Xilinx Design Flow for Intel FPGA and SoC Users

User Guide

UG1192 (v2.2) February 9, 2018

XILINX
ALL PROGRAMMABLE™

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
02/09/2018	2.2	Updated Table 1-1. Removed obsolete video links for Customizing and Instantiating IP, Programming and Debugging Design in Hardware, Inserting Debug Cores into the Design, and Debugging Remotely Using Vivado.
05/12/2017	2.1	Added Spartan-7 FPGA and Zynq-7000 AP SoC single-core information throughout document. Added note to Table 2-2.
11/25/2015	2.0	Added Zynq-7000 AP SoC, Artix-7 FPGA, Cyclone V FPGA, and Cyclone V SoC FPGA information to Chapter 2, Architecture Analysis. Added Chapter 4, SoC Conversion. Reformatted Chapter 6, RTL Conversion and added a Recommendation following Figure 6-1.
09/30/2015	1.0	Initial Xilinx release.

Table of Contents

Introduction

Overview

This document highlights some of the differences between Xilinx® and Intel® FPGA and SoC architectures, as well as some differences between the design flows in the Vivado® Design Suite and Quartus® II applications. This guide is designed to provide you with a general understanding of IP core conversion, RTL conversion, along with some of the differences between synthesis, implementation, and verification steps and techniques. Once you are armed with this information, you can begin targeting Xilinx FPGAs and SoCs for your current or next design cycle.

The Vivado Design Suite from Xilinx is an industry-leading solution for Xilinx FPGA and SoC devices including 7 series FPGAs, Zynq®-7000 All Programmable (AP) SoCs, and UltraScale™ architecture devices, which apply leading-edge ASIC techniques to a fully-programmable architecture. If you are currently using the Intel Quartus II Design tools for FPGAs and SoCs, you are a few short steps away from working with Xilinx FPGAs and SoCs using the Vivado Integrated Design Suite.

Additionally, only Xilinx provides:

- UltraFast™ Design Methodology, which is a comprehensive set of design methodologies focused on maximizing system performance, reducing risk, and enabling accelerated and predictable design cycles.
- Vivado High Level Synthesis (HLS), which accelerates IP creation by enabling C, C++, and System C specifications to be targeted to Xilinx All Programmable devices without the need to create RTL.

Product Selection

This section describes the Xilinx product portfolio for high-end, mid-range, and low-range FPGAS, and programmable SoCs. Xilinx offers the broadest multi-node portfolio of All Programmable (AP) FPGA and SoC products with an extensive range of density, performance, power and temperature grade options to meet the requirements of virtually any application. As detailed in Chapter 2, Architecture Analysis, comparison with Intel devices and comprehensive resources are available to aid in device selection. For direct assistance in the selection process, contact Xilinx, partner FAEs, or your sales person.

Product Portfolio

For new designs, Xilinx recommends selecting a device from the product portfolio listed in Table 1-1. The key features and specifications for each family are summarized. Refer to the family product selector guides and data sheets for a complete list of device capabilities.

Table 1-1: **Xilinx Product Portfolio**

High-end FPGAs	**Virtex® UltraScale+™ FPGA** (Product Selector Guide)			
	• 16 nm FinFET+ process • 100G EMAC w/RSFEC	• 862K - 2.9M LCs • 150G Interlaken	• DDR4-2,666 Mb/s • Footprint migration from Virtex UltraScale FPGA	• Max GTs: 128@33 Gb/s • Up to 432 Mb UltraRAM memory
	Virtex UltraScale™ FPGA (Product Selector Guide)			
	• 20 nm process	• 783K - 5.5M LCs	• DDR4-2,400 Mb/s	• Max GTs: 60@30.5/ 60@16.3 Gb/s
	• 100G EMAC	• 150G Interlaken	• VCXO/fractional PLL integration	
	Virtex-7 FPGA (Product Selector Guide)			
	• 28 nm process	• 326K - 2M LCs	• DDR3-1,866 Mb/s	• Max GTs: 96@13.1 /16@28 Gb/s
	• AMS integration			
Mid-range FPGAs	**Kintex® UltraScale+ FPGA** (Product Selector Guide)			
	• 16nm FinFET+ Process	• 356K - 1.1M LCs	• DDR4-2,666 Mb/s	• Max GTs: 32@33 /44@16.3 Gb/s
	• 100G EMAC w/RSFEC	• 150G Interlaken	• VCXO/fractional PLL integration	• Up to 31.5 Mb UltraRAM memory
	Kintex UltraScale FPGA (Product Selector Guide)			
	• 20 nm process	• 318K - 1.5M LCs	• DDR4-2,400 Mb/s	• Max GTs: 64@16.3 Gb/s
	• 100G EMAC	• 150G Interlaken	• VCXO integration	
	Kintex-7 FPGA (Product Selector Guide)			
	• 28 nm process	• 66K - 478K LCs	• DDR3-1,866 Mb/s	• Max GTs: 32@12.5 Gb/s
	• AMS integration			

Table 1-1: **Xilinx Product Portfolio *(Cont'd)***

Cost-optimized FPGAs	**Artix®-7 FPGA** (Product Selector Guide)			
	• 28 nm process	• 13K - 215K LCs	• DDR3-1,066 Mb/s	• Max GTs: 16@6.6 Gb/s
	• AMS integration			
	Spartan®-7 FPGA (Product Selector Guide)			
	• 28 nm process	• 6K - 102K LCs	• DDR3-800 Mb/s	• N/A
	• AMS integration			
Programmable SoCs	**Zynq UltraScale+ MPSoC** (Product Selector Guide)			
	• 16 nm FinFET+ process	• 103K-1.1M LCs	• Quad ARM® Cortex™-A53	• Max GTs: 28@33/44@16.3 Gb/s
	• Dual ARM Cortex-R5	• Power islands and domains	• ARM Mali-400MP GPU / H.265 video Codec	• DDR4/LPDDR4 support
	Zynq-7000 AP SoC (Product Selector Guide)			
	Single-Core			
	• 28 nm process	• 23K - 444K LCs	• Single ARM Cortex-A9	• Max GTs: 4@6.25 Gb/s
	• AMS integration	• Power domains	• DDR3 / LPDDR2 support	
	Dual-Core			
	• 28 nm process	• 28K - 444K LCs	• Dual ARM Cortex-A9	• Max GTs: 16@12.5 Gb/s
	• AMS integration	• Power domains	• DDR3 / LPDDR2 support	

Device Performance Options

The Xilinx FPGA and SoC devices are typically offered in three speed grades to meet the performance needs of a broad range of applications. For all device families listed in Table 1-1, device speed grades are assigned as -1 -2, and -3. For Xilinx devices, -3 is the fastest speed grade. Conversely, for Intel devices, a lower number speed grade indicates a faster device. Speed grades are generally separated by 10-15% in core fabric performance and the operating frequencies of individual device blocks (e.g., ARM processors, block RAM, DSP, transceivers) typically scale with the speed grade as well.

Device Power Options

The Xilinx FPGA and SoC portfolio is optimized for the best performance per watt characteristics. For applications requiring additional power reduction beyond the standard device offerings, multiple device power options are available. All product families listed in Table 1-1 offer devices with substantially lower static power, while maintaining the same level of performance. Additionally, select devices allow a lower operating voltage to significantly reduce both static and dynamic power. The Xilinx power estimator (XPE) can be used to calculate power reductions asSoCiated with these device power options. The XPE can be downloaded from www.xilinx.com/power.

Device Temperature Grades

The Xilinx FPGA and SoC devices are typically offered in commercial (Tj = 0 to 85C), extended (Tj = 0 to 100C), and industrial (Tj = –40 to 100C) temperature grades. Spartan-7 devices have -40 to +125C temp range on commercial devices. Specific devices or speed grades are only available in a subset of temperature grades (see the specific family product selector guide and data sheet for details). Some device families listed in Table 1-1 also offer Xilinx automotive (XA) or defense (XQ) grade devices with additional temperature grades. Automotive grade devices support Q-grade (Tj = –40 to 125C), and defense grade devices support military (Tj = –55C to 125C) temperature grades.

