XAPP1031(v1.0.1)
December 19, 2007

Decreasing Simulation Runtimes with System Generator for DSP Hardware Co-Simulation

Author: Jacobus Naude

Summary

This document provides an overview of Hardware Co-Simulation in System Generator for DSP from a performance perspective, and provides information to help reduce long simulation run times.

After reading this document, you will:

- have a basic understanding of how a System Generator for DSP design fits into the Simulink® simulation environment.
- understand the motivation for and operation of frame-based Hardware Co-Simulation implementations.
- be able to choose an implementation approach that provides the shortest simulation runtime for a specific simulation. This application note serves as a reference for detailed information and design files for each implementation described.
- acquire other helpful information to consider (apart from the implementation approach) that can also decrease simulation runtimes.

This document contains detailed tables and descriptive graphs. All the example designs used for analysis are included in the ZIP file that can be downloaded along with this application note. The ZIP file also contains an Excel spreadsheet which can be used as a template to graph your own results.

Introduction

System Generator for DSP provides the functionality for performing Hardware Co-Simulation for designs that run both in hardware and in software. Hardware Co-Simulation can verify the operation of designated parts of a design in hardware to significantly decrease simulation runtimes. This is a powerful feature of System Generator for DSP, especially when considering the parallel nature of FPGA devices. Hardware Co-Simulation can make it possible to complete even very long simulations within a much shorter period of time.

Many factors can influence runtimes. This document summarizes and analyzes these factors, providing a quick, detailed reference for making informed design and implementation decisions.

The design examples described in this document are small and simulate quickly when the number of Simulink simulation cycles are kept relatively small. However, in practice, the requirements of a specific run are not always determined by the number of Simulink simulation cycles. In most cases, termination criteria of a simulation is based on the number of data samples to be processed.

To illustrate this, consider calculating the Bit Error Rate (BER) of a communication system. To calculate a single point on a BER plot, a large number of data must be processed by the system being analyzed. To plot the BER points of a system up to 10^{-9}, the system must process 10^9 points, of which one bit will contain erroneous data at a certain Eb/No value. To create a plot that contains sufficient points to draw a proper curve, several points need to be processed, which requires a large amount of data be processed. In such a scenario, Hardware Co-Simulation becomes a very attractive alternative.

Note: A good practical example of using Hardware Co-Simulation can be found in [Ref 3] in which BER measurements of a 3GPP Turbo Encoder/Decoder Forward Error Correction system is performed.

Simulink Simulation Environment Overview

System Generator DSP simulation performance results are analyzed by reviewing simulation run times. In addition, the way that a System Generator for DSP simulation fits into a Simulink simulation also affects performance. The following sections are included to provide background about how System Generator for DSP blocks fit into the Simulink simulation environment. This information is necessary for understanding some parts of the analyses presented in this document.

System Generator and the Simulink Environment

The clock used in a Simulink simulation is comprised of steps that execute over the time period defined prior to starting a simulation. Step size is determined by the Simulink solver, and can be either **fixed** or **variable**. The solver uses the settings specified in the Solver pane of the **Simulation > Configuration** menu. System Generator for DSP only supports variable-step solvers and only this approach is discussed.

To explain how the step size is determined, consider a Simulink simulation environment that contains two Simulink blocks and a System Generator design (represented by the System Generator token), as shown in Figure 1. In this illustration, variable step sizes in the Simulink time vector will accommodate all the multiples of the design blocks' sample times, as demonstrated below:

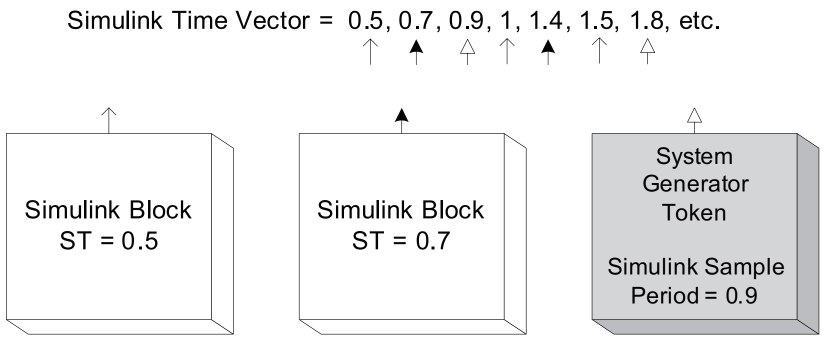

Simulink Time Vector = 0.5, 0.7, 0.9, 1, 1.4, 1.5, 1.8, etc.
Simulink Block ST = 0.5
Simulink Block ST = 0.7
System Generator Token
Simulink Sample Period = 0.9

Figure 1: **How Simulink Step Sizes are Determined**

It is easier to understand how the blocks in a design are called and change state when a clock is present. Figure 2 illustrates this using clock-enable-type (CE) signals to depict the number of times a block changes state in a simulation. Clock-type inputs (Call inputs) illustrate the number of times a block is called.

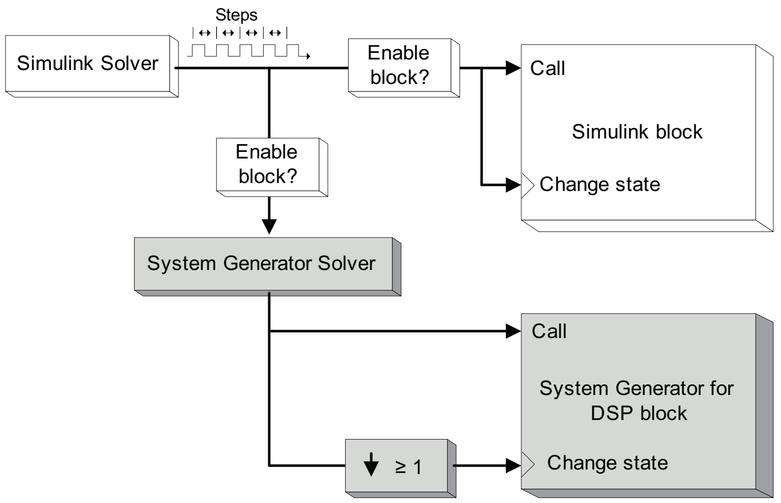


Figure 2: **System Generator for DSP in a Simulink Simulation Environment**

If the current step in the time vector matches an integer multiple of a Simulink block sample period, the block is activated. The same is true for the System Generator part of the design, in which the current step should match the *Simulink sample period* parameter for the System Generator token. When activated, the System Generator solver calls all the System Generator blocks and produces a clock-type signal to drive the System Generator part of the design.

A major difference between the Simulink and System Generator blocks is that a System Generator block is called on every clock cycle (of the clock produced by the System Generator solver), while its state only changes on clock edges (steps) defined by its *sample time* parameter. This is illustrated using the down sample block in Figure 2. System Generator requires all blocks in a design to be sampled at a time that is an integer-multiple of the Simulink system period. This is because the System Generator part of a design is created for implementation in hardware.

Simulink provides a Profiler tool that captures and reports the number of calls and state changes made in a design. This tool can be found and enabled by using the Tools menu in Simulink. This tool was used to analyze the different implementations described in "Appendix A: Detailed Implementation Information." The results are also provided in the appendix.

Hardware Co-Simulation

It is important to understand how a Hardware Co-Simulation fits into a Simulink simulation. There are two clocking schemes available: single-stepped, and free-running.

- In a **single-stepped** simulation, the clock from the System Generator solver is sent to the logic running in hardware over the Hardware Co-Simulation interface.
- In a **free-running** simulation, the part of the design that is running in software is not synchronized with the part running in hardware. The clock driving the logic in hardware is typically an onboard oscillator, which easily outperforms the single-stepped clock provided by Simulink.

Effects of Block Calls and State Changes on Simulation Runtimes

The number of calls and state changes provide helpful information for analyzing different design implementations. A good example to demonstrate how the number of calls and state changes in a design can be used to increase performance is a frame-based Hardware Co-Simulation implementation (detailed in the following section). A frame-based Hardware Co-Simulation contains logic in hardware that is only called (woken-up) on every n^{th} clock cycle, while the Simulink blocks are called on every clock cycle. Because the number of data transfers between the software and the hardware is less frequent, simulation times are reduced. The design presented in "Free-Running Simulation with Shared Memories Configured as Shared FIFOs," page 23 is a practical example this.

Notes:

1. The changing of states is known as solving the model, thus the name Solver. A description of the mathematical models used during the solving of the model is beyond the scope of this document, but more information can be found in [Ref 1].
2. Fixed-step solvers may outperform variable-step solvers in certain models because variable-step solvers determine if the blocks in the design need to be activated continually during simulation. **However, the fixed-step solver is not supported** in System Generator for DSP. With System Generator for DSP 9.2i, a warning message appears when a simulation is run with a fixed-step solver.

Frame-Based Data Transfer System Overview

The following section, "Different Implementation Methods," demonstrates that implementations with frame-based transfers between Simulink and the logic that are implemented in hardware provide the most significant drop in simulation runtimes. Because of this, a section explaining the motivation for and operation of frame-based transfers is included.

As mentioned in the "Simulink Simulation Environment Overview" section, a free-running clocking scheme normally outperforms a single-stepped clocking approach. However, such an approach also results in the software and hardware being out of sync, making it necessary to discuss the interface needed to cross the two clock domains within a simulation if communication between the software and hardware is required. System Generator for DSP provides shared memories for this purpose. Three types of shared memory are available: Shared Registers, Shared FIFOs, and Shared Memory. These memories should be compiled as described in the "Single Shared Memory Compilation" section of the *System Generator for DSP User Guide* [Ref 2] when using them to cross the two clock domains. A brief description of each type is discussed below.

- **Shared Registers:** A shared register is controlled by a clock from one clock domain only, as shown in the "Co-Simulating Shared Registers" section of the *System Generator for DSP User Guide* [Ref 2]. Because of this, it is not an optimal solution for crossing clock domains.
- **Shared FIFOs**: This type of shared memory is optimal for crossing clock domains because it provides all the control options found on normal FIFOs—commonly used for synchronization. This option can use either a free-running or single-stepped clocking scheme. The free-running approach provides the best results.
- **Shared Memory:** Shared Memory blocks can be configured as Shared FIFOs or as Lockable Shared Memory. Shared FIFOs provide the control features needed for crossing clock domains. The Shared Memory blocks can use both a single-stepped and free-running clock when configured as Shared FIFOs. When configured as Lockable Shared Memory, a free-running approach must be used to avoid the time-outs that will occur when using a single-stepped clock. In both cases, a free-running clock produces the best results. See the "Co-Simulating Lockable Shared Memories" section of the *System Generator for DSP User Guide* [Ref 2] for more information.

