from the Vivado Design Suite.

There is also a workspace with all the items included. If you want to just exit the Vivado Design Suite and start SDK, choose the workspace provided.

The design generates traffic from the PL to the PS DDR to emulate an application. To do this, the program uses an IP core called the AXI Traffic Generator (ATG), which is documented fully in the *LogiCORE™ IP AXI Traffic Generator v2.0 Product Guide* (PG125) [Ref 2]. The ATG can mimic standard interfaces like Ethernet, USB, and PCIe® or be configured in a custom manner to better match a customer application. The reference design with this application note contains six ATGs that are connected to the four high-performance (HP) ports, the accelerator coherency port (ACP), and one general purpose (GP) port of the Zynq-7000 SoC. These are pre-configured, but you can change them for the system you would like to model. The procedure to change the design (specifically the ATGs) to model an application is described in System Performance Analysis. The block diagram of the design (in the Vivado Design Suite) is shown in Figure 4.





Figure 4: System Performance Analysis Block Diagram



The ATGs are in a hierarchical block to make the top-level diagram cleaner. The TrafficGen block was created to have the ATG, a constant block, and an AXI interconnect block to make one block at the top level instead of three. The block diagram in Figure 5 shows what a TrafficGen block looks like inside the hierarchy. The constant block starts the ATG running and the interconnect changes the ATG AXI4 to the Zynq-7000 AP SoC AXI3 interface.



Figure 5: TrafficGen Block Diagram

All six ATGs are configured in the same manner, having a high-level traffic profile and then a traffic profile of data. The settings in the data flow have the percentage of reads and writes each at 50%, and an interval fixed at 50 cycles. The 50 cycles are measured from the start of one transaction to the start of the next transaction. (The start of the transaction is the presentation of the address on the AXI interconnect block.) Each ATG runs off an FCLK from the PS and is configured for 100 MHz. This generates 256 MB/s of reads and writes on each of the interfaces for the HP and ACP ports. The GP port has a 32-bit data width, so the traffic that is generated is half of the other ATGs in the design. Therefore, the GP port has 128 MB/s of reads and writes. The configuration of each ATG is shown in Figure 6.



Re-customize IP	And Total Strike Strike Land State	a direct	X				
AXI Traffic Generator (2.0)			A				
💕 Documentation 🛅 IP Location			2. 				
Show disabled ports	Component Name perftest_axi_traffic_gen_0_0						
	Profile Selection O Custom  High Level Traffic						
	High Level Traffic Traffic Profile ① Yideo ② PCIe ② Ethernet ③ US8 ④ D	ata					
	AXI Options AXI Master Width 64 * AXI Base Address 0x20000000	AXI Options AXI Master Width 64 * AXI Base Address 0x2000000					
-s_axi_aclk -s_axi_aresetn M_AXI⊕	Traffic Gen      @ Repetitive One-Shot						
-core_ext_start irq_out -	DATA Mode						
-core_ext_stop	Pattern Options	Transaction Type					
	Traffic Pattern Fixed +	Transaction Type Read Write *					
	Transfer Length (Min)	Read Share (%) 50	[0100]				
	Transfer Length (Max) 256 * Transfer Length (Avg) 16 *	Write Share (%) 50	[0100]				
	Inter Transaction Gap	Transaction Addressing					
	ITG Type Fixed *	Addressing Fixed -					
	ITG (Clocks) 50 [01023]	Transaction Seed 1	[1100]				
۲							
			OK Cancel				

x1219\_06\_112514

Figure 6: AXI Traffic Generator Configuration



The calculation to get the rate in MB/s is shown in Equation 1 and the numbers from the ATG configuration are in Equation 2.

$$\frac{3 \times N}{\frac{1}{f} \times I} = Rate$$
 Equation 1

where

B = Bytes in a word

N = Number of words in TX

I = Interval

f = PL clock frequency (in MHz)

Or

$$\frac{8 \times 16}{\left(\frac{1}{100}\right) \times 50} = 256$$
 Equation 2

When the ATG is programmed to have 50% reads and 50% writes, there will be 256 MB/s on the write channel of the AXI and 256 MB/s on the read channel. The ATG issues the read and write transactions simultaneously, so the AXI interface has a total of 512 MB/s of traffic. Therefore, for this example there are 512 MB/s on each HP port and the ACP. Table 1 shows the traffic on each port and the total MB/s being transferred from the PS to the PL. The maximum bandwidth for the PS DDR is included and is based on a 32b DDR running at 533 MHz speed.

	HP0	HP1	HP2	HP3	ACP	GP1
read	256	256	256	256	256	128
write	256	256	256	256	256	128
Total for port	512	512	512	512	512	256
Total	2816					
Maximum PS DDR bandwidth	4264					
% of maximum PS DDR bandwidth	66.04%					

Table 1: HP & ACP Port Bandwidth of PS DDR

Table 1 shows that the amount of traffic from the PL is about 60% of the maximum of the PS DDR. The interfaces from the PL to the PS can generate more traffic than the DDR controller can handle. When configuring the ATG traffic, you need to be aware of a possible overload of the PS DDR controller. In other words, the ATGs can never move more traffic than the maximum bandwidth of the PS DDR, no matter how they are configured. Creating a simple table like this is good practice to see if the PL is going to be more than 80% of the theoretical maximum of the PS DDR. If so, there could be an issue with performance not meeting expectations, and/or the latency could be high.

You now have a design that can model a system. How do you know if your performance metrics are met? Is there actually 256 MB/s of traffic on the HP and ACP ports? To get this data from the design, another Xilinx IP needs to be used. The AXI Performance Monitor (APM) is used to collect the data from each of the AXI interfaces that are being modeled, or the HP, ACP, and GP



ports. The configuration of the APM is shown in Figure 7.

	1					
ow disabled ports	Component Name perftest_axi_	perf_mon_(	)_0			
-je,ax	Basic Monitor Interfa	ces(0-3)	Monit	or Interfaces(4-7)		
-jerolitan -jerolitan	Slot 0			Slot 1		
s destructions s des	AXI Protocol	AXI3	-	AXI Protocol	AXI3	*
w deuorjajaw ⇔jauanata	Enable Synchronizer (Auto)	false	-	Enable Synchronizer (Auto)	false	Ŧ
- Journal - Mailaniah	ID Width (Auto)	1	-	ID Width (Auto)	1	Ŧ
-skilleriersen -skillerierier -skillerierier	Data Width (Auto)	64	-	Data Width (Auto)	64	Ŧ
−ska, jan,anda −ska, jan,ansata	Address Width (Auto)	32	-	Address Width (Auto)	32	Ŧ
−րեվ, լովկոց −րեվ, լովկոց, լոր −րեվ, Հուլան,	Slot 2			Slot 3		
=slatterterster niengi =slatterting	AXI Protocol	AXI3	+	AXI Protocol	AXI3	*
-stationed -stationed	Enable Synchronizer (Auto)	false	+	Enable Synchronizer (Auto)	false	*
-stationing -stationingstop	ID Width (Auto)	1	Ŧ	ID Width (Auto)	1	*
nia (uni adi nia (uni adian nia 4 ni lun	Data Width (Auto)	64	-	Data Width (Auto)	64	Ŧ
−slat,4,mt,brashap −slat,5,mt,aak	Address Width (Auto)	32	-	Address Width (Auto)	32	+
- bit to bit						

x1219\_07\_112514

Figure 7: AXI Performance Monitor Configuration

The APM is in the Profile mode, supported in the SDK 2014.1 and later. There are other modes that are documented in the *LogiCORE IP AXI Performance Monitor v5.0 Product Guide* (PG037) [Ref 3].