Architecture Analysis

Overview

This chapter compares the Xilinx Artix®-7, Zynq®-7000 SoC, Kintex® UltraScale+™, Virtex® UltraScale+, Kintex UltraScale™, and Virtex UltraScale devices to the Intel Cyclone® V, Cyclone V SoC, Arria 10®, and Stratix® 10 devices, in these areas:

- I/O
- Transceivers
- Clocking (MMCM, BUFG)
- Internal Memory Resources
- DSP
- External Memory Interfaces
- Processing System (PS)
- Analog Mixed Signal
- Board Design

I/O

When converting a design from Intel to Xilinx, the designer needs to be aware of the bank voltages, constraints, and restrictions for general purpose I/O (GPIO) banks.

For details on I/O requirements, availability, and pinouts for the applicable Xilinx device, see:

- *UltraScale Architecture SelectIO Resources Guide* (UG571) [Ref 1]
- *Kintex UltraScale and Virtex UltraScale FPGAs Packaging and Pinouts Product Specification User* Guide (UG575) [Ref 6]
- Chapter 4, "SelectIO Signaling" in the *UltraScale Architecture PCB Design User Guide* (UG583) [Ref 7]

- *Zynq-7000 All Programmable SoC Packaging and Pinout Product Specification* (UG865) [Ref 31]
- *7 Series FPGAs SelectIO Resources User Guide* (UG471) [Ref 32]
- *7 Series FPGAs Packaging and Pinout Product Specification* (UG475) [Ref 33]

IMPORTANT: *These documents are accessible from xilinx.com or the Xilinx Documentation Navigator.*

Table 2-1 shows a comparison of common I/O features for Xilinx UltraScale FPGAs and Intel Arria 10 and Stratix 10 FPGAs.

Table 2-1: **Xilinx UltraScale FPGA and Intel Arria 10 and Stratix 10 FPGAs GPIO Comparison**

Feature	Xilinx UltraScale FPGA	Intel Arria 10 and Stratix 10 FPGAs	Notes
Clock inputs	GCLK	GCLK/RCLK/PCLK	Global clocking dedicated pins.
Bank voltages	HP banks (1V - 1.8V) HR banks (1.2V - 3.3V) HD banks (1.2V - 3.3V)	GPIO banks (1.2V - 1.8V) HV banks (2.5V or 3.3V)	Devices have compatible bank voltages.
Bank size	52 I/O total per bank HR (52 - 156) HP (208 - 706)	48 I/O total per bank LVDS I/O (148 - 768) 3V I/O (0 - 96)	I/O bank size varies between devices. High-performance and high-range banks are similar.
LVDS on-chip termination	DIFF_TERM 100Ω differential termination	RD OCT 100Ω differential termination	LVDS termination is compatible.
Memory on-chip termination	DCI (VRP pin 240Ω to ground)	OCT RZQ 240Ω to ground	Compatible termination for memory support.
Bank application support	HD banks: I2C, UART, SPI, RGMII, legacy interfaces HP banks: LVDS and DDR HR banks: LVDS, DDR and legacy interfaces	HV banks: I2C, UART, SPI, RGMII, legacy interfaces GPIO banks: LVDS and DDR	HD banks replace HR banks in UltraScale+ FPGAs.
Equalization	Transmitter pre-emphasis Receiver equalization	LVDS I/O - Pre-emphasis	Only Xilinx devices support linear receiver equalization on V_{ref} based receivers.

Table 2-2 shows a comparison of common I/O features for Xilinx Artix-7/Spartan-7 FPGAs and Zynq-7000 AP SoC and Intel Cyclone V FPGA and Cyclone V SoC FPGA.

Table 2-2: **Xilinx Artix-7/Spartan-7 FPGA and Zynq-7000 AP SoC and Intel Cyclone V FPGA and Cyclone V SoC FPGA GPIO Comparison**

Feature	Xilinx Artix-7/Spartan-7 FPGA and Zynq-7000 AP SoC(1)	Intel Cyclone V FPGA and Cyclone V SoC FPGA	Notes
Bank voltages	HP banks (1.2V - 1.8V) HR banks (1.2V - 3.3V)	GPIO banks (1.2V - 3.3V)	Devices have compatible bank voltages.
User I/O	I/O per bank (50) (see Xilinx pin planner) 230–530 total pins	I/O per bank (14–80) (see Intel chip planner) 208–560 total pins	I/O bank size varies between devices. Pin planning tools help with the conversion.
LVDS pairs	LVDS full duplex bank support	Dedicated LVDS TX only or RX only bank	Xilinx provides LVDS bank flexibility.
LVDS on-chip termination	OCT DIFF_TERM 100Ω differential termination	RD OCT 100Ω differential termination	LVDS termination is compatible.
Termination	(HP) DCI (HR) Untuned termination	OCT only on memory banks	Xilinx DCI cascade saves pins and provides additional termination.
Bank application support	HP banks: LVDS and DDR HR banks: LVDS, DDR, and legacy interfaces	LVDS and DDR	Similar bank application support.

Notes:

1. Artix-7 and Spartan-7 FPGAs only have HR banks.

Vivado I/O Pin Planner

The Vivado® I/O Pin Planner helps to constrain general purpose I/O. The pin planner provides designers with die and package visibility to ensure that optimal pin planning is achieved the first time. A design can be created just for pin planning to determine I/O compatibility across devices, analyze SSO, manage package trace delays, and optimize placement based on intended data flow.

On-Chip Termination

For UltraScale devices, Xilinx uses digitally controlled impedance (DCI) to provide flexibility for termination. DCI uses a dedicated pin (VRP) with a pull-down resistor for tight impedance control.

For Artix-7/Spartan-7 FPGAs and Zynq-7000 AP SoCs, Xilinx uses 3-state digitally controlled impedance (T_DCI) to control the output drive impedance (series termination), provide parallel termination of an input signal to VCCO, or provide split (Thevenin) termination to VCCO/2.

Xilinx also uses an on-chip differential termination resistor of 100Ω for LVDS pairs. Specific termination options are described in "Appendix A" of the *UltraScale Architecture SelectIO Resources Guide* (UG571) [Ref 1] and "Appendix A" of the *7 Series FPGA SelectIO Resources Guide* (UG471) [Ref 32].

Transceivers

Xilinx transceivers are organized in quads that contain four full-duplex SerDes, each with a complete physical medium attachment (PMA) and physical coding sublayer (PCS). Xilinx UltraScale device transceivers are optimized for high speed and low latency. The GTP and GTX transceivers for the Zynq-7000 AP SoC and Artix-7 FPGA provide the most accessible performance in a 28 nm FPGA.

Transceiver Clocking

Transceiver clocking is highly flexible with multiple clock inputs available, as well as up to six phase-locked loops (PLLs) per quad.

For UltraScale devices, each transceiver contains its own ring oscillator PLL, and has access to either of two high-speed LC tank PLLs per quad. The LC tank PLLs also include a fractional PLL mode. UltraScale device transceiver clocking supports multi-rate Ethernet and video applications without needing external circuitry or sacrificing adjacent transceivers. Table 2-3 shows a comparison of Xilinx UltraScale FPGA and Intel Arria 10 FPGA transceiver clocking features.

Table 2-3: **Xilinx and Intel Transceiver Clocking Comparison**

Clocking	UltraScale FPGA GTH	UltraScale FPGA GTY	Arria 10 FPGA
Ring PLL density	1:1	1:1	1:3 (lose use of one RX)
Ring line rate	0.5-12.5 Gb/s	0.5-12.5 Gb/s	0.611-10.3125
LC PLL density	1:2	1:2	1:3
LC PLL line rate	1.0-16.3 Gb/s	1.0-32.75 Gb/s	0.611-28.3 Gb/s
TX phase interpolator	7-bit resolution	7-bit resolution	No
fPLL	QPLL support in UltraScale+ FPGA	QPLL support	ATX, fPLL
Reference clock routing	Two dedicated pin, four north/south routes	Two dedicated pin, four north/south routes	One dedicated pin, RX pin, clk network, PLL cascade, global fabric clock input

The Artix-7 FPGA GTP transceiver and the Zynq-7000 AP SoC GTP and GTX transceiver's flexible clocking architecture allows a wide variety of interfaces to run at a higher density than other similar transceivers in the industry. The GTP transceiver has two oscillators per

quad. The GTX transceiver has a dedicated ring oscillator in each transceiver with a high-performance LC tank oscillator that is shared across a quad.