In summary, a free-running clock is the best choice, and the options available for solving the problem of software and hardware logic not being synchronized were outlined. A piece of logic in hardware runs much faster than the rest of the simulation, where the input to the hardware logic is transferred to the shared memory interface on every Simulink clock cycle. It may seem that this setup could easily outperform a simple software-only simulation, but sending data over the shared memory interface causes an overhead that can actually make it slower. Figure 3 illustrates this behavior.

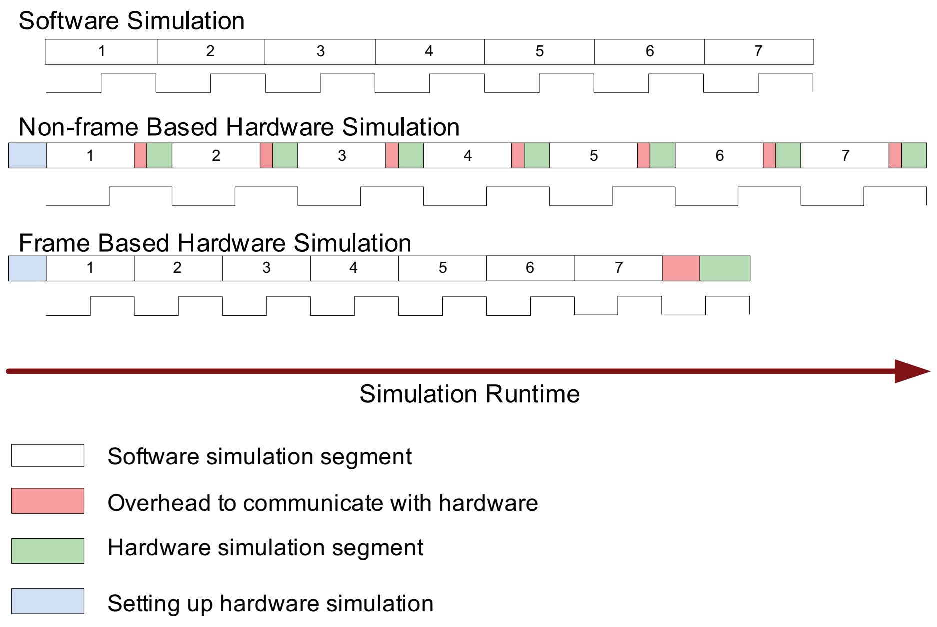

Software Simulation
1 2 3 4 5 6 7
Non-frame Based Hardware Simulation
1 2 3 4 5 6 7
Frame Based Hardware Simulation
1 2 3 4 5 6 7
Simulation Runtime
Software simulation segment
Overhead to communicate with hardware
Hardware simulation segment
Setting up hardware simulation

Figure 3: **Effect of Frame-based Transfer on Simulation Runtime**

It is assumed that a hardware simulation segment is completed at a faster rate when compared to the same amount of processing in a software simulation segment. This is true when the logic processed in hardware would cause a bottleneck in the Simulink simulation, or when the hardware logic is clocked by a free-running clock.

In Figure 3, seven blocks of data are shown queued for processing. Because communication with the hardware only occurs once during each frame, frame-based simulations runtimes are the shortest. A simulation normally consists of multiple simulation lines to form a longer simulation, depending on the runtime. Therefore, the longer the simulation time (or larger amount of data to be processed), the greater the difference between the two methods becomes.

Notes

1. This is a very simplified demonstration. A real simulation contains more building blocks than the four used in this explanation.
2. The three timelines shown in Figure 3 are used to illustrate frame-based transfers only. Timelines may differ, depending on the design used. For example, a non-frame-based hardware simulation containing no shared memory will outperform a software simulation during long simulation times. This is demonstrated in the results described in the section, "Different Implementation Methods."
3. For the three timelines shown in Figure 3, it is assumed that the software simulation is performed on a model without any shared memory (similar to the model described in the "Simulink Simulation" section). For the second timeline, a single stepped model that includes shared memory will behave in this way (similar to the model described in the "Single-Stepped Hardware Co-Simulation" section). The third timeline is based on a model containing a frame-based shared memory interface between the software and hardware (similar to the model described in the "Free-Running Simulation with Shared Memories Configured as Shared FIFOs" section of this document.)

A frame-based system is shown in Figure 4.

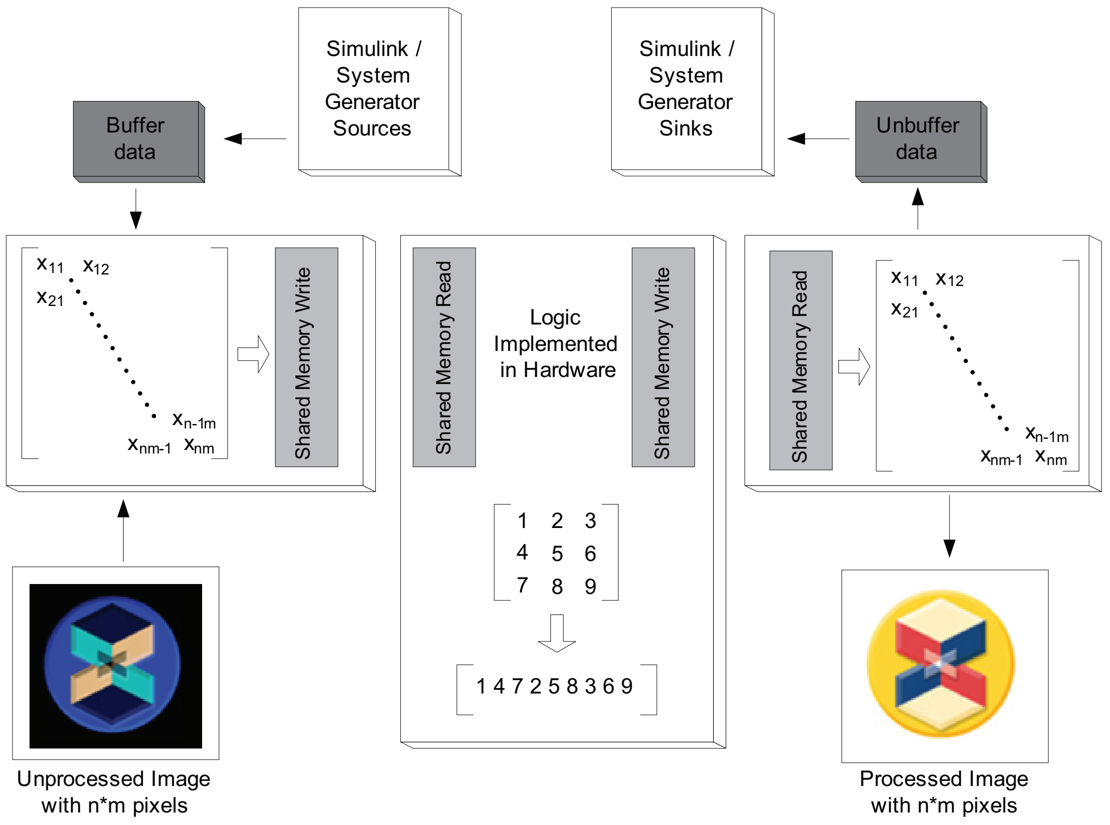

Simulink / System Generator Sources
Buffer data
Simulink / System Generator Sinks
Unbuffer data
x11 x12
x21
xn-1m
xnm-1 xnm
Shared Memory Write
Shared Memory Read
Logic Implemented in Hardware
Shared Memory Write
Shared Memory Read
1 2 3
4 5 6
7 8 9
1 4 7 2 5 8 3 6 9
Unprocessed Image with n*m pixels
Processed Image with n*m pixels

Figure 4: **Frame-based Data Transfer System Overview**

In Figure 4, the blocks on the left and right represent blocks running in software. The center block represents the logic that is implemented and running in hardware. The system contains shared memories used for synchronization and to transfer data between software and hardware clock domains. This figure also shows that data is buffered until enough data to form a single frame is collected.

The shared memories can accept input data in three ways: Scalar, Vector and Matrix. Which one is used depends upon the application. Two options are illustrated in the above figure. One option is to use a Simulink buffer to form a one-dimensional vector to fit an application that uses one-dimensional block/array data. The other option shows input data as an image consisting of n x m pixels. In such an application there is no need to buffer data as a full matrix is available on every *sample period.* An example of such a system is described in the “Real-Time Signal Processing Using Hardware Co-Simulation” section of the *System Generator for DSP User Guide* [Ref 2].

Notes

1. The Shared Memory blocks can either be configured as Shared FIFOs (see “Free-Running Simulation with Shared Memories Configured as Shared FIFOs,” page 23) or Lockable Shared Memory (See “Free-Running Simulation with Lockable Shared Memories,” page 25).
2. The depth and width of the shared memories match the depth and width of the input and output matrixes.

Different Implementation Methods

In this section, a simple model is described and is followed by a short overview of seven different implementations that can be used to simulate the design. These approaches are then compared and contrasted. Detailed descriptions of every implementation approach can be found in corresponding subsections of "Appendix A: Detailed Implementation Information," page 18.

Demonstration Design

Figure 5 illustrates a simple 32-tap MAC FIR that removes high frequency components from a noisy input signal. The left-hand side of this figure shows a slider gain block that controls the amount of noise added to the generated sine wave. The scope on the right is used to compare the noisy output to the filtered clear output. The expected output on the scope is shown in Figure 6.

Note: This model is described in the "Frame-based Acceleration using Hardware Co-simulation" section of the *System Generator for DSP User Guide* [Ref 2].