The Profile mode shows bandwidth and latency at a high level. SDK reads the APM every 50 ms for the data on each slot or interface. The other modes are for more detailed gathering of data at the interface. The Advanced and Trace modes take data at a higher rate than the Profile mode. These modes are not supported by SDK and are not part of this application note.

The configuration for the APM is shown in Figure 7. Six interfaces are selected. Each of the monitor interfaces are set up the same way for AXI3 (Zynq-7000 AP SoC) ports and for 64 bits of data. The only exception is the GP port, which has 32 bits of data width. The first four monitor interface configurations are shown in Figure 7 (these are the HP ports, the fifth is the ACP, and the sixth is the GP). The first five ports of the APM must be connected this way (HP0 to slot 0, HP1 to Slot 1, HP2 to Slot 2, HP3 to Slot 3, and ACP to Slot 4) because this is what SDK is



designed for. If you have a customer design, set it up the same way otherwise the results are incorrect.

The settings of the Zynq-7000 AP SoC AXI ports are as shown in Figure 8. This is configured to connect the traffic generators to the different interfaces in the Zynq-7000 AP SoC.

ZYNQ7 Processing Syst	tem (5.4)			
🍘 Documentation 🚳 Presets 📄	) IP Location 🍈 Import XPS Settings			
Page Navigator « PS	S-PL Configuration			Summary Report
Zynq Block Design 🛛 🔺	Search: Q-			
PS-PL Configuration	I Name	Select	Description	
Peripheral I/O Pins	• General	50000	bosciption	
	DMA Controller			
MIO Configuration	GP Master AXI Interface			
Clock Configuration	M AXI GP0 interface		Enables General purpose AXI master interface 0	
	● M AXI GP1 interface		Enables General purpose AXI master interface 1	
DDR Configuration	GP Slave AXI Interface			
SMC Timing Calculation	S AXI GPO interface	<b>V</b>	Enables General purpose 32-bit AXI Slave interface 0	
	S AXI GP1 interface		Enables General purpose 32-bit AXI Slave interface 1	
Interrupts	HP Slave AXI Interface			
	G AXI HPO interface		Enables AXI high performance slave interface 0	
	● S AXI HP1 interface		Enables AXI high performance slave interface 1	
	G AXI HP2 interface		Enables AXI high performance slave interface 2	
			Enables AXI high performance slave interface 3	
	ACP Slave AXI Interface			
		<b>V</b>	Enables AXI coherent 64-bit slave interface	
	Tie off AxUSER		Tie off AxUSER signals to always enable coherency	
	PS-PL Cross Trigger interface		Enables PL cross trigger signals to PS and vice-versa	
	2006 2004			

x1219\_08\_120114

Figure 8: Zynq AXI Port Settings

The design is set up to model and collect system performance data or what is called *instrumented for system performance*.

You can create a bitstream by clicking **Generate Bitstream** in the Flow Navigator pane on the right-hand side of the Vivado window. If you do not want to perform this step, you can use the bitstream in the workspace that is included in the ZIP file. The steps on how to use the bitstream are in the next section, so keep the Vivado IDE running.

There is another portion of system performance that was discussed in the Introduction. How do you know what the CPUs are doing while the hardware is being modeled in the PL? For this, the Zynq-7000 architecture has performance monitoring units (PMUs) in the PS to collect data



about the ARM Cortex-A9 while it is running. There are also performance counters in the L2-Cache that are collected and shown to you. The metrics that are gathered are:

- CPU level 1 cache miss rate
- CPU level 1 cache access
- CPU stall cycles per read instruction
- CPU stall cycles per write instruction

With the metrics about the software running on the CPUs and the hardware being modeled, you can analyze system performance before starting a design.

#### **System Performance Analysis**

The next step is to run the model and show what is occurring at various analysis points. To view the performance metrics, the design needs to be run in SDK. The first run is done without software. This shows the performance of the HP and GP ports to the DDR through the PS memory controller alone.

If you selected not to generate the bitstream, you can start SDK and select the workspace when SDK starts using the directory called Workspace, which was part of the ZIP file. It should look like Figure 11, so skip down to step 4, page 16.



To set up for system analysis:

 From the File menu in the Vivado IDE, first export the hardware for SDK by selecting File > Export > Export Hardware, then start SDK using the exported hardware definition by selecting File > Launch SDK. In the SDK window, shown in Figure 9, select File > Import > General > Existing Projects into Workspace and click Next.