Transceiver Equalization

The UltraScale FPGA GTY and GTH transceivers represent the third generation of fully adaptive receiver equalization. This technology sets the bar for ease of use and continues to be the best in class transceiver for the foreseeable future. The receiver equalizer includes an auto-adapting AGC, CTLE, and DFE (not found in GTP transceivers) that work in tandem to optimize link integrity over the PCB channel, backplane, or cable. The auto-adaptation virtually eliminates the need to hand tune individual channels. Table 2-4 shows a comparison of the Xilinx UltraScale FPGA and Intel Arria 10 FPGA transceiver equalization features.

Table 2-4: **Xilinx UltraScale FPGA and Intel Arria 10 FPGA Transceiver Equalization Comparison**

Transmitter Equalization	UltraScale FPGA GTH	UltraScale FPGA GTY	Arria 10 FPGA
TX FIR	3 taps, 15 dB	3 taps, 15 dB	5 taps, 15 dB
Receiver Equalization			
RX CTLE	15 dB	15 dB	
RX CTLE adaptation	Fully adaptive	Fully adaptive	One time adaptation
RX DFE taps	11 fixed	15 fixed	7 fixed taps, 4 floating
RX DFE adaptation	Fully adaptive	Fully adaptive	Fully adaptive
RX loss @Nyquist (dB)	25 dB+ up to 16.3 Gb/s	25 dB+ up to 28.21 Gb/s	30 dB to 17.4 Gb/s, 15 dB up to 28 Gb/s
Backplane line rate	up to 16.3G	28.21 Gb/s	up to 17.4

Table 2-5 shows a comparison of the Xilinx 7 series and the Intel Cyclone V FPGA transceivers.

Table 2-5: **Xilinx 7 Series and Intel Cyclone V FPGA Transceiver Equalization Comparison**

Transmitter Equalization	7 Series GTP	7 Series GTX	Cyclone V FPGA GX	Cyclone FPGA V GT
TX FIR	3 taps, 15 dB	3 taps, 15 dB	5 taps, 15 dB	5 taps, 15 dB
Receiver Equalization				
RX CTLE	15 dB	15 dB	15 dB	15 dB
RX CTLE adaptation	Fully adaptive	Fully adaptive	One time adaptation	One time adaptation
RX DFE taps	No DFE	5 taps	No DFE	No DFE
RX DFE adaptation	N/A	Fully adaptive	N/A	N/A
RX loss @Nyquist (dB)	15 dB	25 dB+	15 dB	15 dB

Debug/Margin Analysis

The UltraScale FPGA, 7 series FPGA, and Zynq-7000 AP SoC transceivers contain a non-destructive eye scan block to allow for real-time link margin analysis using mission traffic. The integrated bit error ratio test (IBERT) transceiver debug software can be used to analyze transceiver and channel performance, without needing a fully functional FPGA image. For more information on IBERT, see the applicable product guide:

- *IBERT for UltraScale GTH Transceivers LogiCORE IP Product Guide* (PG173) [Ref 2]
- *IBERT for UltraScale GTY Transceivers LogiCORE IP Product Guide* (PG196) [Ref 3]
- *IBERT for 7 Series GTP Transceivers v3.0 Product Guide* (PG133) [Ref 26]
- *IBERT for 7 Series GTX Transceivers v3.0 Product Guide* (PG132) [Ref 27]

Clocking (MMCM, BUFG)

Both Xilinx and Intel architectures use clock generation and management circuitry to bring clock signals into the device, multiply, divide and phase shift clock signals and route clock signals to their destinations.

Architecture

UltraScale and UltraScale+ devices use a new ASIC-style clocking structure that is a uniform matrix of horizontal and vertical clock routing tracks that can connect any clock to any location in the device. The Vivado tools handle clock layout for you and take advantage of the ability to collocate clock sources with asSoCiated clouds of logic. For more information, see the *UltraScale Architecture Clocking Resources User Guide* (UG572) [Ref 4].

The Artix-7/Spartan-7 FPGA and Zynq-7000 AP SoC clock structure is built around a vertical clock spine that runs the height of the device. Global clock buffers are located in the center of the device and can connect clock signals from sources to any location in the device. At the entry to every clock region is a collection of horizontal buffers that drive the clock signals horizontally into the clock region and, ultimately, to the clocked elements. There are 32 global capable clock signals in the Artix-7/Spartan-7 FPGA and Zynq-7000 AP SoC.

The Cyclone V FPGA clocking structure consists of global, regional, and periphery clock networks that cover successively smaller regions of the device. The Cyclone V FPGA has a maximum of 16 global capable clocks. Consequently, when converting a Cyclone V FPGA design to an Artix-7/Spartan-7 FPGA design, there are enough global clocks for a successful conversion.

Inputs

In the UltraScale architecture, external clock signals are brought on chip through global clock inputs or GC pins. There are four GC pin pairs (total of eight pins) in every I/O bank that have direct connection to global clock buffers, mixed-mode clock managers (MMCMs), and PLLs. GC pins can be single-ended or differential and can be configured to any supported I/O standard.

In Artix-7/Spartan-7 FPGAs and Zynq-7000 AP SoCs, external clock signals are brought on chip through clock capable inputs or CCIO pins (general purpose I/O pins should not be used for clock signals). There are four CCIO pin pairs (total of eight pins) in every I/O bank that have a direct connection to global clock buffers, MMCMs, and PLLs. CCIO pins can be single-ended or differential and can be configured to any supported I/O standard.

Phase-locked Loops

In the UltraScale architecture, there is a Clock Management Tile (CMT) asSoCiated with every I/O bank. Each CMT contains one MMCM and two PLLs. In the Xilinx Artix-7/Spartan-7 FPGA and Zynq-7000 AP SoC, there is a CMT asSoCiated with every I/O bank. Each CMT contains one MMCM and one PLL.

The MMCMs serve as frequency synthesizers, jitter filters and clock deskew, and contain seven output clocks, four of which have 180 degree phase shift output, fractional divide outputs and clock input switching circuitry. The MMCMs are primarily for clocking the device logic (CLBs, DSP, RAM etc.). The PLLs are primarily for generating the required clock signals for memory interfaces, but also have connections to the device logic so they can be used as extra clocking resources if additional functionality is required.

Internal Memory Resources

Both Xilinx and Intel architectures contain internal memory resources for storing and buffering data.

Layout

The UltraScale, Artix-7 and Spartan-7, and Zynq-7000 AP SoC devices contain distributed RAM (look-up tables configured as memory blocks) and 36 Kb block RAM. The UltraScale+ devices add 288 Kb UltraRAM blocks to the memory hierarchy.

Distributed Memory

The Xilinx architecture allows a certain percentage of the available LUTs to be configured as small memory blocks for coefficient storage and low capacity buffering. Each LUT can be configured as a 64-bit memory allowing a total of 512 bits per Xilinx CLB. Table 2-6 shows a comparison of distributed memory resources for Xilinx and Intel.

Table 2-6: **Xilinx and Intel Distributed Memory Comparison**

	Xilinx Artix-7/ Spartan-7 FPGA	Xilinx Zynq-7000 AP SoC	Xilinx UltraScale FPGA	Xilinx UltraScale+ FPGA	Intel Cyclone V FPGA	Intel Cyclone V SoC FPGA	Intel Arria 10 FPGA	Intel Stratix 10 FPGA
Memory per LUT	64 bit	64 bit	64 bit	64 bit	64 bit	64 bit	64 bit	64 bit
Maximum capacity	2.9 Mb	6.7 Mb	29 Mb	51 Mb	1.7 Mb	0.6 Mb	12.7 Mb	29 Mb

Block Memory and UltraRAM

The Xilinx block RAM is a 36 Kb block that can be configured as two independent 18 Kb blocks. Intel devices use either M20K or M10K blocks with 20 Kb and 10 Kb capacity, respectively. Unlike the Xilinx block, the M20K and 10K blocks cannot be split.