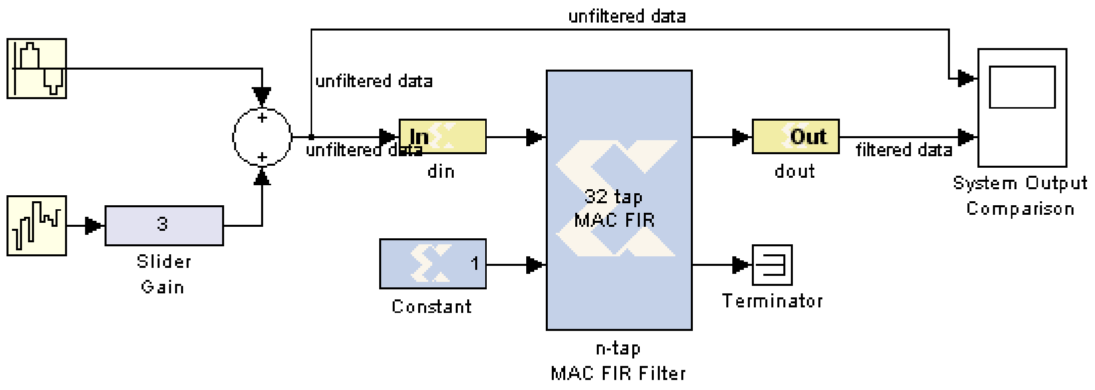


Figure 5: **Demonstration Design Comparing Simulation Runtimes**

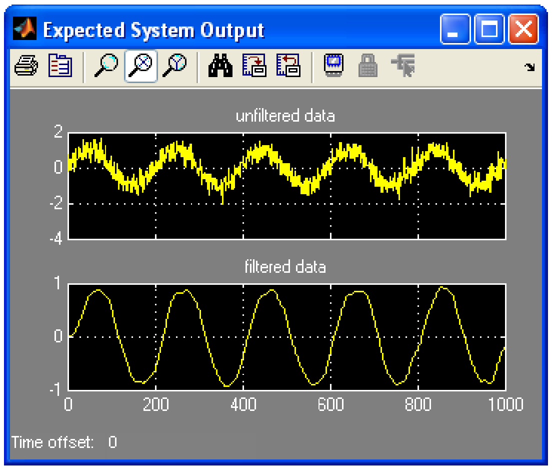


Figure 6: **Demonstration Design Expected Output**

Overview of Implementation Approaches

A short overview of each implementation method is provided in the following sections. For detailed descriptions of each approach, please see "Appendix A: Detailed Implementation Information."

Simulink Simulation

A normal Simulink simulation was performed on the basic demonstration model that is shown in Figure 5. See "Simulink Simulation," page 18 for detailed information.

ModelSim Simulation

The demonstration model was generated with the Compilation target option set to "HDL Netlist" and the Create test bench option enabled. The generated HDL was then simulated using the Mentor Graphics® ModelSim® simulator. The source file compilation, design load time, and simulation runtimes were recorded and are used for comparison with the other implementations. See "ModelSim Simulation," page 19 for more information.

Software-based Shared FIFOs

In the second approach, Shared FIFOs are used to communicate between the top-level design blocks and the lower-level filter logic. See "Software-Based Shared FIFOs," page 20 for more information.

Single-Stepped Hardware Co-Simulation

The lower-level filter logic was compiled into a Hardware Co-Simulation token in this implementation. This is a normal, basic single-stepped Hardware Co-Simulation of the design. See "Single-Stepped Hardware Co-Simulation," page 21 for more information.

Hardware-Based Shared FIFOs

This approach is almost identical to the Software-based Shared FIFOs approach. The difference is that the lower-level filter logic and shared FIFOs are compiled into a Hardware Co-Simulation token and clocked by a single stepped clock. See "Hardware-Based Shared FIFOs," page 22 for more information.

Free-Running Simulation with Shared Memories Configured as Shared FIFOs

This implementation contains a Shared Memory interface between the top-level design blocks and the logic implemented in hardware. The data is passed over the Shared Memory interface in a frame-based manner as described in "Frame-Based Data Transfer System Overview," page 4 of this document. The Hardware Co-Simulation token is clocked by a free-running clock, and the Shared Memory blocks are configured as Shared FIFOs. See "Free-Running Simulation with Shared Memories Configured as Shared FIFOs," page 23 for more information.

Free-Running Simulation with Lockable Shared Memory

This implementation also contains a Shared Memory interface between the top-level design blocks and the logic that is implemented in hardware. The data is passed over the Shared Memory interface in a frame-based manner and the Hardware Co-Simulation token is clocked by a free-running clock. The Shared Memory blocks are configured as Lockable Shared Memory. See "Free-Running Simulation with Lockable Shared Memories," page 25 for more information.

Comparison of Implementations

A comparison of the results obtained for each implementation method is provided to show which method may produce the best results for different simulation times. Results are compared at three different simulation runtimes: 1,000, 10,000 and 100,000. This is followed by a comparison of the three fastest hardware implementations over very long simulation runtimes.

Short Runtime Comparison: 1000s

For short simulation times, comparison shows that a software implementation outperforms the hardware implementations. This is expected, as the Hardware Co-Simulation implementations must initialize the hardware (configure FPGAs, etc.) as indicated by the *Sysgen: Initialize Simulation* portions of Figure 7. This is not necessary for any software-only implementations.

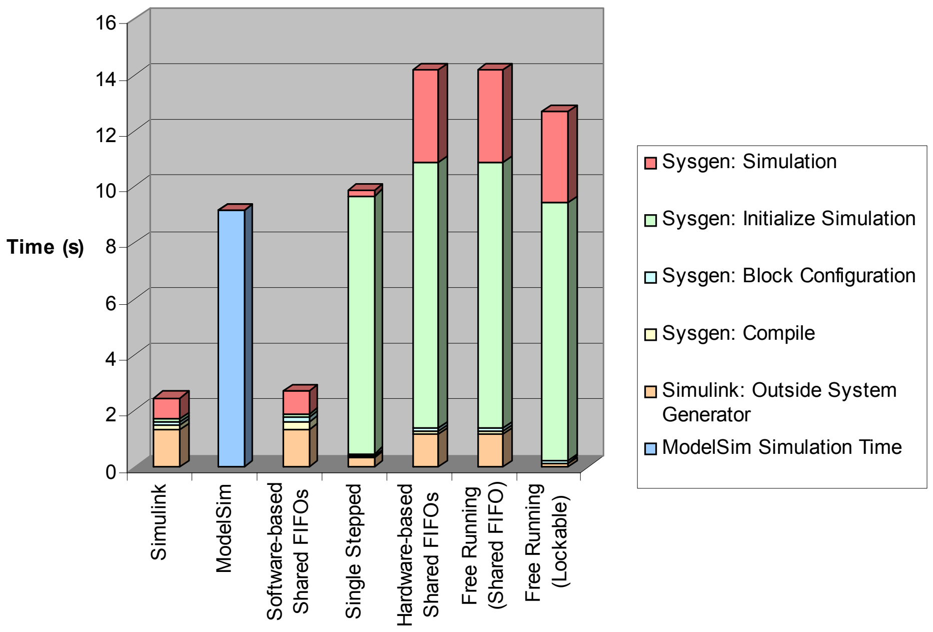


Figure 7: **Simulation Runtime Comparison: 1,000s**

Intermediate Runtime Comparison: 10,000s

As shown, the runtimes of software and hardware implementations are fairly close to each other. This is due to the time required to initialize the hardware simulations (program the FPGA, etc.) as that is fairly constant and independent of simulation runtime. This is the major disadvantage of the hardware implementations with a short simulation.

Figure 8 illustrates the intermediate simulation times comparison.

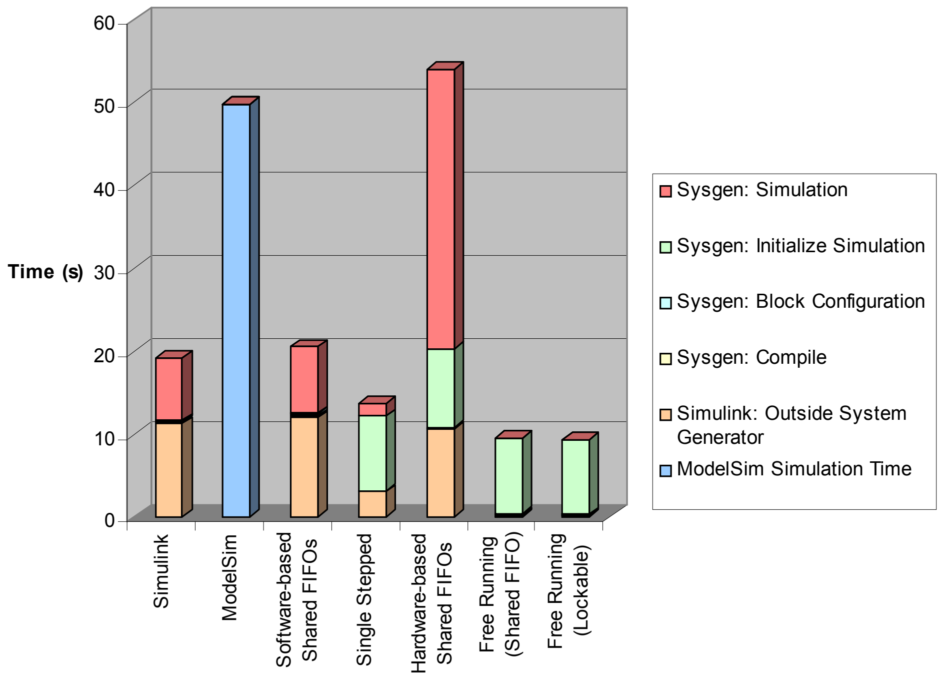


Figure 8: **Simulation Runtime Comparison: 10,000s**

Long Runtime Comparison: 100,000s

Figure 9 shows the longer runtime comparison. As indicated, the Hardware Co-Simulation implementations clearly outperform the software implementations, with the frame-based free-running simulations outperforming a single stepped Hardware Co-Simulation.

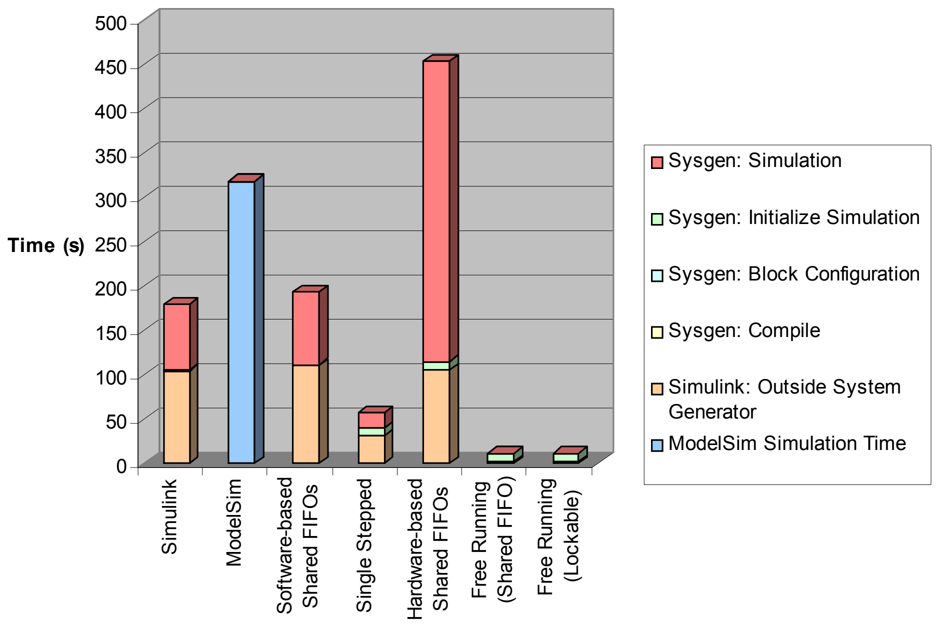


Figure 9: **Simulation Runtime Comparison: 1000,000s**

Very Long Runtimes Comparison

The runtime comparisons shown in Figures 7 through 9 indicate that Hardware Co-Simulation provides the best simulation runtimes as simulation time increases. It also demonstrates that the ratio between the software- and hardware-based approaches increases as simulation time increases. However, these graphs provides limited information about how the Hardware Co-Simulation approaches perform when compared, so a comparison of the three fastest Hardware Co-Simulation approaches at very long runtimes is provided next.