C/C++ - perftest_wrapper_hw_platform_0/system.hdf	- Xilinx SDK		- 10	- • • • · · · ·
File Edit Source Refactor Navigate Search Pro	piect XilinxTools Run Window Help			
	🗖 📾 🛪 🖄 🕇 🗑 🛪 🖄 🛪 🖨 🗛 🛪 🗐 🖉	∎ \$1 <b>-</b> 51 <b>-</b> 15 6 -	- ⇔ - [ ल*	
			Qui	ck Access
ြို့ Project Explorer 🛛 📄 🔄 🔻 🍸 🖳 🗖	🛅 system.hdf 🛙			📲 Outline 🛛 💿 Make Tar 🗖 🗖
Image: perftest_wrapper_hw_platform_0           Image: perftest_hd_tcl	perftest_wrapper_hw_platform_0 Hardware Platform Spec	ification	^	An establica is makes with bla
perftest_butch	Design Information			An outline is not available.
e ps7_init_gpl.c			=	
c ps/_init_gpl.h	Target FPGA Device: /z020			
c ps/_init.c	Created With: Vivado 2015.3			
c ps/_init.h	Created On: Thu Sep 24 08:58:11 2015			
s7_init.ntmi	Address Map for processor ps7_cortexa9_0			
in system.hdf	ns7 intr dist 0 0xf8f01000 0xf8f01fff			
	ps7 scutimer 0 0xf8f00600 0xf8f0061f			
	ps7 slcr 0 0xf8000000 0xf8000fff			
	ps7 scuwdt 0 0xf8f00620 0xf8f006ff			
	ps7 l2cachec 0 0xf8f02000 0xf8f02fff			
	ps7 scuc 0 Oxf8f00000 Oxf8f000fc			
	ps7 gspi linear 0 0xfc000000 0xfcffffff			
1	axi_perf_mon_0_0x43c00000_0x43c0ffff			
	ps7 pmu 0 0xf8893000 0xf8893fff			
	ps7_afi_1 0xf8009000 0xf8009fff			
	ps7_afi_0 0xf8008000 0xf8008fff			
	ps7_qspi_0 0xe000d000 0xe000dfff			
	ps7_afi_3 0xf800b000 0xf800bfff			
	ps7_afi_2 0xf800a000 0xf800afff			
	ps7_globaltimer_0 0xf8f00200 0xf8f002ff			
	ps7_dma_s 0xf8003000 0xf8003fff			
	ps7_iop_bus_config_0 0xe0200000 0xe0200fff			
	ps7_xadc_0 0xf8007100 0xf8007120			
	ps7_ddr_0 0x00100000 0x3fffffff		-	
	Overview			
👛 Target Connections 🛛 🛛 📽 📟 🗖	🖹 Problems 🛛 🖉 Tasks 📃 Console 🔲 Properties 🖳 SDK Termi 📄 🗖	SDK Log 🛛		
🛛 🕞 Hardware Server	~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~	14:49:32 INFO :	Launching XSE	DB server: xsdb.bat C:/Xilinx/S
🕟 🗁 Linux TCF Agent	0 items	14:49:34 INFO :	: XSDB server h	has started successfully.
🕨 🔈 🗁 QEMU TcfGdbClient	Description Resource Path	14:49:34 INFO :	Processing co	mmand line option -hwspec C:/>
		-		
		-		
				-
				-
				x1219 09 101615

Figure 9: SDK Imported Hardware Screen



2. Select the root directory provided in the ZIP file as the Workspace. There should be two items selected, **perfmon** and **standalone\_bsp\_0**, as shown in Figure 10.

Import	Sec. Sec.				
nport Projects Some projects cannot workspace	: be imported b	ecause they alre	ady exist in th	ne	
Select root directory:	C:\XAPP1219	\Workspace		•	Browse
) Select archive file:				-	Browse
'rojects:					
perfmon (C:\XAF	P1219\Worksp	ace\perfmon)			Select All
perttest_wrapper     RemoteSystems7	_nw_platform_l FempFiles (C:\X	(C:DAPP1219 APP1219\Work:	\workspace\\ space\Remote	eSyste	Deselect All
🔲 RemoteSystems7	FempFiles (C:\X	APP1219\Work:	space\Worksp	Dace\	Refresh
v standalone_bsp_	U (C:\XAPP1219	I\Workspace\W	orkspace\star	ndalo	
✓ standalone_bsp_	U (C:\XAPP1219	I\Workspace\W	orkspace\star	ndalo •	
standalone_bsp_     standalone_bsp_      options     Search for nested pro     Copy projects into w	u (C:\XAPP1219 m ojects vorkspace	I\Workspace\W	orkspace\star	hdalo ⊂	
standalone_bsp_      options     Search for nested pri     Copy projects into w      Working sets	u (C:\XAPP1219 m ojects vorkspace	I\Workspace\W	orkspace\star	hdalo ∟	
standalone_bsp_      options     Search for nested pro     Copy projects into w      Working sets     Add project to work	u (C:\XAPP1219 m ojects vorkspace	I\Workspace\W	orkspace\star	hdalo ⊑	
standalone_bsp_      options     Search for nested pro     Copy projects into w      Working sets     Add project to work      Working sets:	U (C:\XAPP1219 Djects vorkspace cing sets	I\Workspace\W	orkspace\star	- t	Select
standalone_bsp_      options     Search for nested pro     Copy projects into w      Working sets     Add project to work      Working sets:	U (C:\XAPP1219 Djects vorkspace cing sets	Next >	orkspace\star		Select

Figure 10: SDK Import Projects into Workspace



3. Click **Finish**. The project in SDK should look like Figure 11.

sox C/C++ - perftest_wrapper_hw_platform_0/system.hdf	- Xilinx SDK	_	- 11				
File Edit Source Refactor Navigate Search Pro	ject Xilinx Tools Run Window Help						
🖻 🗸 🖯 🗗 👌 🐘 🗸 🐇 🖉 🕅	- 	🔉 🛪 🔪 🍅 🛷 🖛 🗐	<b>■ \$ + \$ + \$ 6 + 6 + </b>   =				
			Qu	ick Access			
Project Explorer 🛛 📄 🔄 🔻 🗖 🗖	system.hdf 🛛			🖺 🗄 Outline 🛛 🛞 Make Tar 🖓 🗖			
🕨 😂 perfmon	perftest wrapper by platform 0 Hard	ware Platform Specif	ication				
	perfect_mapper_nm_placform_orana	An outline is not available.					
i perftest_bd.tcl	Design Information						
pertest_wrapper.bit	Target EBGA Device: 77020		E				
ns7 init gplic	Created With: Vivado 2015.3						
ns7 initc	Created On: Thu Sep 24 08:58:11 2015						
lo ps7 init.h							
s7_init.html	Address Map for processor ps7_cortexa9_0						
😰 ps7_init.tcl	ps7 into dist 0.0xf8f01000.0xf8f01ff	f					
💼 system.hdf	ns7 scutimer 0.0xf8f00600.0xf8f0061	f					
▶ 10 standalone_bsp_0	ps7 slcr 0 0xf8000000 0xf8000ff	f					
	ps7_scuwdt_0 0xf8f00620 0xf8f006f	f					
	ps7_l2cachec_0 0xf8f02000 0xf8f02ff	f					
	ps7_scuc_0 0xf8f00000 0xf8f000fc						
	ps7_qspi_linear_0 0xfc000000 0xfcffffff						
	axi_perf_mon_0 0x43c00000 0x43c0fff						
	ps7_pmu_0_0xt8893000_0xt8893tf	t					
	ps/_ati_1 Ux18009000 0x1800911	E 6					
	ps/_am_0_0x100000000x10000011	1 f					
	ns7 afi 3 Oxf800b000 Oxf800bff	f					
	ps7_afi 2 0xf800a000 0xf800aff	f					
	ps7_globaltimer_0 0xf8f00200 0xf8f002f	f					
	ps7_dma_s 0xf8003000 0xf8003ff	f					
	ps7_iop_bus_config_0 0xe0200000 0xe0200ff	f					
	ps7_xadc_0 0xf8007100 0xf800712	0					
	ps7_ddr_0 0x00100000 0x3ffffff	f .	-				
	Overview						
👛 Target Connections 🕴 🔒 🚇 🗖 🗖	😨 Problems 🕱 🧔 Tasks 💻 Console 🔲 Propertie	s 📃 SDK Termi 🖳 🗖	SDK Log 🕱				
Aardware Server		$\bigtriangledown$	14:49:32 TNEO : Launching VS	DB server: xsdb bat C:/Xilinx/4			
🛛 🕞 Linux TCF Agent	0 items		14:49:34 INFO : XSDB server	has started successfully.			
D 🔁 QEMU TcfGdbClient	Description	Resource Path	14:49:34 INFO : Processing c	ommand line option -hwspec C:/>			
				-			
	•	۴	•	Þ			