The UltraScale+ devices introduce a 288 Kb memory element called UltraRAM. UltraRAM provides hundreds of megabits of on-chip SRAM for applications that require more storage than traditional block RAMs can offer. Typical UltraRAM applications include networking and video. For more information, see the *UltraScale Architecture Memory Resources User Guide* (UG573) [Ref 5]. Table 2-7 shows a comparison of memory resources for Xilinx and Intel.

Table 2-7: **Xilinx and Intel Block Memory Comparison**

	Xilinx Artix-7/ Spartan-7 FPGA	Xilinx Zynq-7000 AP SoC	Xilinx UltraScale FPGA	Xilinx UltraScale+ FPGA	Intel Cyclone V FPGA	Intel Cyclone V SoC FPGA	Intel Arria 10 FPGA	Intel Stratix 10 FPGA
Blocks	36 Kb/18 Kb block RAM	36 Kb/18 Kb block RAM	36 Kb/18 Kb block RAM	36 Kb block RAM and 288 Kb UltraRAM	M20K	M20K	M20K	M20K
Maximum capacity	13.1 Mb	27.18 Mb	133 Mb	526 Mb	12.2 Mb	5.1 Mb	53 Mb	229 Mb

DSP

Both Xilinx and Intel architectures contain DSP blocks that can be used for signal processing applications. Both Intel and Xilinx devices include pre-adders and post-adders that can be used for filter applications.

The Intel Cyclone V and Cyclone V SoC devices support 9x9, 18x19, and 27x27 multipliers. The Stratix 10 and Arria 10 devices support 18x19 and 27x27 multipliers. The Xilinx 7 series and UltraScale devices support 25x18 and 27x18 multipliers, respectively.

For larger widths, the Vivado Design Suite infers multiple DSPs with cascade. Xilinx 25x18 multipliers can easily emulate the 9x9 small multipliers from Intel.

External Memory Interfaces

When moving external memory interfaces from Intel to Xilinx there are several key architectural differences. Table 2-8 shows a comparison of the external memory interfaces for the Xilinx UltraScale architecture and the Intel Arria 10 architecture.

Table 2-8: **Xilinx UltraScale FPGA and Intel Arria 10 FPGA External Memory Interface Comparison**

Feature	Xilinx UltraScale FPGA	Intel Arria 10 FPGA	Notes
Recommended impedance	40Ω	40Ω	Some interfaces vary. See documentation for exact impedances.
On-chip termination	Digitally controlled impedance (DCI) VRP pin 240Ω to ground	OCT 240Ω RZQ to ground	
Address and command topology	Fly-by	Fly-by	
Number of independent interfaces per bank	2 Must be same voltage	1	Two independent PLLs in UltraScale/UltraScale+ allows unique frequencies, reference clocks can be shared.
Number of dependent interfaces	2 Must be same voltage	2 Must be same frequency, reference clock, voltage	Two independent PLLs in UltraScale/UltraScale+ allows unique frequencies, reference clocks can be shared.
Number of banks x16 DDR3/4	1	2	52 pins per bank fits a x16 interface in a single bank.
Clock/Address/ Command	Grouped	Fixed location based on hard controller	
DDR3/4 controller	Soft block	Hard controller	
DDR3/4 calibration	MicroBlaze™ processor	Hard NIOS	
Debug	Vivado lab tools and XSDB debugger	Signal tap	

Table 2-9 shows a comparison of the external memory interfaces for the Xilinx Artix-7/Spartan-7 and Zynq-7000 AP SoC architecture and the Intel Cyclone V and Cyclone V SoC architecture.

Table 2-9: **Xilinx Artix-7/Spartan-7 FPGA and Zynq-7000 AP SoC and Intel Cyclone V FPGA and Cyclone V SoC FPGA External Memory Interface Comparison**

Feature	Xilinx Artix-7 and Spartan-7	Xilinx Zynq-7000	Intel Cyclone V/ Cyclone V SoC FPGA	Notes
DDR3 FMAX	Artix-7: 1,066 Mb/s Spartan-7: 800 Mb/s	Up to 1,866 Mb/s	800 Mb/s	
Recommended impedance	40	40	40	Some interfaces vary. See documentation for exact impedances.
On-chip termination	IN_TERM Untuned 40, 50, or 60 Ohms	IN_TERM Un-tuned 40,50, or 60 Ohms	OCT 240 RZQ to ground to meet fmax untuned with lower performance	Xilinx no external reference pin required for untuned termination.
Address and command topology	Fly-by	Fly-by	Fly-by	
Number of independent interfaces per bank	1		1	
Controller	Soft		Hard for FMAX Soft for de-rated	
Debug	Hardware debugger		Signal Tap	

Use the Vivado I/O Pin Planner to assign DDR pins. The pin planner performs on-the-fly validation of pin assignment rules for the respective I/O banks.

Processing System (PS)

Xilinx Zynq-7000 AP SoC devices offer a high performance hardened processing system. Table 2-10 shows the key architectural differences between the Xilinx Zynq-7000 AP SoC processing sub-system and the Intel Cyclone V SoC FPGA processing system.

For more details on the Xilinx Zynq-7000 AP SoC processing system, see the *Zynq-7000 All Programmable SoC Overview* (DS190) [Ref 34] and the *Zynq-7000 All Programmable SoC Technical Reference Guide* (UG585) [Ref 35].

Table 2-10: **Xilinx Zynq-7000 AP SoC and Intel Cyclone V SoC FPGA Processing System Comparison**

	Xilinx Zynq-7000 AP SoC	Intel Cyclone V SoC FPGA
Processor core	Dual ARM® Cortex™-A9 MPCore™ with CoreSight™	Single or dual-core ARM Cortex-A9 MPCore with CoreSight
Processor extensions	NEON™ and single/double precision floating point for each processor	NEON and single/double precision floating point for each processor
Maximum frequency	Up to 1 GHz	Up to 925 MHz
L1 Cache	32 KB instruction, 32 KB data per processor	32 KB instruction, 32 KB data per processor
L2 Cache	512 KB	512 KB
On-chip memory	256 KB	64 KB
External memory support	DDR3, DDR3L, DDR2, LPDDR2	DDR3, DDR3L, DDR2, and LPDDR2
External static memory support	2x Quad-SPI, NAND, NOR	Quad-SPI, NAND
DMA channels	8 (4 dedicated to programmable logic)	8
Peripherals	2x UART, 2x CAN 2.0B, 2x I2C, 2x SPI, 4x 32b GPIO	2x UART, 2x CAN 2.0B, 4x I2C, 2x SPI, Up to 134 GPIO
Peripherals w/ built-in DMA	2x USB 2.0 (OTG), 2x tri-mode Gigabit Ethernet, 2x SD/SDIO	x USB 2.0 (OTG), 2x tri-mode Gigabit Ethernet, SD/SDIO/MMC, NAND
Security	RSA authentication, AES and SHA256b decryption for FSBL (first stage boot loader), bitstreams and other loadable modules (such as kernel, data tables, etc.)	

Analog Mixed Signal

Every Xilinx UltraScale FPGA and UltraScale+ FPGA contains a system monitor block (SYSMON) that contains on-chip temperature and supply sensors, as well as up to 17 external analog inputs and an integrated analog-to-digital converter (ADC).

The Xilinx Artix-7/Spartan-7 FPGAs and Zynq-7000 AP SoCs contain a high performance analog mixed signal block called the Xilinx analog-to-digital converter (XADC). The XADC has the same functionality as the SYSMON in UltraScale FPGAs, but with the addition of dual 12-bit 1 MSPS ADCs.

Table 2-11 shows a comparison of the analog mixed signal (AMS) capabilities for Xilinx and Intel. For more details on the SYSMON, see the *UltraScale Architecture System Monitor User Guide* (UG580) [Ref 36]. For more details on the XADC, see the *7 Series FPGAs and Zynq-7000 All Programmable SoC XADC Dual 12-Bit 1 MSPS Analog-to-Digital Converter User Guide* (UG480) [Ref 37].