The result of this comparison is shown in Figure 10, and reveals two important things:

1. With small to relatively long simulation times, a simple Hardware Co-Simulation without Shared Memory is comparable to the frame-based methods (Figures 7 through 9). However, as the number of points to be processed increases (which is the case in most systems), and the simulation times get extremely long, the frame-based methods clearly outperform a simple Hardware Co-Simulation without Shared Memory.
2. Of the two frame-based methods, Shared Memories configured as Shared FIFOs provides shorter simulation runtimes than the Lockable Shared Memory approach provides. In addition, the Lockable Shared Memory approach is more difficult to implement and demonstrates less desirable performance in the demonstration model. However, this approach has the advantage of protected memory, which is not the case when Shared Memories are configured as Shared FIFOs. In protected access mode, the System Generator co-simulation hardware must acquire a lock over the shared memory object before it can access its contents.

Figure 10 illustrates the comparison of hardware-based approaches with a simulation runtime set to 2x10^{7}s.

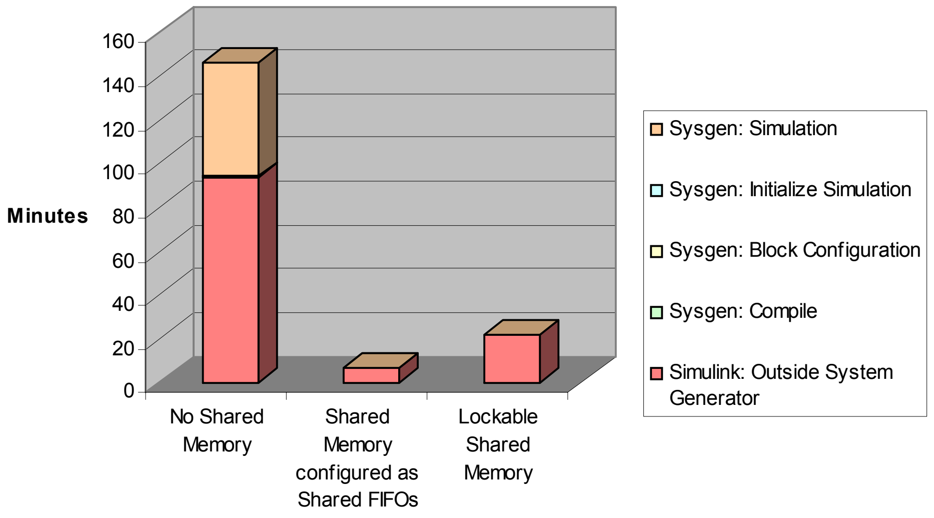


Figure 10: **Simulation Runtime Comparison: Runtime set to 2x10^7 (in seconds)**

Notes:

1. The Y axis of this graph is comprised of minutes and not seconds, as shown in previous graphs.
2. For the hardware simulations shown in Figure 10, a Point-to-Point (P2P) Ethernet 1 Gbps interface was used; however, results are not dependant on the interface used. Thus, if one approach performs better than another approach, it will be the case for all the interfaces—as the analysis does not include the block initialization times.

3. All measured simulation runtimes are based on models that were previously initialized and cached by System Generator for DSP. Because all approaches described contain different blocks, the initialization times also differ. For example, with the P2P Ethernet approach, the amount of time that it takes to create the SystemACE™ configuration file is not considered.
4. While this comparison contrasts the different methods performance, simulation time results are dependent on the specific demonstration design used.
5. The exact numbers used to plot the graphs are provided in the tables in "Appendix A: Detailed Implementation Information," page 18. These numbers are all measured by System Generator for DSP and are printed in the `model_name_sysgen.log` file upon termination of the simulation. This file is placed in the directory which Matlab is pointing to and should be the directory containing the model. The times listed in the tables were recorded after each model was initialized, so the times are longer when a simulation is run for the first time. This is because the design must be cached by System Generator for DSP. In the case of the P2P Ethernet Hardware Co-Simulations, SystemACE configuration files are also created if needed. To achieve the times listed in the tables, the model should be run a second time with the SystemACE files already present in the appropriate directory.

Simulation Runtime Influences

As shown in "Different Implementation Methods," page 7, the approach taken when implementing a design greatly influences the simulation runtime of the design. Additional factors affecting Hardware Co-Simulation performance, apart from the implementation method, can also influence simulation runtime, and are described in this section.

Interfaces

System Generator for DSP provides a number of interfaces for Hardware Co-Simulation, each resulting in a different simulation runtime. The demonstration model is again used as a basis to analyze and compare each interface. For this analysis, the filter part of the design (Figure 5) is compiled and a single stepped Hardware Co-Simulation is performed, similar to that shown in Figure 20, with the interface being changed for comparison.

The available interfaces on the ML402 development board are compared in Table 1 and plotted in Figure 11.

Table 1: **Interface Comparison Using ML402 Development Board**

Simulink simulation cycles = 100 000	JTAG: Platform USB (12 MHz)	JTAG: Parallel IV (5 MHz)	P2P Ethernet 10 Mbps	P2P Ethernet 100 Mbps	P2P Ethernet 1 Gbps
Simulink: Outside System Generator	29.88	30.42	127.74	28.99	28.57
Sysgen: Block Configuration	0.09	0.08	0.19	0.09	0.09
Sysgen: Compile	0.04	0.04	0.31	0.04	0.05
Sysgen: Initialize Simulation	3.66	4.98	0.10	9.08	8.94
Sysgen: Simulation	117.53	41.14	86.06	17.69	15.68
TOTAL	151.2	76.67	214.41	55.89	53.33

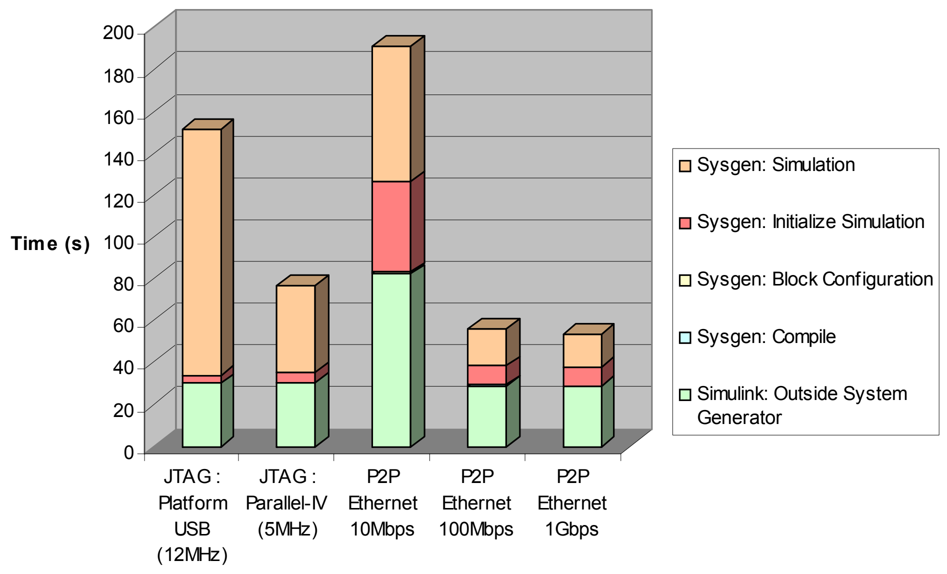


Figure 11: **Interface Performance: ML402 Evaluation Platform**

XtremeDSP Development Kit-IV Interface Comparison

Table 2 and Figure 12 provide a comparison of the different interfaces available on the XtremeDSP Development Kit-IV.

Table 2: **Interface Comparison Using XtremeDSP Development Kit-IV**

Simulink simulation cycles = 100 000	JTAG: Platform USB (3 MHz)	JTAG: Parallel IV (5 MHz)	PCI
Simulink: Outside System Generator	33.08	28.77	28.13
Sysgen: Block Configuration	0.09	0.09	0.09
Sysgen: Compile	0.04	0.04	0.05
Sysgen: Initialize Simulation	10.93	7.92	3.45
Sysgen: Simulation	130.93	40.27	4.56
TOTAL	177.97	77.09	36.28

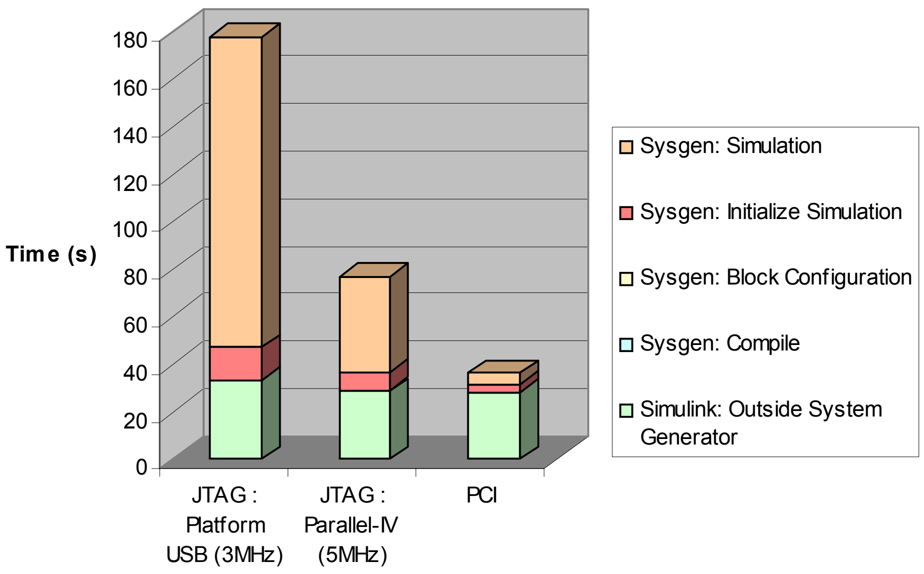


Figure 12: **Interface Performance - XtremeDSP Development Kit-IV**

Notes

1. The USB cable interface on the XtremeDSP Development Kit-IV can only be used with a free-running clock, and is not included in this comparison.
2. A network-based Ethernet interface is not analyzed here, but the expected performance will be between a JTAG interface and a P2P Ethernet 10 Mbps interface.
3. Although JTAG simulation is not the fastest interface, it can be used to support numerous new platforms. See the "Supporting New Platforms through JTAG Hardware Co-Simulation" section of the *System Generator for DSP User Guide* [Ref 2].
4. See [Ref 4] for a Hardware Co-Simulation interface comparison using a different model.