x1219\_11\_101615

Figure 11: SDK Project Workspace

- 4. Make sure your computer is connected to a ZC702 evaluation board using two USB connectors. The first is connected to the programming cable and the second is connected to the UART so you can see the output on the terminal.
- 5. On the ZC702 board, flip the power switch to ON so you can program the device.
- 6. Connect to Terminal 1 to view the output from the C program. To connect to the terminal:
  - a. Open up the **SDK Terminal** tab at the bottom of the SDK window (shown in Figure 11 as **SDK Termi...**).
  - b. After you have the tab open, go to the upper right and click the + sign to add a terminal connection.



c. Change the settings to match those in Figure 12. Note that the COM3 might be a different COM# on your machine.

SDK	Connect to s	serial port 📃
ſ	Basic Setting	32
	Port:	сомз 🗸
	Baud Rate: 🛛	115200 👻
	▼ Advance :	Settings
	Data Bits:	8 -
	Stop Bits:	1
	Parity:	None 🔻
	Flow Contro	ol: None 🔻
	Timeout (se	c):
	ОК	Cancel

Figure 12: UART Terminal 0 Settings



To program the PL:

- Use the program FPGA command found under Xilinx tools. The bitstream pretest\_wrapper.bit is chosen automatically for you by SDK. Select Program in that screen.
- When the "DONE" LED (DS3) has gone High, create a new debug configuration by selecting Run > Debug Configurations and choose Xilinx C/C++ Application (System Debugger), as shown in Figure 13.

Debug Configurations	Maker State	X
Create, manage, and run configur	itions	Ť.
Performance Analysis     Target Communication Fram     Xilinx C/C ++ application (GD     Xilinx C/C ++ application (Sys     Xilinx C/C ++ application (Sys     Filter matched 5 of 15 items	<ul> <li>Configure launch settings from this dialog:</li> <li>Press the 'New' button to create a configuration of the selected type.</li> <li>Press the 'Duplicate' button to copy the selected configuration.</li> <li>Press the 'Delete' button to remove the selected configuration.</li> <li>Press the 'Filter' button to configure filtering options.</li> <li>Edit or view an existing configuration by selecting it.</li> </ul>	
?		Debug Close

x1219\_13\_101615

Figure 13: Debug Configuration Settings

3. After a new window has opened, change the debug type to **Standalone Application Debug**.



4. Change the name to **perf\_mon**. The window should be as shown in Figure 14.

ate, manage, and run configuration n or Debug a program using System Deb	<b>is</b> ugger.						Ŕ
Performance Analysis	Name: New_configura	tion Applicat	tion   🕬= Arguments   🌄 Environment   🖬 Syr	mbol Files 🦻 🦻 Sou	urce) 🔏 Pa	ath Map 🔲	Common
Target Communication Framewor Kilinx C/C++ application (GDB) Kilinx C/C++ application (System I Kilinx C/C++ application (System I	Debug Type: Standald	one Applio	cation Debug ▼ New				
Fr New_configuration	Hardware platform: Bitstream file:	latform: perftest_wrapper_hw_platform_0 le: perftest_wrapper.bit			Search	Browse	Generate
	FPGA Device: PS Device:	PGA Device: Auto Detect S Device: Auto Detect			Select	DLOMOSE	
	<ul> <li>Reset entire system</li> <li>Program FPGA</li> <li>Run ps7_init</li> <li>Run ps7_post_cor</li> <li>Enable Cross-Trig</li> </ul>	m nfig Igering	Summary of operations to be performed Following operations will be performed befor 1. Reset entire system. 2. Program FPGA fabric (PL). 3. Run ps7_init. (Only first time after System 4. Run ps7_post_config. (Only first time after	ore launching the reset or board po r System reset or b	debugger. wer ON) ioard powe	r ON)	*
rmatched 6 of 16 items	<u></u>					Apply	Revert
					C	Debug	Close

Figure 14: Debug Configuration Standalone Application Settings



5. Change to the **Application** tab and check the **Download Application** box. The project name and application boxes should be filled in automatically, as shown in Figure 15.

ok Debug Configurations						٤	
Create, manage, and run configuration Run or Debug a program using System Debu	<b>s</b> Igger,					Ś	
Image: Constraint of the second s	Name: New_configura	tion pplication 🛛 🕬= A 📝 Download app	Arguments 🚾 Environment	🖆 Symbol Files 🎉 Source	e) 🔏 Path Map 🗄	Common	
nx C/C++ application (GDB) nx C/C++ application (System Debugger o nx C/C++ application (System Debugger)	ps7_cortexa9_0 ps7_cortexa9_1	Project Name:	perfmon			Browse	
New_configuration		Application:	Debug/perfmon.elf		Search	Browse	
		Data Files to do	ocessor program entry o download before launch				
		File		Address		Add	
						Kemove	
						Relative	
Filter matched 6 of 16 items					Apply	Revert	
•					Debug	Close	

Figure 15: Debug Configuration Adding Download of Software

6. Select the box that says **Stop at program entry**. This ensures the program has an automatic breakpoint, and the program starts when you tell SDK.



7. Next, click the **Debug** button to start the debug session. A **Confirm Perspective Switch** dialog box appears. Select **Yes** in that box. Your SDK session is now as shown in Figure 16.