Table 2-11: **Xilinx and Intel AMS Comparison**

	Xilinx Artix-7/ Spartan-7 FPGA	Xilinx Zynq-7000 AP SoC	Xilinx UltraScale/ UltraScale+ FPGA	Intel Cyclone V FPGA	Intel Cyclone V SoC FPGA	Intel Arria 10 FPGA
Temperature Sensor	±4°C max	±4°C max	±4°C max	No	No	Yes
On-chip supply sensor	±1% max	±1% max	±1% max	No	No	Yes
External Analog Inputs	Up to 17	Up to 17	Up to 17	No	No	2
Dual 1 MSPS ADC	Yes	Yes	–	No	No	No

Board Design

Xilinx provides you with several tools for getting your board design right the first time.

Schematic Review

When a schematic is ready to be reviewed, the *UltraScale Architecture Schematic Review Checklist* (XTP344) and the *7 Series Schematic Review Recommendations* (XMP277) can be used as comprehensive checklists for schematic review.

Power Consumption

The Xilinx Power Estimator (XPE) utility is a spreadsheet-based power estimation tool similar to the EPE. It is a widely used utility for device selection, power supply planning, and thermal modeling. It can also receive post-implementation data from the Vivado design tools.

VIDEO: *If the design is complete, customers can use the information from the Vivado Report Power video as input for their PCB design.*

Power Supply Information

Xilinx devices typically require the same number or fewer voltage rails than Intel devices require. A given FPGA requires various voltage supplies depending on the design requirements. A list of the supply voltages along with their recommended operating conditions can be found in the device data sheet. Other useful documents for PCB designers are:

- For UltraScale FPGAs: *UltraScale Architecture PCB Design User Guide* (UG583) [Ref 7]
- For Artix-7/Spartan-7 FPGAs: *7 Series FPGAs PCB Design Guide* (UG483) [Ref 38]
- For Zynq-7000 AP SoC: *Zynq-7000 All Programmable SoC PCB Design Guide* (UG933) [Ref 39]

PCB Decoupling Guidelines

Xilinx provides decoupling capacitor guidelines based on the device and package combination that you are using. Xilinx devices also include decoupling capacitors on the device substrate, which provide better high-frequency noise isolation than PCB mounted caps. See the applicable user guide for the guidelines:

- For UltraScale FPGAs: *UltraScale Architecture PCB Design User Guide* (UG583) [Ref 7]
- For Artix-7/Spartan-7 FPGAs: *7 Series FPGAs PCB Design Guide* (UG483) [Ref 38]
- For Zynq-7000 AP SoCs: *Zynq-7000 All Programmable SoC PCB Design Guide* (UG933) [Ref 39]

Transceiver decoupling guidelines are in the asSoCiated GTH or GTY transceiver user guide.

Simulation Tools for PCB Designers

Xilinx provides industry standard simulation models for signal integrity and thermal simulation. For GPIO performance, IBIS and SPICE models are available. For transceivers, Xilinx provides IBIS-AMI models for accurate algorithmic modeling of channel compensation in the auto-adapting equalizers. Thermal models are also available.

Chapter 3

Quartus II Design Tools to Vivado Design Tools

Overview

Managing a project using the Intel® Quartus® II design tools is very similar to managing a project using the Xilinx® Vivado® design tools. Both of these design tools are used to manage sources, constraints, and project settings through an integrated development environment (IDE). Both tools also allow project management through scripting. However, in the Vivado design tools, there is a design flow option in addition to the project concept, and this flow instead uses Tcl commands to control the entire flow. In this non-project mode, a project is still created in memory, but project files are not written to disk.

These Xilinx user guides provide detailed information about the Vivado Design Suite:

- To get started learning about the Vivado Design Suite, see the *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 8]
- For detailed information on the Vivado Design Suite flows, see the *Vivado Design Suite User Guide: Design Flows Overview* (UG892) [Ref 9]
- For a step-by-step walkthrough of a Vivado Design Suite project, see the *Vivado Design Suite Tutorial: Design Flows Overview* (UG888) [Ref 10]

The Vivado Design Suite allows you to work in project mode or non-project mode. Irrespective of the mode you are working in, the Vivado tools can be run using the GUI or Tcl commands. Consequently, any Quartus II use model should be available in the Vivado Design Suite.

Vivado IDE Projects

Both the Xilinx and Intel design tools use an integrated design environment (IDE) with a New Project wizard. This wizard provides a guided step-by-step process for creating a project with sources, constraints, and project settings targeting a programmable device. There is no direct migration available between Quartus II files (QPF, QSF, QXP, etc.) and

Vivado Design Suite project files. As a result, a Vivado Design Suite project must be created from scratch using the New Project wizard.

For assistance in using this wizard, see the "Creating Projects" section of the *Vivado Design Suite User Guide: Using the Vivado IDE* (UG893) [Ref 11]. This user guide can also step you through running your project through the Vivado IDE all the way to creating a bitstream.

Vivado Design Suite Tcl Projects

Tcl is the native programming language for the Vivado Design Suite. Tcl provides the full power and flexibility of the language to control the application, access design objects and their properties, and create custom reports. Quartus II also uses Tcl to specify settings and constraints rather than using the IDE. The Tcl commands in Quartus II do not map directly to Vivado Design Suite Tcl commands, so any Tcl scripts used in the Quartus II flow must be recreated for the Vivado Design Suite. For more information on what Tcl commands are available in the Vivado Design Suite, see the *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 12].

UltraFast Design Methodology

While the Quartus II Handbook provides some information on design best practices, Xilinx has heavily invested in customer success and provides the comprehensive *UltraFast Design Methodology Guide for the Vivado Design Suite* (UG949) [Ref 13]. This guide helps you maximize system performance, reduce risk in using Xilinx solutions, and enable accelerated and predictable design cycles. This guide covers all aspects of:

- Board and device planning
- Design creation and IP integration
- Implementation and design closure
- Configuration and hardware debug

The Vivado Design Suite automates part of the UltraFast design methodology through linting rules (used to analyze code for potential errors), checklists in the documentation navigator, design rule checks (DRCs), and templates for high-level design (HLD) coding style and constraints. For more information on the UltraFast design methodology, see the UltraFast Design Methodology website. The Xilinx Documentation Navigator is part of the Vivado tools, but can also be downloaded from the Downloads site for standalone use.

Quartus II Tools to Vivado Tools

Most of the capabilities available in the Quartus II tools are also available in the Vivado Design Suite. However, the terminology and the naming of the individual tools and file formats differ in some cases. Table 3-1 lists the more commonly-used tools and file extensions and the corresponding name in the Vivado Design Suite.

Table 3-1: **Mapping Quartus II and Vivado Tools and File Extensions**

Description	Quartus II Tools	Vivado Design Tools
Timing analyzer	TimeQuest	Vivado report_timing
A tool used to connect various IP cores to create a sub-system or design	Qsys	Vivado IP Integrator
A tool used to create and simulate DSP type designs using DSP blocks	DSP builder	System Generator
A tool for programming hardware	Programmer	Vivado programmer
Tools used to generate FPGA programming files or flash files	quartus_asm	write_bitstream/ write_cfgmem (Tcl command)
In-system logic analyzer tool	SignalTap II logic analyzer	Vivado logic analyzer
Tool to control virtual stimuli and observe response	In system sources and probes editor	VIO window in the Vivado logic analyzer
An IP core used with an analyzer tool to monitor serial transceiver performance	Transceiver toolkit	IBERT
A tool used to monitor and debug calibration debug status for external memory interfaces	External memory interface toolkit	Memory calibration debug (available for UltraScale devices)
Project file	.qpf+.qsf	.xpr
IP configuration file	.qip	.xci
Block diagram file	.bdf	.bd
Design module created with the Qsys tool	.qsys	.bd
Project archive	.qar	.zip
Technology-specific netlist	.vqm	.EDIF or .v
Tcl interpreter to interact with the hardware bus interface	System console	Tcl console
Hardware IP to hook into the bus interfaces	JTAG to Avalon master bridge	JTAG-to-AXI master IP

SoC Conversion

Overview

There are some differences between Zynq-7000 AP SoC and the Cyclone V SoC FPGA devices, However, at a high level, they are similar (dual-core ARM® Cortex™-A9 integrated with and tightly coupled to an FPGA). The Xilinx tools, IP cores, software utilities, and runtime software make it easy for you to migrate your designs to a Zynq-7000 AP SoC based platform. The amount of effort and the time it takes to migrate depends on the complexity of the design.