Number of Ports

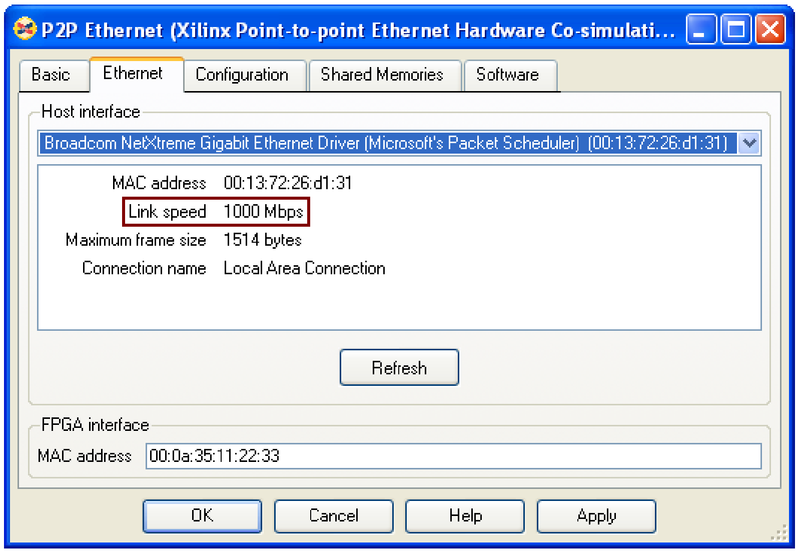
The number of ports on a Hardware Co-Simulation block can greatly influence simulation runtime, due to the amount of time required to multiplex these ports into a single bus to transfer data to the FPGA. To increase performance, you can either minimize the number of input/output ports, or manually concatenate the data into a single bus. Concatenating ports should not result in a bus width exceeding 32 bits. If the bus becomes wider than 32 bits it would be preferable to create multiple 32-bit wide busses.

NIC Settings

The Network Interface Card (NIC) features can also affect the runtime of a Hardware Co-Simulation, as described in this section.

Performance Available (10 Mbps, 100 Mbps, 1 Gbps)

To verify the speed used by System Generator for DSP during a P2P Ethernet Hardware Co-Simulation, connect the PC and the board using an Ethernet cable. The correct speed appears in the Hardware Co-Simulation token properties window, shown in Figure 13.


P2P Ethernet (Xilinx Point-to-point Ethernet Hardware Co-simulati...

Broadcom NetXtreme Gigabit Ethernet Driver (Microsoft's Packet Scheduler) (00:13:72:26:d1:31)

MAC address 00:13:72:26:d1:31
Link speed 1000 Mbps
Maximum frame size 1514 bytes
Connection name Local Area Connection

MAC address 00:0a:35:11:22:33

Figure 13: **Link Speed Verification: P2P Ethernet**

Jumbo Frame Support

When enabled, Jumbo Frame Support can increase simulation performance. To verify that a card can support Jumbo Frames on a Windows platform, you can view the properties for the card in the Device Manager tool. After opening the Device Manager, click on the Advanced tab. A property for Jumbo Frames should be present. You can then enable or disable this feature.

Simulation Runtime

The amount of data being processed is directly related to the simulation runtime. As shown in "Different Implementation Methods," page 7, some approaches perform well at short simulations and not as well at long simulations. In contrast, some approaches, such as frame-based free-running, perform much better during long simulation and not as well during short simulations. When the amount of data to be processed is relatively small, a single-stepped simulation may outperform a free-running simulation.

Depth of Shared Memory

In a frame-based, free-running Hardware Co-Simulation, frame size and the depth of the Shared Memories used can positively influence the runtime of a simulation when chosen correctly. Because a frame of data is sent to the logic implemented in hardware when the Shared Memories are filled, bigger shared memories (up to a certain point which is design-dependant) can provide better performance, as it reduces the number of times data is transferred between Simulink and the FPGA.

Available PC Resources

Apart from the physical hardware configuration of the PC used, the number of processes running can also influence simulation runtime. Running a simulation with the Simulink Profiler enabled will take considerably longer than when the Simulink Profiler is disabled.

Hardware Platform

The hardware platform on which the Hardware Co-Simulation is implemented can affect the simulation in these ways:

- The available interfaces will affect runtimes.
- The onboard oscillator can affect the simulation runtime depending on the design used for a free-running simulation, as this clock is used to drive the design.

Notes for Higher Performance FPGA Design

See the "Notes for Higher Performance FPGA Design" section of the *System Generator for DSP User Guide* [Ref 2] for design tips that can help to increase design performance.

Conclusion

This application note presented a performance-focused overview of the Hardware Co-Simulation capabilities of System Generator for DSP. The analysis of different implementation approaches have shown that, depending on the simulation runtime, certain approaches clearly outperform other approaches. In summary:

- For short simulations a software approach is the best option
- For long simulations a hardware approach is the best option

When comparing hardware implementations, a frame-based implementation performs significantly better than a non-frame-based implementation. As the simulation runtime increases, the ratio between the different implementations also increases, as shown in Figure 14.

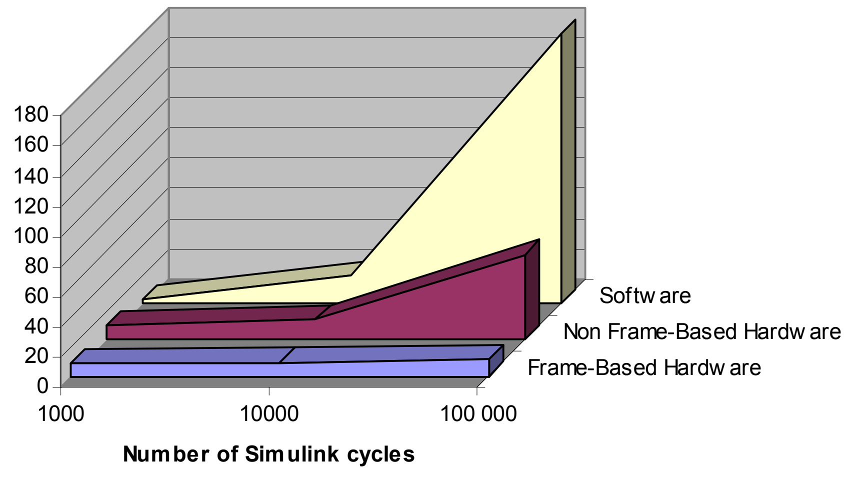


Figure 14: **Performance Gap - Software vs. Hardware Implementations**

Consider the target application and requirements of the simulation when choosing your implementation approach. The "Different Implementation Methods," page 7 section of this document presented information to assist you in selecting the best approach for your design. After determining the approach, see "Appendix A: Detailed Implementation Information" for more detailed information.

In addition to the implementation method, other factors can affect performance, most notably the interface used. These factors are described in "Simulation Runtime Influences," page 12.

Reference Design Files

The reference design files are available for download from the Xilinx website.

The ZIP file contains the structure and content shown in Figure 15. Each implementation directory contains ready-to-use files required to run the simulation for each specific implementation on the ML402 Evaluation Platform. The designs included in the archive are not board- or interface-dependent, and can be easily regenerated to run on any target platform and interface. More information can be found in the included `readme.txt` file.

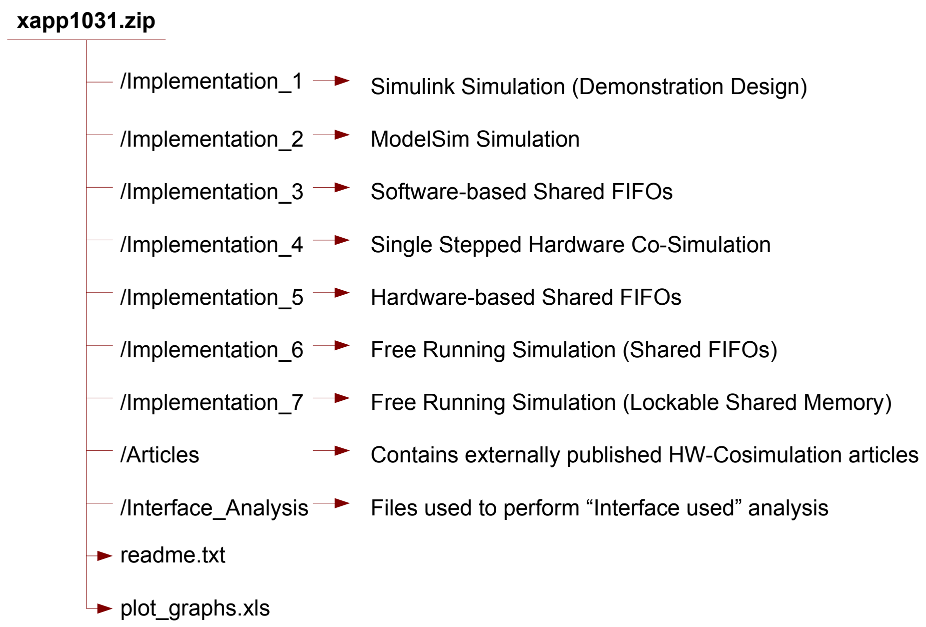

xapp1031.zip
/Implementation_1
Simulink Simulation (Demonstration Design)
/Implementation_2
ModelSim Simulation
/Implementation_3
Software-based Shared FIFOs
/Implementation_4
Single Stepped Hardware Co-Simulation
/Implementation_5
Hardware-based Shared FIFOs
/Implementation_6
Free Running Simulation (Shared FIFOs)
/Implementation_7
Free Running Simulation (Lockable Shared Memory)
/Articles
Contains externally published HW-Cosimulation articles
/Interface_Analysis
Files used to perform “Interface used” analysis
readme.txt
plot_graphs.xls

Figure 15: **File Structure and Contents of ZIP file**

Analysis Test Equipment

This section provides more information about the hardware and software used to obtain the results and performance numbers provided in this document.