1993 Debug - perfmon/src/belloworld.c - Xilinx SDK		The second s	-	
File Edit Source Refactor Navigate Search Project Xilinx Tools Run Window Help				
Circle B     Circ	2 i+ =. 🕱 🍅 🖉		(-) ▼ (-) ▼ (-)	
		0	uick Access	📑 🛛 📅 C/C++ 🗱 Debug
At Dahura ₩	(X)= Variables 😚 🧟 Pi	vaakunginta 1919 Pagistara 🐼 VM	D Concolo 🔽 VSDI	R Concola 📑 Madular
E New configuration (local)		eacpoints and registers a xin		
APU	Nama	Turna	( <u></u> ) ~ 4	
🔺 🧬 ARM Cortex-A9 MPCore #0 (Breakpoint: main)	Marrie Marstatur	int		10/08/0
0x0010055c main():/src/helloworld.c, line 50	b + perfmon confia	XAxiPmon Confid	a *	{DeviceId=0x0004. BaseAd
UXUU103390_start(): xii-crtU.S, line 128	perfmon	XAxiPmon	,	{Config={DeviceId=0x7795
ARM Cortex-A9 MPCore #1 (Suspended)	(×)= fpga_clk1	u32		0x00118070
3 xc7z020	(x)= prev_samples	u32		0x0000001e
				A
				-
	4			4
🕞 system.hdf 🗮 Disassembly 🔯 helloworld.c 🔀		- 8	📴 Outline 🖾 🚺	╕ᆙᇲᇲᄵᆃᅑᄆᆸ
#define PEREMON BASEADDR XPAR AXI PERE MON Ø BASEADDR		*	stdio.h	
			u platform	uh -
e int main()			💶 xparame	eters.h
			📲 xaxipmo	n.h
<pre>printf("Performance test started\n\r"); int started</pre>		<b>—</b>	sleep.h	
<pre>init platform();</pre>		=	# esc	
			# pos()	
cls; pos(0,0):			# PERFMC	N_ID
F(-)-/)			# PERFMC	N_BASEADDR
XAxiPmon_Config* perfmon_config = XAxiPmon_LookupConfig(PERFMON_ID);			main():	int
XAxiPmon perfmon;				
<pre>status = XAxiPmon_CfgInitialize(&amp;perfmon, perfmon_config, PERFMON_BASEADDR);</pre>				
<pre>if (Status := XSI_SUCCESS) {     printf("Error initializing axi perfmon\r\n");</pre>		-		
•		4		
📥 Target Conn 🕱 🖳 🗖 📮 Console 🧟 Tasks 📮 SDK Terminal 🖹 Problems 🜔 Executables 🧻 N	1emory 🛛 🗖 🗍	🔄 SDK Log 🖾		
	🔢 🔄 🐻 🗸 🔻	08:52:14 INFO : Context	for processor	'ps7_cortexa9_0' is selec 🔺
> 🗁 Hardware Server Monitors 🚦 💥 💥		08:52:14 INFO : 'con' o	command is execu	uted.
🔉 🧁 Linux TCF Agent		targets -set -nocase -fil	lter {name =~ "#	ARM*#0" && itag cable name
QEMU TcfGdbClient		con		J
		End of So	ript	
		< III		• •
				,
				x1219 16 101615

Figure 16: Debug Perspective

- 8. Close the **SD log** tab.
- 9. Select the **Terminal 1** tab so you can see what the program prints.



10. Select the start button, shown circled in red in Figure 17.

The code in hello\_world now runs and shows the number of transactions for the ports that have the ATG connected to them. Then the program does a quick calculation and shows the MB/s. Your SDK session should now be as shown in Figure 17.

son Debug - perfmon/src/helloworld.c - Xilinx SDK	Contraction of the local division of the loc	and the second s			23
File Edit Source Refactor Navigate Search Project Xilinx Tools Run Window Help					
	ur. ið 🗟 🗷 🖉 🦉 /	✓ • ⊿ ½ • ♀ ↓ • ♥	← ▼ ♀ ▼   ™	- f	
		Qu	uick Access	🔡 🖬 C/C++ 🕸	Debug
🏇 Debug ⊠ 🦉 🙀 🖬 🔽 🖻 🗆	🛛 🕪= Variables 🖾 🤷 Break	points 배 Registers 📓 XMI	D Console 🖬 XSDB C	onsole 🛋 Modules	
			🖄 📲 🕒	8) / × 🖗 🗂	2
ARM Cortex-A9 MPCore #0 (Running)	Name	Туре	V	alue	^
ARM Cortex-A9 MPCore #1 (Suspended)	(x)= status	int XAviPmon Config	* (1	049840 DeviceId=0x0004 Base	Ad.
<b>6</b> <sup>6</sup> xc7z020	perfmon	XAxiPmon	{	Config={DeviceId=0x77	795
	•	III			•
					÷
	•		(=)		+
💼 system.hdf 🔤 Disassembly 🖻 helloworld.c 🛛		- 8	🗄 Outline 🖾 📄	∮ <u>*</u> ≷ ≷ ● ₩ ~	
#define PERFMON_BASEADDR XPAR_AXI_PERF_MON_0_BASEADDR		*	stdio.h		
			ynarionnin xparameter	s.h	
		=	🞴 xaxipmon.h	1	
<pre>printf("Performance test started\n\r"); int status:</pre>			sleep.h		
<pre>init_platform();</pre>			# cls		
cls;			# pos()	10	
pos(0,0); # PERFMON_ID # PERFMON BASEADDR				BASEADDR	
VAviDmon Confid* norfmon confid - YAviDmon LookunConfid/DEDENON TO).		*	main() : int		
🖳 Console 🖉 Tasks 📮 SDK Terminal 🕱 🦹 Problems 🚺 Executables 🔋 Memory		8]		<b>ਙ = + ×</b> <i>Q</i>	
Connected to: Serial ( COM4, 115200, 0, 8 )					
H0101 read count - 260271568 (256 000 MBstee/ser): H0101 write count - 260271656 (256 000 MBstee/ser)					*
HP[1] read count = 2092/1002 (200.000 MBytes/sec); HP[1] write count = 2092/1000 (200.000 MBytes/sec)					
HP[2] read count = 269271656 (256.000 MBytes/sec); HP[2] write count = 269271656 (256.000 MBytes/sec)					
HP[3] read count = 269271600 (256.000 MBytes/sec); HP[3] write count = 269271656 (256.000 MBytes/sec)					
ACP read count = 269271592 (256.000 MBytes/sec); ACP write count = 269271656 (256.000 MBytes/sec)					
GP read count = 134635792 (128.000 MBytes/sec); GP write count = 134635828 (128.000 MBytes/sec)					
				Send	Clear
				ocita	21551

x1219\_17\_101615

#### Figure 17: Debug Perspective with Results in Terminal 1

The results in the terminal should show that the HP ports and ACP all have about 256 MB/s, while the GP port has 128 MB/s. This shows how to get data from the APM using C code. However, in this version of SDK there are new features to give you the performance data for HP0-3 and the ACP automatically.



There is a modified version of perf\_mon that only reads the GP port, which you can use to read the GP port. The procedure is as follows:

- 1. Close this debug perspective by selecting **Window > Close Perspective** because you are going to start a new Performance Analysis Perspective in SDK.
- 2. To run a Performance Perspective, select **Run > Debug configurations** as you did with perf\_mon.
- 3. This time select a **Performance Analysis** perspective, as shown in Figure 18.



#### Figure 18: Debug Configuration Standalone Application Settings

4. Rename the configuration to be Performance. This time you do not need an application to read the APM since that will be done automatically by SDK.



5. Click on the **Debug** button and answer **Yes** to the question about switching perspectives. The result is shown in Figure 19.