This chapter describes converting a hardware and software design targeting an Intel Cyclone V SoC FPGA to a Xilinx Zynq-7000 AP SoC in the following areas:

- Hardware Design
  - Processing System Configuration
  - RTL Logic and PL Configuration
- Software Development
  - Bare-metal Application
  - Operating Systems Based Applications
  - Boot and Configuration

Figure 4-1 shows a typical development flow for the Zynq-7000 AP SoC.

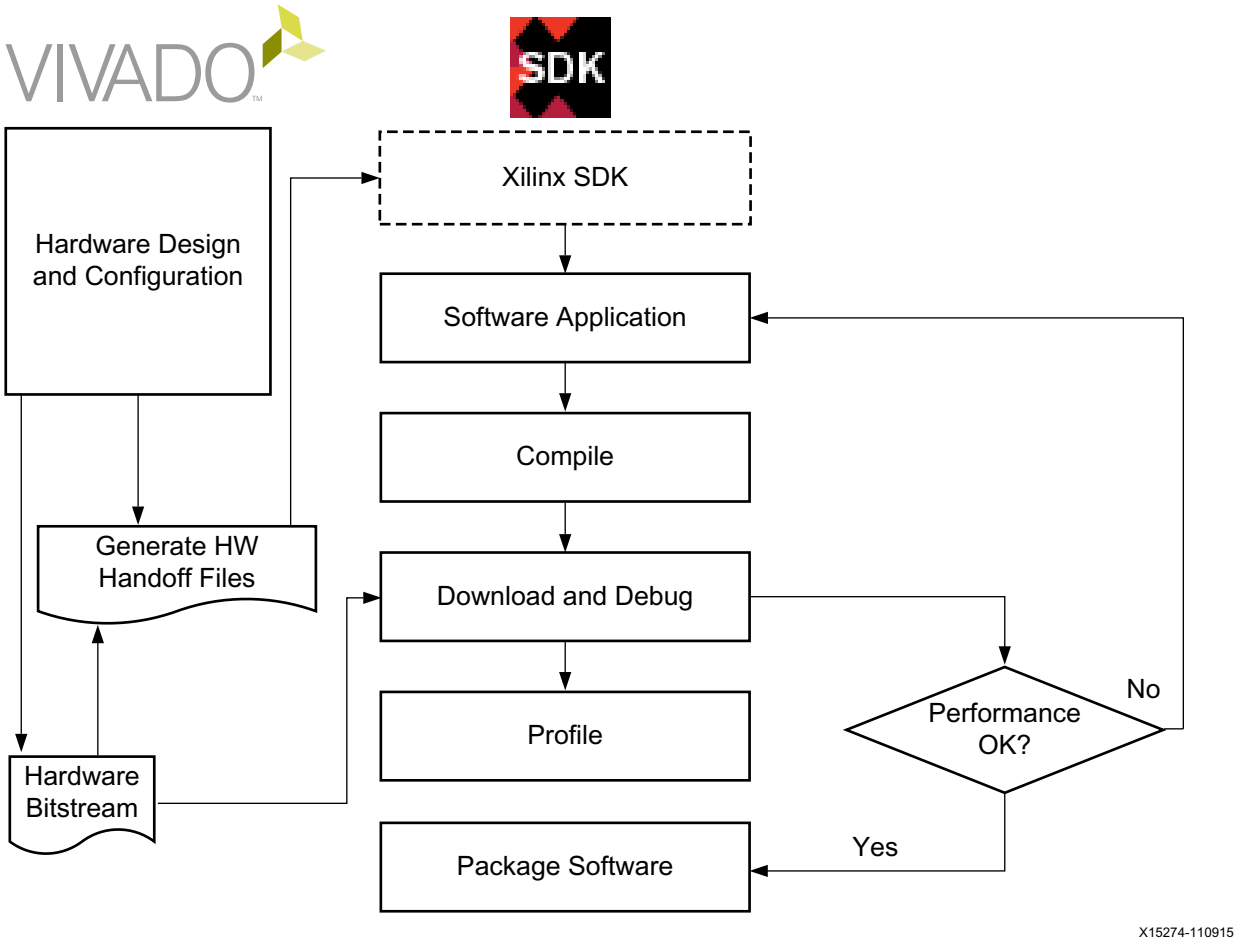

Figure 4-1: **Zynq-7000 AP SoC Development Flow**

Terminology Differences

Many of the Xilinx devices and tools are similar to those from Intel. Some of the concepts and acronyms in the Xilinx documentation have an equivalent in the Intel documentation. Table 4-1 shows a comparison of the more commonly used Xilinx terms and the Intel equivalent.

Table 4-1: **Xilinx and Intel Terminology Comparison**

Intel Cyclone V SoC FPGA	Xilinx Zynq-7000 AP SoC	Description
HPS	Processing system (PS)	Dual core A9 and hardened peripherals.
FPGA	Programmable logic (PL)	
Qsys	Vivado IP integrator	Both tools are used to configure and connect IP blocks. The IP integrator GUI makes it much more convenient to add and connect IP cores.
System console	Hardware manager	Both enable programming the hardware logic and debugging the design running on the fabric.
On-chip memory	Block RAM	Memory available in the PL to be configured in various ways for use in the system.
On-chip RAM – 64 KB	On-chip memory (OCM) – 256KB	
PreLoader	First stage boot loader (FSBL)	The start-up code that typically runs after the ROM code has been executed.
.sof file	Bitstream	Configuration data for the device that is downloaded from the tools.

Additionally, there are ports between the PS and PL (HPS and FPGA for Intel) that are similar but slightly different. For Xilinx, there are general purpose AXI ports, high-performance AXI ports, accelerator coherency port (ACP), etc. For Intel, there are FPGA-to-SDRAM interface, FPGA-to-HPS bridge, HPS-to-FPGA bridge, and lightweight HPS-to-FPGA Bridge, etc.

Hardware Design

The Zynq-7000 AP SoCs and Cyclone V SoC FPGAs include a processing system (PS) block and a fabric block called the programmable logic (PL) block. These two blocks must be configured to create a working system on which to run the software. Configuring the PS is described in this chapter. Converting the PL is described in Chapter 5, IP Core Conversion and Chapter 6, RTL Conversion.

Processing System Configuration

The Zynq-7000 AP SoC includes a processing system (PS) similar to the HPS component in the Cyclone V SoC FPGA. The Zynq-7000 AP SoC PS block contains either dual or single ARM Cortex-A9 cores, and support for peripherals. PS configuration includes identifying peripherals and configuring individual components in the system. Identifying peripherals for migration is relatively simple. First, identify the peripherals configured as a part of the current design, and then map them with the ones supported by the Zynq-7000 SoC.

The Vivado IP integrator is an advanced tool that allows easy configurability for the PS components (similar to Qsys for the Cyclone V SoC FPGA). In addition to configuring the PS, it presents a GUI front-end for adding IP cores on the PL side of the Zynq-7000 AP SoC. See the *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994) [Ref 43] and the *Zynq-7000 All Programmable SoC Technical Reference Guide* (UG585) [Ref 35] for more information on creating a Zynq-7000 AP SoC based PS configuration.

Figure 4-2 shows the Zynq-7000 AP SoC PS (dual core) configuration possibilities in the Vivado Design Suite. The blocks highlighted in green can be clicked to enable detailed configuration.