Hardware

- Simulations were performed on a system with the following specifications:
  - Microsoft Windows NT platform
  - Microsoft Windows XP Version 5.1 (SP2) operating system
  - Intel based, Pentium Class, Dual Core, ~3192 MHz processor
- The hardware platforms used are the ML402 evaluation platform [Ref 5] and the XtremeDSP Development Kit IV. [Ref 6]
- Broadcom NetXtreme 57xx Gigabit Controller (without support for Jumbo Frames)

Software

- System Generator for DSP v9.2i SP1
- Mentor Graphics® ModelSim® PE v6.2a
- WinPCap v4.0
- MATLAB® 2007a
- ISE v9.2.03i

Appendix A: Detailed Implementation Information

In this Appendix, the implementations of the demonstration design described in "Different Implementation Methods," page 7 are discussed in detail. The description of each implementation contains the following information:

- A short overview of the implementation.
- Where appropriate, figures are provided to illustrate specific implementations.
- An overview of the amount of times that important blocks in the design are called and change state is provided in table format.
- Simulation runtimes are provided in a table format.
- Important notes for each implementation are added at the end of each section.

Simulink Simulation

A normal Simulink simulation is performed on the demonstration model without any modifications, as shown in Figure 5. Table 3 provides the number of times Simulink blocks and System Generator blocks in the demonstration model are called and change state. The results of the runtime analysis of this implementation is shown in Table 4. The data in the Simulation Runtimes tables for each implementation described in the following sections were used to compare the different approaches discussed in "Different Implementation Methods," page 7.

Table 3: **Block Calls and State Changes**

Simulink Simulation Cycles	1,000	10,000	100,000
Simulink Blocks			
Calls Made	1001	10001	100001
State Changes	1001	10001	100001
System Generator for DSP Blocks			
Calls Made	32001	320001	3200001
State Changes	(1)	(1)	(1)

1.The blocks defined with a Sample Period of one will make 1001 state changes. The blocks defined with a Sample Period of 1/32 will make 32001 state changes. For the last two columns these values should be multiplied with the correct factors (10x and 100x, respectively).

Table 4: **Simulation Run Times**

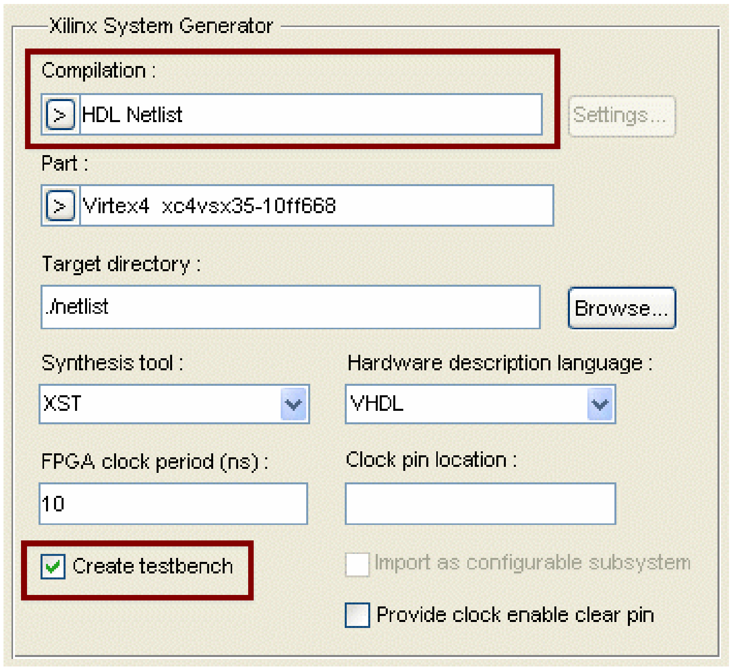
Simulink Simulation Cycles	1.000	10,000	100,000
Simulink: Outside System Generator	1.33	11.26	104.43
Sysgen: Block Configuration	0.12	0.11	0.11
Sysgen: Compile	0.17	0.17	0.18
Sysgen Initialize Simulation	0.09	0.09	0.08
Sysgen: Simulation	0.74	7.48	74.14
TOTAL	2.44	19.11	178.94

Notes:

1. The 32-tap filter multiply-accumulate (MAC) engine is running 32 times faster than the rest of the design. For this reason, the Simulink system period is set to 1/32 in System Generator token. This is important to know when considering the various implementations.
2. The filter is a masked subsystem. Double-click the filter to change filter parameters. To see the filter logic, right-click on the filter block and select **Look Under Mask**.

ModelSim Simulation

The simulation time of a VHDL ModelSim Simulator simulation is analyzed in this section. For these purposes, the model was generated with a HDL Netlist compilation target with the Generate test bench option enabled, as shown in Figure 16.


Xilinx System Generator

Compilation : HDL Netlist

Part : Virtex4 xc4vsx35-10ff668

Target directory : ./netlist

Synthesis tool : XST

Hardware description language : VHDL

FPGA clock period (ns) : 10

Clock pin location :

[✓] Create testbench

[ ] Import as configurable subsystem

[ ] Provide clock enable clear pin

Figure 16: **ModelSim HDL Settings**

After the model is compiled to VHDL, a behavioral simulation can be easily run using the test bench that is generated from the Simulink blocks that drive System Generator for DSP blocks. The clock enable on every individual block is determined by the *sample time* parameter for each block. The behavioral simulation in ModelSim PE gives the expected output for this model, and is shown in Figure 17.

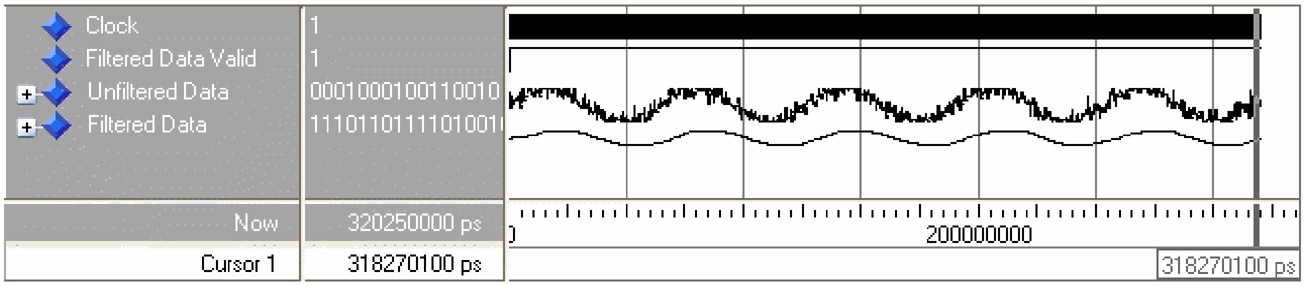


Figure 17: **ModelSim Behavioral Simulation Output**

A DO script can measure the time needed to compile the source files, load the design, and then run the simulation. Note that this does not include the time needed to generate the model in System Generator for DSP which produces the VHDL output.

An example script file is shown below and can be used to measure the times listed in Table 5.

```
vlib work
set compiletime [time {vcom -93 -nowarn 1   demonstration_model.vhd}]
set compiletime [time {vcom  -nowarn 1 demonstration_model_cw.vhd}]
set compiletime [time {vcom -93 -nowarn 1 demonstration_model_tb.vhd}]
set runtime [time {vsim -L work -t ps  demonstration_model_tb}]
view wave
add wave *
view structure
view signals
set NumericStdNoWarnings 1
run 0
set NumericStdNoWarnings 0
set runtime [time {run 3202500.000000 ns}]
```

For the lines in the above script file that do not have compile time or run time commands, the time to perform the line was assumed to be small enough not to be included in the analysis. The VHDL behavioral simulation results are shown in Table 5.

Table 5: **VHDL Behavioral Simulation Run Times**

Simulink Simulation Cycles	1,000	10,000	100,000
Compilation Times			
demonstration_model.vhd	1.69	1.65	1.78
demonstration_model_cw.vhd	0.87	0.87	0.94
demonstration_model_tb.vhd	0.16	0.24	0.18
Loading Design Times			
vsim command	1.47	1.62	1.5
Actual Simulation Times			
run command	4.98	45.34	313.73
TOTAL	9.17	49.72	367.89

As this simulation was done outside the Simulink environment, a table with results on Block Calls and State Changes was not included. A behavioral simulation was used as a benchmark because it uses the Xilinx UniSim Libraries which do not include any timing information that could slow down simulation times.

Software-Based Shared FIFOs

The software simulation of this implementation is detailed in this section. In this approach, the original design was modified to use shared FIFOs to communicate between the top-level and the lower-level filter sub-system. Figure 18 shows the top level of the design and Figure 19 shows the contents of the lower-level filter sub-system. Table 6 contains the simulation runtimes for the Software-based Shared FIFO implementations.

In "Hardware-Based Shared FIFOs," the same design was used for a Hardware Co-Simulation where the communication between the PC and the FPGA was achieved using the shared FIFOs interface.

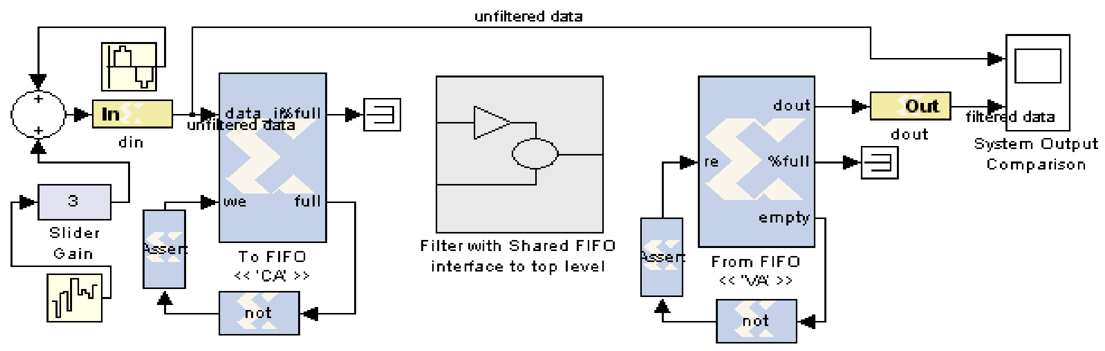


Figure 18: **Shared FIFO - Top Level**

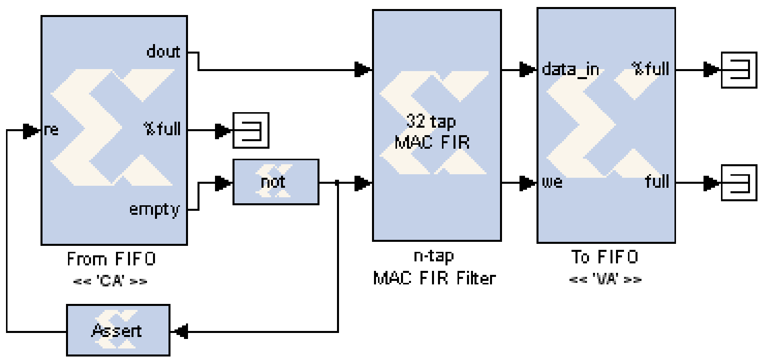


Figure 19: **Shared FIFO - Lower Level**

Table 6: **Simulation Runtimes for Software-based Shared FIFOs**

Simulink Simulation Cycles	1,000	10,000	100,000
Simulink: Outside System Generator	1.32	11.97	110.64
Sysgen: Block Configuration	0.18	0.16	0.17
Sysgen: Compile	0.27	0.27	0.27
Sysgen: Initialize Simulation	0.10	0.09	0.10
Sysgen: Simulation	0.83	8.06	81.51
TOTAL	2.71	20.56	192.69

The Block Calls and State Changes data is the same as Table 3 for this approach. This implementation is an inefficient approach and is provided only for comparison to the hardware implementation of the same approach described in "Hardware-Based Shared FIFOs."