Son Performance Analysis - perftest_wrapper_ht	w_platform_0_Traces/Trace	s/Run_Oct-05_	08-56/APM_Trace_C	Oct-05_08-56/APM_Tr	ace_Oct-05_08-56 Xilinx S	DK	
File Edit Navigate Search Project Xili	nx Tools Run Window	Help					
🖻 🗕 🗔 📥 🔜 📾 🖬 🖬 🖬	► ☆ • O	- 0 🕪		.e== *	<ul> <li></li></ul>	(⇒ + ⇔ +   ≓	
					a 0 4		
						Quick Access	Performance Analysis
Project Explorer 🔀 📃 🗖	🖂 PS Performance 🗠	PS Performan	ce 🖂 APM Perfo	orman 🖂 APM Pe	forman 🛛 🖂 MicroBla	ze Perfo 🛃 MicroBlaze P	)erfo 🔲 Statistics 🔲 Properties 👘 🗖
							ि
> 🚰 perfmon	00.00.00 000 000 000-00.0	1-30 637 024 41	0 TO 2	TOR	1 9.01	5 90	13 8074
🔺 💷 perftest_wrapper_hw_platform_0 📃	White Terror stienes and	1.55.057 024 41	3 3201_0	3201	2 3000	0 3000	1_5 3201_1
apmconfigs	Write Transactions per m	15	2000.0	2000.	2000	.0 2000	2000.0
📓 perftest_bd.tcl	Minimum Write Latency		32.0	32.0	32.0	32.0	32.0
📄 perftest_wrapper.bit	Maximum Write Latency		32.0	32.0	32.0	32.0	32.0
🖻 ps7_init_gpl.c	Average Write Latency		32.0	32.0	32.0	32.0	32.0
💼 ps7_init_gpl.h	Write Throughput (MB/s	ec)	256.0	256.0	256.0	256.0	0 256.0
c ps7_init.c							
💼 ps7_init.h	Read Transactions per m	s	2000.0	2000.	0 2000	.0 2000	J.0 2000.0
ps7_init.html	Minimum Read Latency		37.0	33.0	48.0	40.0	33.0
😰 ps7_init.tcl	Maximum Read Latency		81.0	73.0	83.0	74.0	69.0
system.hdf	Average Read Latency		38.5	34.2	49.6	42.3	35.2
▷ 🛃 perftest_wrapper_hw_platform_0_1 +	Read Throughput (MB/s	ec)	256.0	256.0	256.0	256.0	0 256.0
•							
	× [			, III ,			4
🍇 🕅   🖬 🎽	ulut Mistanava 😒 🖬 🕫	tana 📑 Calan	1				
🔺 🎲 APU 🔷		ters - Color	,				
a 🔊 ARM Cortex-A9 MPCore #(			1			-	
0xfffff28	Selection Start 17:00:00.	000 000 000	8				
= 0xffffff20	Selection End 17:00:00.	000 000 000					
■	Window Span 000	100 000 000	0				
👂 🔊 ARM Cortex-A9 MPCore #1 🖕			000000000000000000000000000000000000000	0			00:00:00:100:000:000
< III +	8						
	o Birddhiadallain	additional and de	dificient of the dama	d de la di Millio de la de	undelin er eli bil en klii	hadaa aadda ar dd Albidhaadd da yn he	ndahi bila dadi dali mbilahi dali
🖉 🗏 🛃 🕶 😭 🖛	00000000000000000						00.01:39:637 024 42
TCF Debug Virtual Terminal - ARM Cortex-A9	Run Oct-05 08-56/A	PM Trace Oct-	05 08-56				
*	🕒 🔤 🔤 🔤	icaccembly	helloworld c	😤 vil in c 📲	Run Oct-05 08-56/DS Tra	-• Oct-05	Ict-05 08-56/ADM Trace Oct- 22
		r				ic_occosiii ican_o	
	Timestamp	Event type	Contents				
	🚿 <srch></srch>	<srch></srch>	<srch></srch>				
	00:00:00.000 000 001	APM	SLOT_0 Write Byte:	s=4963496, SLOT_0 Wi	ite Transactions=38777, SLC	)T_0 Write Latency=1240864,	, SLOT_0 Read Bytes=4963496, S_OT_0 Read
	00:00:00.022 561 34	APM	SLOT_0 Write Byte:	s=5775672, SLOT_0 W	ite Transactions=45123, SLC	)T_0 Write Latency=1443936,	, SLOT_0 Read Bytes=5775696, S_OT_0 Read
	00:00:00.033 376 91	APM	SLOT_0 Write Byte:	s=2768816, SLOT_0 W	ite Transactions=21631, SLC	)T_0 Write Latency=692192, S	SLOT_0 Read Bytes=2768768, SLOT_0 Read T
	00:00:00.056 223 510	APM	SLOT_0 Write Byte:	s=5848712, SLOT_0 W	ite Transactions=45693, SLC	T_0 Write Latency=1462176,	, SLOT_0 Read Bytes=5848752, SLOT_0 Read
-	00:00:00.338 838 76	APM	SLOT_0 Write Byte:	s=72349440, SLOT_0 V	/rite Transactions=565231, S	LOT_0 Write Latency=180873	360, SLOT_0 Read Bytes=72349512, SLOT_0 F
• III •	*						······

x1219 19 101615

Figure 19: Performance Perspective

The results are the same as shown by reading the APM. There is a little more information because SDK not only reads the APM but the PMU in the PS to give statistics about the APU or APM. The PL and PS performance tables are graphs of the tables in the summary. The summary is a snapshot in time but the graphs show the data over time. The MicroBlaze<sup>™</sup> processor tabs are not part of this application note and you can close them.

Now you have some performance data, but what does it all mean? The first step is to understand what each of the values represents. The second and harder step is to analyze this data and determine if the performance that is desired has been achieved.



Start by selecting the **APU Performance Summary** tab. This tab summarizes the PMU counters and APM metrics in one place. The first table shows the values from the PMU counters that give metrics on each of the Cortex-A9 processors. They deal with the level 1 (L1) cache for each processor and the level 2 (L2) cache, which is a unified cache shared by the two Cortex-A9 cores. The PS performance metrics include the following:

- CPU utilization (%): Percentage of non-idling CPU clock cycles.
- CPU instructions per cycle (IPC): Estimates the 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: Estimates the number of stall cycles per instruction due to memory writes (write) and data cache refills (read).

The APM metrics are defined as follows:

• Read/Write Transactions.

The number of read or write transactions issued by the ATG, typically a large number because there are many transactions issued by the ATG.

• Read/Write Latency - Average.

The latency of the transaction. This is measured from the acceptance of the command to the last word of the burst.

Because the latency is measured as stated above, writes are measured as they are accepted, making the latency low. Thus, if there is no back pressure in the system, this number is slightly higher than the burst size. This was done to show when back pressure starts for the write transactions. You can tell when the writes get stalled in the slave this way. When the write latency goes beyond the burst length, these transactions get stalled in the slave, which is the PS in this design.

• Read/Write Latency - Std. Dev.