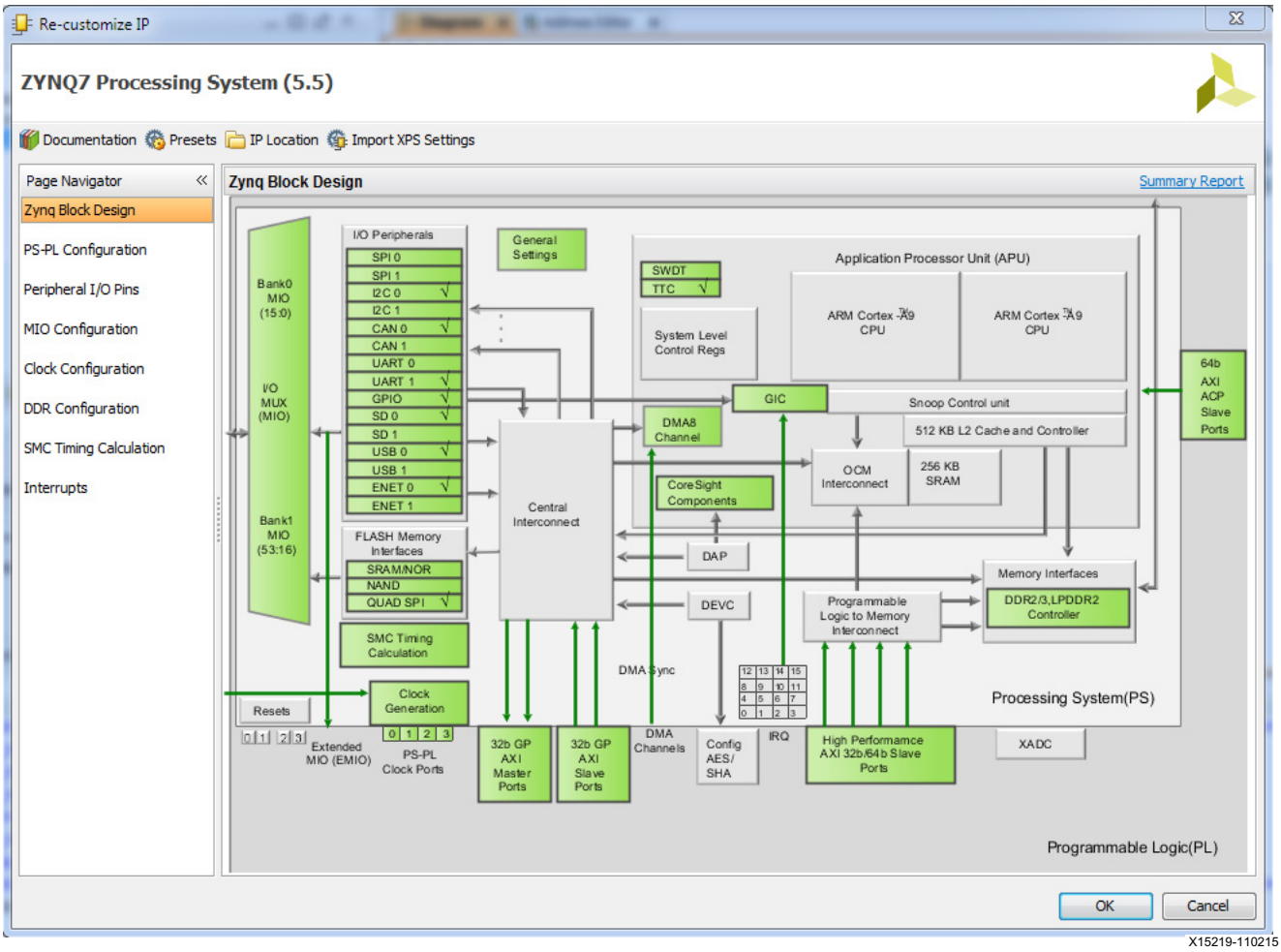

Figure 4-2: **Zynq-7000 AP SoC PS Configuration (Dual Core)**

Interconnect

The application processor unit (APU), memory interface unit, and the I/O peripherals (IOP) are connected to each other and to the PL through a multilayered ARM AMBA® AXI interconnect. The interconnect is non-blocking and supports multiple simultaneous master-slave transactions. Some of the interconnect features include:

- Based on AXI high-performance datapath switches.
- Central interconnect switch based on ARM NIC-301.
  - Master interconnect for slave peripherals.
    - The master interconnect switches the low-to-medium speed traffic from the central interconnect to M_AXI_GP ports, I/O peripherals (IOP), and other blocks.

    - Slave interconnect for master peripherals.
        - The slave interconnect switches the low-to-medium speed traffic from S_AXI_GP ports, DevC, and DAP to the central interconnect.
- Memory interconnect (PL to memory interconnect).
    - The memory interconnect switches the high-speed traffic from the AXI_HP ports to DDR DRAM and on-chip RAM (through another interconnect).
- OCM interconnect.
    - The OCM interconnect switches the high-speed traffic from the central interconnect and the memory interconnect.
- AHB and APB bridges.

PS-PL Interfaces

The Zynq-7000 AP SoC includes a set of PS-PL interfaces that are comparable to the HPS-FPGA bridges. This section highlights the PS-PL interfaces available in the Zynq-7000 AP SoC and how they differ from similar bridges available in the Cyclone V SoC FPGA. The different types of PS-PL AXI interfaces in the Zynq-7000 AP SoC are:

- S_AXI_ACP, one cache coherent port for a master in the PL.
- S_AXI_HP, four high-performance/bandwidth ports for masters in the PL.
- AXI_GP, four general purpose ports (two master ports and two slave ports).

The PS-PL interfaces available in the Zynq-7000 AP SoC differ from the interfaces provided by the Cyclone V SoC FPGA. In the Zynq-7000 AP SoC, there are four GP ports of fixed width (32 bits), two with the PS as master and two with the PL as master. The Cyclone V SoC FPGA provides two high bandwidth user configurable ports (32/64/128 bit), one for HPS to FPGA and the other for FPGA to HPS connection. It also supports a low latency 32-bit HPS to FPGA interface. However, the Cyclone V SoC FPGA has only one master port from FPGA to HPS.

The Zynq-7000 AP SoC, like the Cyclone V SoC FPGA, provides a high throughput datapath (high performance port (HP)) between PL masters and PS memories, including DDR and on-chip RAM. This interface supports up to a 64-bit wide data interface. However, for the Cyclone V SoC FPGA, a similar interface can be configured up to 256-bit wide. Consequently, if your current design uses 256-bit wide interfaces, use an AXI data width converter IP (available in the Vivado IP catalog) to generate 64-bit interfaces. This is also applicable when migrating a design from 64/128-bit HPS-FPGA (or FPGA-HPS) interfaces to 32-bit GP ports on the Zynq-7000 AP SoC.

Performance is more than just the width of an interface – it is based on the complex interaction of several pieces and needs to be modeled and measured. The Xilinx SDK provides a system performance modeling (SPM) and system performance analysis utility for modeling, measuring, and optimizing the overall system performance. See the *Xilinx Software Development Kit (SDK) - System Performance Analysis* (UG1145) [Ref 44] and the *System Performance Analysis of an All Programmable SoC Application Note* (XAPP1219) [Ref 45].

Table 4-2 lists the components in the Zynq-7000 AP SoC PS block and contrasts them against the ones supported by HPS in the Cyclone V SoC FPGA. Table 4-2 shows that there is almost equivalent peripheral and system configuration support for the Zynq-7000 AP SoC. See the *Zynq-7000 All Programmable SoC Technical Reference Manual* (UG585) [Ref 35] to understand configurations supported for each peripheral in more detail.

Table 4-2: **Zynq-7000 AP SoC PS Block and Cyclone V SoC FPGA HPS Comparison**

Cyclone V SoC FPGA	Zynq-7000 AP SoC	Vivado Design Suite Representation
Clock manager	Clock management and clock generation	Clock configuration wizard in Vivado IP integrator
Reset manager	Reset system	Processor system reset IP in Vivado IP integrator
Interconnect	ARM AMBA interconnect	See "Chapter 5, Interconnect" in the *Zynq-7000 All Programmable SoC Technical Reference Manual* (UG585) [Ref 35].
HPS-FPGA AXI bridge	PS-PL interfaces	GP, HP, and ACP
Cortex-A9 MPU	Dual or Single-core ARM Cortex-A9 based APU	See the *Zynq-7000 All Programmable SoC Technical Reference Manual* (UG585) [Ref 35] and the *Zynq-7000 All Programmable SoC Software Developers Guide* (UG821) [Ref 46]
Coresight technology debug and trace	Coresight™ technology JTAG and DAP subsystem	See the *Zynq-7000 All Programmable SoC Technical Reference Manual* (UG585) [Ref 35] and the *Zynq-7000 All Programmable SoC Software Developers Guide* (UG821) [Ref 46]
SDRAM controller subsystem	DDR memory controller	DDR configuration
On-chip RAM and BootROM	OCM and BootROM	
NAND flash controller	Static memory controller (can be used as a NAND flash controller, or parallel port memory controller)	
SD/MMC controller	SD/SDIO controller	MIO configuration in Vivado IP integrator
Quad SPI flash controller	Quad SPI flash controller	MIO configuration in Vivado IP integrator
FPGA manager	Processor configuration access port (PCAP)	
System manager	System level control registers (SLCR)	See the "4.3 SLCR Registers section in Chapter 4, System Addresses" in the *Zynq-7000 All Programmable SoC Technical Reference Manual* (UG585) [Ref 35]