Single-Stepped Hardware Co-Simulation

In the Single-stepped Hardware Co-Simulation approach, System Generator blocks (Figure 5) are compiled into a Hardware Co-Simulation token and a normal P2P Ethernet simulation is analyzed. This design is shown in Figure 20. Simulation runtimes for this approach are summarized in Table 7.

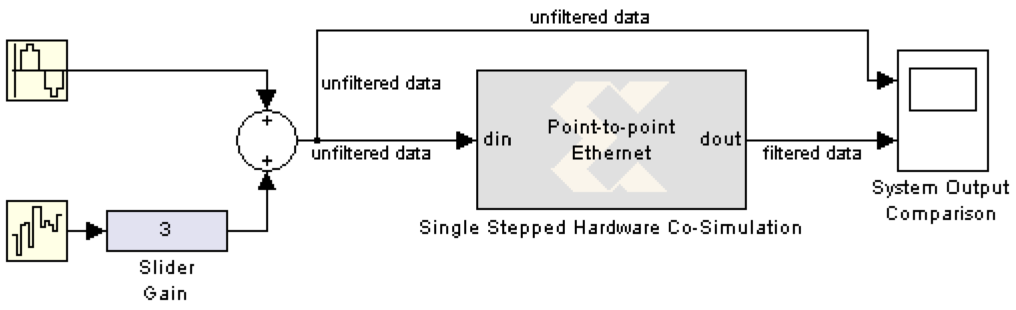


Figure 20: **Single-stepped Hardware Co-Simulation**

This approach does not include any shared memory, and a single-stepped simulation was performed to keep the hardware and software synchronized.

Table 7: **Simulation Run Times - Single-Stepped Hardware Co-Simulation**

Simulink Simulation Cycles	**P2P Ethernet HW-Cosim (1 Gbps)**		
	1,000	**10,000**	**100,000**
Simulink: Outside System Generator	0.33	3.02	30.90
Sysgen: Block Configuration	0.09	0.08	0.09
Sysgen: Compile	0.05	0.05	0.04
Sysgen: Initialize Simulation	9.22	9.02	9.06
Sysgen: Simulation	0.18	1.57	16.70
TOTAL	9.87	13.79	56.79

Notes:

1. The *Block Calls and State Changes* data is the same as is summarized in Table 3 for this approach. It should be noted that the Solver sees and handles the Hardware Co-Simulation token as one block in the design. It does not report any numbers for the logic implemented in hardware.
2. The System Generator clock as defined in Figure 2, page 2 is used as the clock input of the system and is sent to the part of the design running in hardware over the Hardware Co-Simulation interface. The number of state changes of the logic blocks implemented in the hardware is again determined by the *sample time* parameter for each block.
3. This approach is the easiest way to perform a Hardware Co-Simulation and provides adequate simulation times in most designs. This is the best approach to use when verifying a small design in hardware.
4. A free-running simulation in this scenario will not produce proper results because the FPGA and PC are not synchronized.

Hardware-Based Shared FIFOs

This approach is a Hardware Co-Simulation approach of the design described in "Hardware-Based Shared FIFOs," and screen-shots of the design are not repeated here. The lower-level filter sub-system shown in Figure 19 is compiled into a Hardware Co-Simulation token which replaces the sub-system block shown in Figure 18. Table 8 provides the simulation runtimes for the hardware-based shared FIFOs approach.

This approach may seem like it would decrease the simulation time as the filter logic is implemented and runs in hardware. However, when comparing the results of this approach to other implementations, it is much slower because the software must still communicate with the hardware on every clock cycle. The increased overhead of the Shared FIFOs makes this approach significantly slower than any of the other approaches.

Table 8: **Simulation Runtimes - Hardware-based Shared FIFOs**

Simulink Simulation Cycles	**P2P Ethernet HW-Cosim (1 Gbps)**		
	1,000	**10,000**	**100,000**
Simulink: Outside System Generator	1.17	10.56	104.90
Sysgen: Block Configuration	0.10	0.10	0.10
Sysgen: Compile	0.12	0.12	0.12
Sysgen: Initialize Simulation	9.46	9.52	9.83
Sysgen: Simulation	3.32	33.71	337.96
TOTAL	14.17	54.01	452.92

Notes:

1. The Block Calls and State Changes data for this approach is the same as is shown in Table 3. It should be noted that the Solver sees and handles the Hardware Co-Simulation token as one block in the design. It does not report any numbers for the logic implemented in hardware.
2. This approach directly relates to the second timeline shown in Figure 3, page 5, which explains the long simulation runtimes of this approach.

Free-Running Simulation with Shared Memories Configured as Shared FIFOs

The free-running approach is the most important of the approaches presented in this document, as it provides the largest performance gain. The motivation for a free-running frame-based Hardware Co-Simulation was provided in "Frame-Based Data Transfer System Overview," page 4 and is not repeated here. Table 9 and Table 11 contain important summary information for these implementations.

Figure 21 shows the top level of the design with the P2P block added. The lower-level logic shown in Figure 19 is compiled to form the Hardware Co-Simulation token represented in Figure 21.

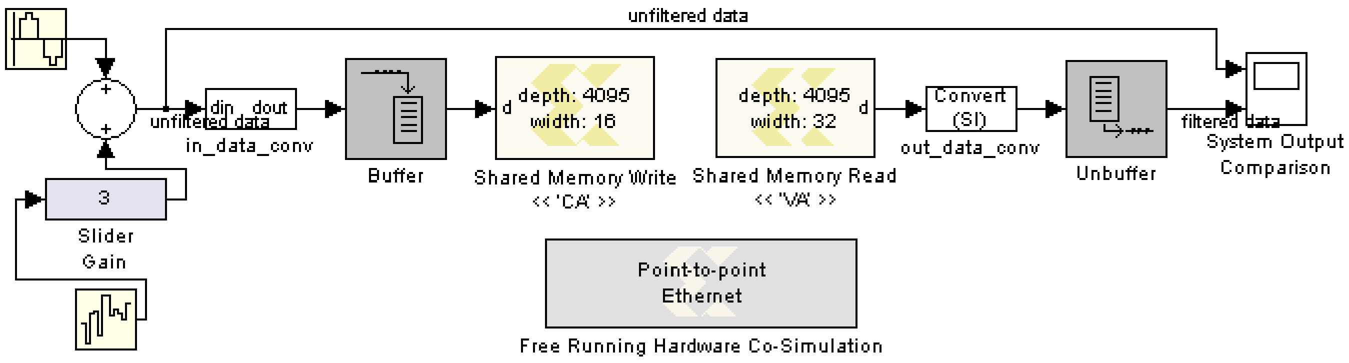


Figure 21: **Hardware Co-Simulation with Shared Memories implemented as Shared FIFOs**

Table 9: **Block Calls and State Changes**

Simulink Simulation Cycles	1,000	10,000	100,000
Simulink Blocks			
Calls Made	1001	10001	100001
State Changes	1001	10001	100001
System Generator for DSP Blocks(2)			
Hardware Co-Simulation Token	(1)	2	11
Shared Memories	(1)	3	25

Notes:

1. A simulation runtime of 1,000 is too small to measure the calls and state changes in this implementation.
2. The numbers for System Generator for DSP blocks are dependent on the depth of the Shared Memories and associated buffers in the design. In this implementation the numbers for System Generator for DSP blocks represent the block calls. The state changes will be dependant on the *Sample Time* parameter for each individual blocks, as shown in Figure 2, page 2.

Table 10: **Simulation Runtimes: Free-Running with Shared Memories**

Simulink Simulation Cycles	P2P Ethernet HW-Cosim (1 Gbps)			
	1,000	10,000	100,000	1,000,000
Simulink: Outside System Generator	0.10	0.24	1.78	16.32
Sysgen: Block Configuration	0.09	0.09	0.09	0.09
Sysgen: Compile	0.04	0.04	0.04	0.04
Sysgen: Initialize Simulation	9.21	9.18	9.30	9.65
Sysgen: Simulation	3.25	0.00	0.00	0.00
TOTAL	9.44	9.55	11.20	26.09

Notes:

1. As stated in "Simulink Simulation," page 18 a free-running approach requires shared memory and control logic to perform synchronization between the simulation and the logic running in hardware. As previously stated, FIFOs are perfect for crossing clock domains and are used in this implementation.
2. The data types used in the top level of this design is another important aspect that must be configured correctly for the design to work. Refer to the properties of the blocks in the `in_data_conv` and `out_data_conv` sub-systems of the design for more information. These blocks are necessary because the input to the shared memory blocks must be one of the following: an 8-bit, 16-bit, or 32-bit signed or unsigned integer.
3. This implementation can further be optimized if a full-frame of data is available every time the System Generator for DSP portion of the design is activated. To illustrate, consider a stream of images for processing. If a new image is available on every Simulink clock cycle, a processed image will be available on the Shared Memory Read block on every clock cycle. This implementation is further described in the "Real-Time Signal Processing Using Hardware Co-Simulation" section of the *System Generator for DSP User Guide* [Ref 2]. This is illustrated with the System Generator token logo in Figure 4, page 6.