This is the standard deviation of the latency measurements.

• Read/Write Throughput (MB/s).

This is the calculated throughput on the interface based on the number of transactions, size of the transactions, and speed of the interface.

Look at the PL performance over time. The ATGs are running when the board comes up. There is a constant or a logic 1 connected to the core start, so it runs after the Zynq-7000 AP SoC is out of reset. The ATGs are also set up to deliver a constant amount of traffic, so the transactions



are a line across the graph, as in Figure 20. The orange box with the + sign is a zoom in and out feature for the graph. This box is circled in Figure 20.



x1219\_20\_101615

*Figure 20:* **PL Performance Panel in Performance Analysis Perspective** 

The top portion of this tab shows the zoom of the area in the orange +. You can click in any of the graphs to place the zoom box. Using the right mouse button, click and drag the zoom box to the area you want to see in more detail. You can also make the zoom area larger or smaller by pressing the CTRL key and using the scroll wheel on your mouse.

The PS Performance tab is similar to the PL, but it graphs the PMU counters and not the data from the APM. With the knowledge of what each of the values represents, you can decide if this meets your requirements. This will vary from application to application. The first item is to make sure that the ATGs are set up with the desired traffic. If the bandwidth and latency are met for the application with the software running, you are done. But, what if your performance items are not met?

There is no simple answer to this question. There is usually not just one parameter that can be adjusted to make performance. There are several items to look at. As mentioned previously, how much traffic goes between the PS and PL? If you are using around 80–90% of the theoretical maximum of the PS DDR, there could be issues. Another item to examine is if all the



traffic is on one or two of the HP ports. If so, can this traffic be divided among more ports? Is the model for the system for the peak traffic, which only occurs 10–50% of the time, yet the model is trying to move the peak traffic 100% of the time?

This is a complex issue of how to tune the Zynq-7000 AP SoC for performance. The one item that is not covered is that the settings for one application might not work for another application because of the different flows and performance needs of the design. For additional information about this topic, refer to *SDK User Guide: System Performance Analysis* (UG1145) [Ref 1] and *UltraFast Embedded Design Methodology Guide* (UG1046) [Ref 4].

**Note:** SDK 2014.3 has the limitation that if the APM counters are read, all Profile Counters will be reset. So, if two entities read the APM, the results will not be correct. If you can, use SDK or the perf\_mon C code to read and view the bandwidth. The recommended method is SDK. The perf\_mon code is shown to read the GP port as needed.



## Modifying and Adding to the Model

There are two ways to take this model and make it more like your application. The first is to add software, and the second is to change the hardware model to mimic the data flows seen in your application. In the previous section, you added some software to the model. To add software, you can replace the hello\_world code in SDK and add your own. The code in hello\_world can be incorporated into your code to read the APM on the GP ports, if that is important. For more information about SDK, refer to the *Zynq-7000 All Programmable SoC Software Developers Guide* (UG821) [Ref 5], or the "Embedded System Design Using the Zynq Processing System" chapter of the *Zynq-7000 All Programmable SoC: Concepts, Tools, and Techniques (CTT): A Hands-On Guide to Effective Embedded System Design* (UG873) [Ref 6]. See also the Embedded Design hub in Document Navigator for useful documents and videos.

1. Select the C/C++ in the import, and select the directory with the C code. Figure 21 and Figure 22 show the windows to import C or C++ code.



Figure 21: Import Existing C code into SDK



New Project		~	
Import Existing Code			
Create a new Makefile project from existing code in that same directory			
Project Name			
		C_	702_HwPlatform
Existing Code Location Browse For Folder	_		23
Select root directory of existing code			
Nesktop			
D Contractions			
Gomputer			
Simple Computer			
NewsGator Social Sites Outlook Plugin V4.0			
Folder: Forrest Pickett			
	<u> </u>		
Make New Folder	Ok		Cancel

x1219\_22\_120314

Figure 22: Import Existing Code



2. Make a new Performance Perspective (Figure 23).

Debug Configurations		-	and the local division of	-	
Start Performane Analysis using System Debu	gger				30-
Image: Second Secon	Name: New_configu Target Setup Available Processor ps7_cortexa9.0 ps7_cortexa9_1	Application          3       Image: Constraint of the second s	ATG Configuration (** Argum olication memstride C:\Tech_Tips\SDK_Perf_Desig ssor gram entry wnload before launch	ents) 🖾 Environment) 🐩 Symbo	Files       **2         Browse         Search         Browse         Add         Remove         Relative
✓         III         ►           Filter matched 12 of 16 items         ►					Apply Revert
0					Debug Close

x1219\_23\_120314

*Figure 23:* **New Configuration** 

See References for more documentation on SDK and software debug.

The second way is to change the hardware model. This can be done in four ways:

- Add more ATGs
- Replace the ATG with your own IP
- Change the traffic flow of the existing ATGs
- Add your own IPI design with APM added

The hardware design is part of this application note, so you can make these modifications or additions.

Adding an ATG is not covered, and the only port left to connect an ATG to the PS is the other GP port. This can be done by copying how the ATG in the design is connected to GP0 and making another TrafficGen connection to GP1. Then, connect the signals to where they need to go, similar to the way the TrafficGen is connected to GP0.



Replacing an ATG with your own IP takes more effort and understanding of Vivado IP Integrator. There are references later in this application note to help you do that. The major item to ensure is that the core has an AXI3 or AXI4 memory mapped interface. If it is AXI4, then there needs to be an interconnect like the one used in the TrafficGen block to connect to the PS.

Changing the flow of ATG configurations to closer mimic application data flows is covered here. To change the ATG configuration:

- 1. Open the design you created at the beginning of this application note in the Vivado IDE (see An Instrumented Design for Performance Analysis).
- 2. Double-click any of the TrafficGen models to get into the hierarchy.

The TrafficGen blocks are numbered 0–5. The first four (TrafficGen0–3) are connected to the HP0–3 ports, TrafficGen4 is connected to the ACP, and TrafficGen5 is connected to S\_AXI\_GP0.



3. After getting into the hierarchy, double-click the **axi\_traffic\_gen\_0** in the diagram. This opens up an ATG configuration pane as shown in Figure 24.