Free-Running Simulation with Lockable Shared Memories

This implementation uses Shared Memory blocks configured as Lockable Shared Memories. The top level and lower levels of the design used for the filter sub-system are shown in Figures 22 through 24. This approach is different from "Free-Running Simulation with Shared Memories Configured as Shared FIFOs" as the Input and Output buffers are wrapped around the filter subsystem (Figure 24). The `filter_in_hardware` subsystem shown in Figure 23 contains a mask that provides the ability to specify the sample times of the buffers and the filter logic.

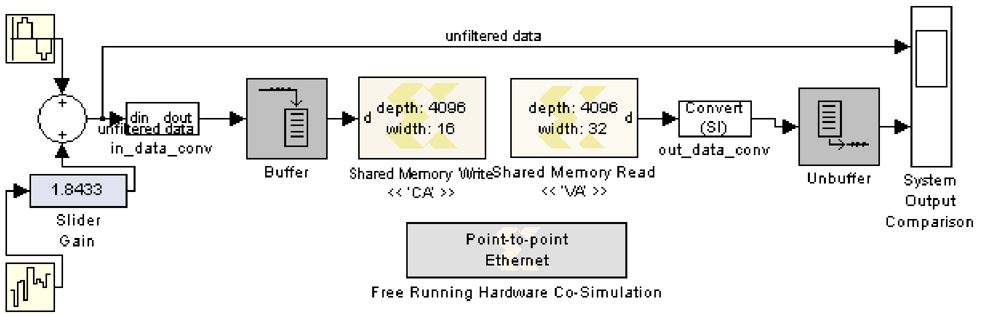


Figure 22: **Lockable Shared Memory Implementation: Top Level**

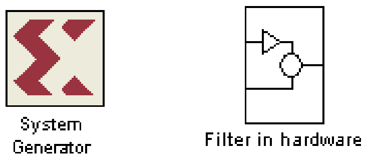


Figure 23: **Filter Sub-system - Top Level**

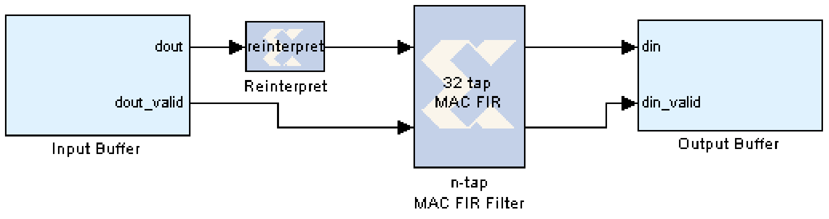


Figure 24: **Filter sub-system - Lower Level**

Table 11: **Block Calls and State Changes**

Simulink Simulation Cycles	1,000	10,000	100,000
Simulink Blocks			
Calls Made	1001	10001	100001
State Changes	1001	10001	100001
System Generator for DSP Blocks(2)			
Hardware Co-Simulation Token	(1)	3	25
Shared Memories	(1)	3	25

Table 11 Notes:

1. A simulation runtime of 1,000 is too small to measure the calls and state changes in this implementation.
2. The numbers for System Generator for DSP blocks are dependent on the depth of the Shared Memories and associated buffers in the design. In this implementation the numbers for System Generator for DSP blocks represent the block calls. The state changes will be dependant on the *Sample Time* parameter for each individual block.

Table 12: **Simulation Runtimes: Free-Running with Lockable Shared Memories**

Simulink Simulation Cycles	P2P Ethernet HW-Cosim (1 Gbps)			
	1,000	10,000	100,000	1,000,000
Simulink: Outside System Generator	0.10	0.22	1.85	25.84
Sysgen: Block Configuration	0.08	0.07	0.07	0.08
Sysgen: Compile	0.04	0.04	0.04	0.04
Sysgen: Initialize Simulation	8.95	8.99	9.02	9.04
Sysgen: Simulation	0.00	0.00	0.06	0.55
TOTAL	9.17	9.33	11.04	35.56

Notes:

1. This implementation uses the Input Buffer and Output Buffer blocks presented in the "Real-Time Signal Processing using Hardware Co-Simulation" section of the *System Generator for DSP User Guide* [Ref 2]. These blocks contain all the control logic needed to cross clock domains in the free-running implementation.
2. The filter subsystem compiled to run in hardware (Figure 24) shows that a Reinterpret block was inserted between the Input buffer block and the filter. This is an important block because the filter mask requires a specific input format for the filter to work properly. In the demonstration model, the mask specifies the input to be of type `Fix_16_15`. The output of the Input buffer is `UFix_16_0` (by default). This was not a problem in any of the previously described implementations because the interfaces between the top-level and lower-level filter subsystems specified the data type given to the filter as `Fix_16_15`.
3. The sample rates are very important in this design and should be chosen carefully for the system to work properly. Figure 25 illustrates the sample times used in this implementation.

   The System Generator for DSP blocks in the top level (the two Shared Memory blocks), have a *sample time* parameter of 4096. This parameter is defined in the System Generator token and corresponds to the depth of the Simulink buffers and Shared Memory blocks. The lower-level system is clocked at 4096/32, which is the smallest rate in the lower-level design. A larger rate will not work because all System Generator blocks must be integer multiples of the period defined in the System Generator token. When the size of the buffers and Shared Memories used in the design are changed, the sample rates throughout the design should be changed accordingly.
4. The data types used in the top level of this design are another important aspect that must be configured correctly. See the properties of the blocks in the `in_data_conv` and `out_data_conv` sub-systems of the design for more information. These blocks are necessary because the input to the shared memory blocks must be one of the following: 8-bit, 16-bit, or 32-bit signed or unsigned integer.
5. Observing the output of this system shows that the results do not appear immediately when the simulation is run, which was the case for all the previous implementations. This is due to the fact that the buffer should fill within the first 4096 simulation steps. The Shared Memory Read block will only be sampled after another 4096 steps, and the correct output starts to appear after 8192 samples.

6. This implementation can also be further optimized if a full frame of data is available on every Simulink clock cycle (step) as described in "Free-Running Simulation with Shared Memories Configured as Shared FIFOs," page 23.

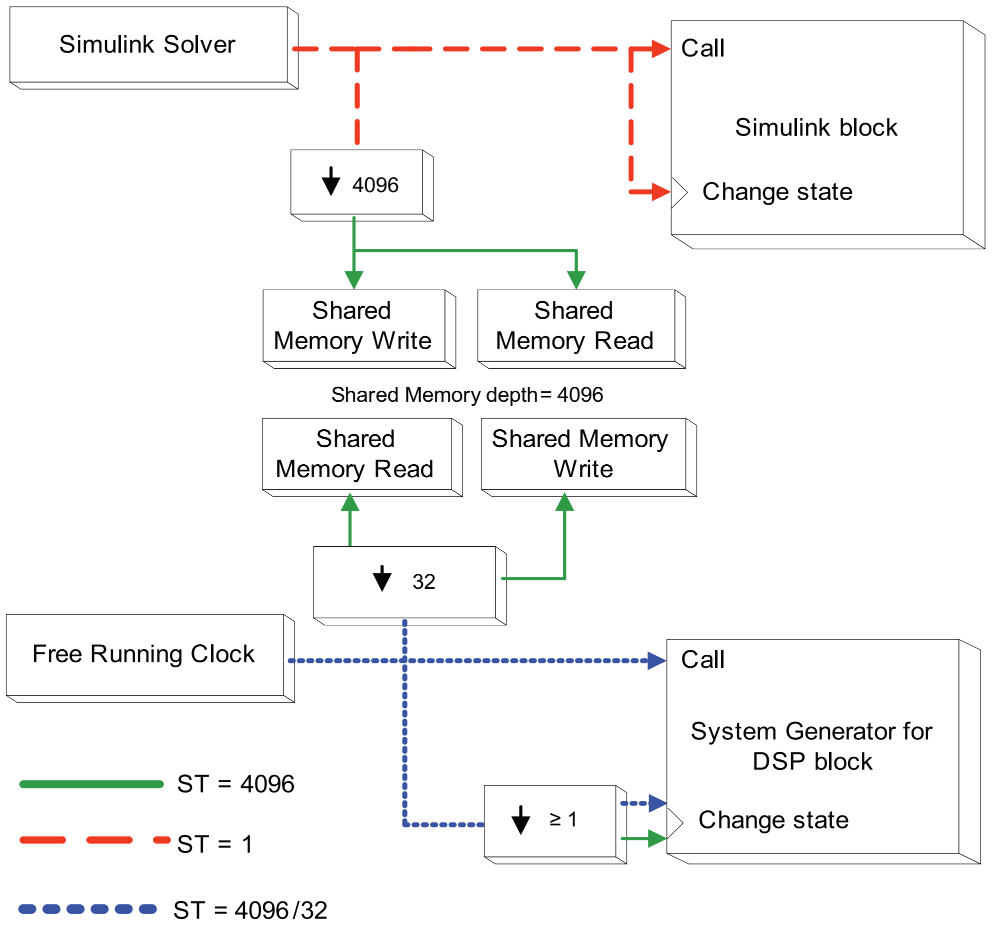


Figure 25: **Clocking Overview - Lockable Shared Memory Implementation**

Getting More Information

The information in this document is presented in graphs to provide a good overview of the performance of the different methods and implementations for those with a basic understanding of Hardware Co-Simulation.

If you are unfamiliar with Hardware Co-Simulation and would like more information about the methods presented in this document, reading through the following topics in the "Using Hardware Co-Simulation" section of the *System Generator for DSP User Guide* [Ref 2] may be helpful.

- Hardware Co-Simulation Blocks
- Hardware Co-Simulation Clocking
- Shared Memory Support
- Ethernet Hardware Co-Simulation
- Frame-Based Acceleration using Hardware Co-Simulation
- Real-Time Signal Processing using Hardware Co-Simulation
- Installing an ML402 Board for Ethernet Hardware Co-Simulation

References

1. Simulink Solvers Description.
2. *System Generator for DSP User Guide*.
3. Xilinx Application Note 948: Hardware Acceleration of 3GPP Turbo Encoder/Decoder BER Measurements Using System Generator.
4. Ben Chan, Nabeel Shirazi, and Jonathan Ballagh. Achieving High-Bandwidth DSP Simulations Using Ethernet Hardware-in-the-Loop, *DSP Magazine*, Issue 2, May 2006. This article is also included in the Reference Design Files downloadable archive.
5. ML402 Evaluation Platform Documentation.
6. XtremeDSP Development Kit IV.

Revision History

The following table shows the revision history for this document:

Date	Version	Description of Revisions
11/30/07	1.0	Initial Xilinx release.
12/19/07	1.0.1	Added missing table.

Notice of Disclaimer