🖵 Re-customize IP	-		X
AXI Traffic Generator (2.0)			
M Documentation 🗁 IP Location			
	Concept Name particult with traffic and 0.0		
Show disabled ports			
	Pronie Selection     Oustom     OHigh Level Traffic		
	High Level Traffic		
	Traffic Profile           Video         PCIe         Ethernet         USB         Image: Display to the second sec	ata	
	AXI Options		
	AXI Master Width 64  AXI Base Address 0x20000000		
	AXI High Address 0x3FFFFFF		
	Traffic Gen		
_s_axi_aresetn M_AXI⊕	Repetitive     One-Shot		
-core_ext_start irq_out - -core_ext_stop	DATA Mode		
	Pattern Options	Transaction Type	
	Traffic Pattern Fixed T	Transaction Type Read Write    Read Store (%)	
	Transfer Length (Max) 256	Write Share (%) 50 [0.100]	
	Transfer Length (Avg) 16 V		
	Inter Transaction Gap	Transaction Addressing	
	ITG Type Fixed 👻	Addressing Fixed -	
	ITG (Clocks) 14 [01023]	Transaction Seed 1 [1100]	
× ×			
		OK	Cancel
			x1219 24 120314

Figure 24: Changing the ATG Configuration

The first item to change is the profile. The custom mode allows you to program the ATG using the Cortex-A9 in the Zynq-7000 platform, or other CPU (such as MicroBlaze processor) in the PL. To do this, the documentation on the ATG in the *AXI Traffic Generator Product Guide* (PG125) [Ref 2] will be needed. The driver software can be located at

C:\Xilinx\SDK\2014.3\data\embeddedsw\XilinxProcessorIPLib\drivers\ trafgen\_v3\_2\doc\html\api\index.html. Use this mode if you want to implement dynamic control of the traffic, which is beyond the scope of this application note.

The next item to configure is the Traffic Profile. There are four predefined profiles for your convenience. They are Video, PCIe, Ethernet, and USB. (For more information about the profiles,



refer to the *AXI Traffic Generator Product Guide* (PG125) [Ref 2].) With each of these modes you can choose different items about the interface and the traffic it produces. If none of these predefined modes meet the traffic you would like to model, then there is the Data mode. This is the mode used in the example. You can change the following items:

- AXI options: data width, base address, and the high address of the transactions generated. These are set for the DDR of the Zynq-7000 AP SoC. One change might be to make these to the on-chip memory (OCM).
- DATA mode:
  - Pattern (fixed or random size), minimum and maximum transfer size for random size, and transfer length for fixed size.
  - Transaction types. Does the ATG do reads, writes, or reads/writes, and percentage of each.
  - The settings for this example have fixed 16-word burst, and 50% read and 50% write.
  - Inter-transaction gap, fixed or random. The setting for the gap is 50 cycles. These are clock cycles for the ATG. This example has a 100 MHz clock.
  - Transaction addressing. Fixed only. This is a limitation where the address could be incrementing or random.

Taking this design and changing the ATGs, a model for application can be made for the Zyng-7000 AP SoC.

**Note:** If the system is stressed, the results in the Performance Visualization will be incorrect. An example of stressing the system is having the reference design shown in the first section to have less than 30 cycles of interval for each ATG. The traffic that this produces is 844 MB/s on each AXI interface for a total of 4,220 MB/s. This is almost 100% DDR utilization at 533 MHz and is not really achievable by most (if not all) DDR controllers. Additional information about stressing the Zynq-7000 AP SoC is provided in *SDK User Guide: System Performance Analysis* (UG1145) [Ref 1].

Another strategy to be considered is replacing an ATG with existing IP (which need to be imported into the Vivado tools already). To add IP, delete an ATG and add your own IP. This entails a working knowledge of Vivado architecture and IP integrator. If you are not familiar with Vivado architecture and IP, see *Vivado Design Suite Tutorial: Designing with IP* (UG939) [Ref 7] and *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 8]. The Vivado 2014.2 - Design Hub in Xilinx Document Navigator [Ref 9] also lists other documents and videos that can be useful.

## Conclusion

This application note describes a holistic way to look at performance for All Programmable SoCs. By looking at performance in this manner, a user can determine whether or not a device can meet their performance needs. A user could just look at CPU speed, but without an effective memory controller the CPU would look a lot slower.

This application note includes a reference design that can be used to model an application in Zynq-7000 AP SoCs, showing what the real performance can be. This model uses a design in



actual hardware that you can combine with your software. This allows you to model your application and evaluate the performance very early in the design cycle.

There is a way to augment this model by changing the ATG or replacing it with your own IP, to better model the application. This allows a deeper examination of what will and will not work on Zynq-7000 devices.

These features of SDK, along with the IP for performance allow you to get a high degree of confidence that Zynq-7000 AP SoCs can meet the performance needs of your application.

## **Reference Design**

You can download the <u>Reference Design Files</u> for this application note from the Xilinx website.

Table 2 shows the reference design matrix.

Parameter	Description		
General			
Author	Forrest Pickett		
Part	XC7Z020		
Source code provided?	Yes		
Source code format (if provided)	C		
Design uses IP from existing reference designs	AXI Traffic Generator		
	AXI Performance Monitor		
Simulation			
Functional simulation performed	N/A		
Timing simulation performed?	N/A		
Test bench provided for functional and timing simulation?	N/A		
Test bench format	N/A		
Simulator software and version	N/A		
SPICE/IBIS simulations	N/A		
Implementation software tool(s) and version	Xilinx SDK, 2014.3		
Static timing analysis performed?	N/A		
Hardware Verification			
Hardware verified?	N/A		
Platform used for verification	Vivado Design Suite 2014.3		

#### Table 2: Reference Design Matrix



#### References

- 1. SDK User Guide: System Performance Analysis (UG1145)
- 2. LogiCORE IP AXI Traffic Generator v2.0 Product Guide (PG125)
- 3. LogiCORE IP AXI Performance Monitor v5.0 Product Guide (PG037)
- 4. UltraFast Embedded Design Methodology Guide (UG1046)
- 5. Zynq-7000 All Programmable SoC Software Developers Guide (UG821)
- 6. Zynq-7000 All Programmable SoC: Concepts, Tools, and Techniques (CTT): A Hands-On Guide to Effective Embedded System Design (<u>UG873</u>)
- 7. Vivado Design Suite Tutorial: Designing with IP (UG939)
- 8. Vivado Design Suite User Guide: Designing with IP (UG896)
- 9. Vivado 2014.2 Design Hub in Document Navigator (Document Navigator is accessed from Vivado Design Suite)
- 10. AXI Reference Guide (UG761)

#### **Revision History**

The following table shows the revision history for this document.

Date	Version	Revision
12/11/2014	1.0	Initial Xilinx release.
11/05/2015	1.1	Updated Figure 4, Figure 5, and Figure 9 to Figure 20. In System Performance Analysis, updated step 6a, step 6b, step 8, and paragraph and note after Figure 20.



#### **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 <a href="http://www.xilinx.com/legal.htm#tos">http://www.xilinx.com/legal.htm#tos</a>; 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 <a href="http://www.xilinx.com/legal.htm#tos">http://www.xilinx.com/legal.htm#tos</a>.

#### Automotive Applications Disclaimer

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

© Copyright 2014–2015 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. Cortex is a trademark of ARM in the EU and other countries. PCI, PCIe, and PCI Express are trademarks of PCI-SIG and used under license. All other trademarks are the property of their respective owners.