Table 4-2: **Zynq-7000 AP SoC PS Block and Cyclone V SoC FPGA HPS Comparison *(Cont'd)***

Cyclone V SoC FPGA	Zynq-7000 AP SoC	Vivado Design Suite Representation
Scan manager	JTAG and DAP subsystem	
DMA controller	DMA controller (DMAC)	Zynq-7000 AP SoC block design
Ethernet MAC	Gigabit ethernet controller (GEM)	ENET0, ENET1 in the Zynq-7000 AP SoC block design
USB OTG controller	USB host, device, and OTG controller	USB 0, USB 1 in the Zynq-7000 AP SoC block design
SPI controller	SPI controller	Zynq-7000 AP SoC block design
I2C controller	I2C controller	MIO configuration
UART controller	UART controller	MIO configuration
General-purpose I/O interface	General purpose I/O	MIO configuration
Timer	Timer	MIO configuration
Watchdog timer	Watchdog timer	MIO configuration

VIDEO: *See the Designing with Vivado IP Integrator video for more information on how to configure the Zynq-7000 AP SoC block using the Vivado IP integrator.*

RTL Logic and PL Configuration

After identifying the configuration required for the PS block, the next step is to migrate the programmable logic part of the design. The Zynq-7000 AP SoC includes FPGA programmable logic that is comparable to the FPGA fabric in the Cyclone V SoC FPGA. Like the FPGA in the Cyclone V SoC, the Zynq-7000 AP SoC PL also includes a set of hardened IP cores that can be used to implement an existing design. The hardened IP cores in the Zynq-7000 AP SoC PL are block RAM, DSP slice, clock management (MMCM and PLL based, Cyclone), and the integrated interface block for PCI Express® designs. Also, the Vivado IP catalog provides soft IP cores that can be used when needed. See the Intellectual Property website to browse IP cores and obtain more information on the IP cores supported by Xilinx or by Xilinx partners. For more information, see Chapter 5, IP Core Conversion and Chapter 6, RTL Conversion.

Software Development

After the hardware for the PS and PL is configured using the Vivado Design Suite, the hardware design files from the Vivado Design Suite are exported to the Xilinx SDK software development tools. The files sent from the hardware flow to the software flow (referred to as handoff or HDF files) contain information such as the hardware specification, PS peripheral information, register memory map, and a bitstream for the PL. The handoff files present the hardware design as a standard memory map such as for any ASSP, even though the hardware could change over time. This hardware handoff file insulates software developers from the changes that might occur on the hardware side during development. Software developers can use the handoff files to design firmware, drivers, and a board-support package (BSP).

Bare-metal Application

Unlike OS-based application porting, bare-metal application porting requires some effort. The bare-metal application must be modified in sync with the hardware platform and the supported address map. Xilinx provides bare-metal drivers (with source code) for the Zynq-7000 AP SoC peripherals and libraries for other hardware configurations. This coupled with BSP generation capabilities in Xilinx SDK simplifies the transition of a bare-metal software application to the Zynq-7000 AP SoC. The availability of source code makes it easier to modify and debug the drivers if needed. After the BSP is set and available, the application can be modified to make use of the Zynq-7000 AP SoC specific hardware components. For this migration, using a Zynq-7000 AP SoC evaluation kit, with available pre-defined BSP and application templates helps expedite the bare-metal application porting process.

Operating Systems Based Applications

An operating system based application migration can be easier than migrating a bare-metal application, for the simple reason that the operating system abstracts the lower layer from the user application and is standard across different processor systems. The migration effort might also depend on the operating system that is being considered. For example, both the Cyclone V SoC FPGA and the Zynq-7000 AP SoC support the Linux operating system. Consequently, migrating a user application for a Linux target or supported RTOS can be done relatively quickly because most of the hardware differences have been abstracted away at the lower levels.

Linux and PetaLinux Tools

Xilinx provides a Linux distribution available from the Xilinx GIT repository. Additionally Xilinx provides the PetaLinux tools that can be used to configure, customize, build, and deploy Linux for the Zynq-7000 AP SoC based on the hardware configuration for the specific system (using hardware handoff files exported from the Vivado Design Suite). Based on the available hardware design, PetaLinux generates a device tree file that enables necessary drivers. The kernel and the RootFS can be further configured to support the user applications.

For Linux-based applications, with few or no modifications, the same application code can be compiled and run on the new Zynq-7000 AP SoC based Linux target. The Linux applications can be compiled using both the Xilinx SDK and the PetaLinux tools. The Xilinx SDK also provides Linux application debug support via the system debugger.

Other Operating Systems

The Xilinx SDK supports the FreeRTOS operating system, including driver support. Consequently, when migrating FreeRTOS based applications, the SDK can be used to generate a BSP based on the hardware handoff files. The drivers can then be modified to support specific requirements, and the same application can be recompiled to run on a Zynq-7000 AP SoC based target. The Zynq-7000 AP SoC is supported by many other operating systems, such as Micrium uC/OS, Android, VxWorks, and QNX. Migrating the applications based on these operating systems is comparable to migrating Linux applications, however, the drivers and the BSP might need to be modified for the new hardware platform. For a complete list of operating systems available for the Zynq-7000 AP SoC, see the Zynq-7000 AP SoC Ecosystem website.

Boot and Configuration

Boot and configuration require some thought and probably some modification in the existing boot code. For the Zynq-7000 AP SoC, the processors in the PS always boot first, allowing a software-centric approach for PL configuration. The PL can be configured as part of the boot process or configured in the future. Additionally, the PL can be completely reconfigured or used with partial dynamic reconfiguration (PR). PR allows configuration of a portion of the PL. This enables optional design changes such as updating coefficients or time-multiplexing the PL resources by swapping in new algorithms as needed. This latter capability is analogous to the dynamic loading and unloading of software modules. The PL configuration data is referred to as a bitstream.

Xilinx provides a first stage bootloader (FSBL) that loads to on-chip memory (OCM). The FSBL initializes all the necessary peripherals and (optionally) loads the available PL bitstream before. It then sends the code to either the bare-metal application or the second stage bootloader (such as U-boot), which then loads the chosen operating system. The Zynq-7000 AP SoC has significantly more OCM than the Cyclone V SoC FPGA (256 KB for the Zynq-7000 AP SoC compared to 64 KB for the Cyclone V SoC FPGA). This increased OCM

provides the possibility for expanding the scope of the customer's existing preloader based bootloader when migrating to a bootloader for the Zynq-7000 AP SoC.

IP Core Conversion

Overview

Porting IP cores can be a difficult task. To perform this task successfully, it is necessary to understand the IP cores flow, the port differences, the functional differences, and how to retest. Each of these issues is described in this section.

IP Cores Flow

The Vivado Design Suite includes many features to simplify adding an IP core to a design. One feature is the Vivado® IP integrator, which is used for generating and quickly connecting multiple IP blocks regardless of the complexity. Another feature is the Vivado IP catalog, which is used to generate IP cores for inclusion in the HDL code.

Inference is supported as well (see the *Vivado Design Suite User Guide: Synthesis* (UG901) [Ref 14]).

Whichever flow you prefer, Xilinx is ready to help you learn and understand any of the Vivado Design Suite features. For QuickTake video tutorials that walk you through the various flows, see the Vivado Video Tutorials website. For comprehensive training classes, see the Training website.

VIDEO: *To see an example of how to put a complex IP core together quickly in the IP integrator, watch the AXI PCI Express® MIG Subsystem Built in IPI video.*

AXI and Avalon Differences

RECOMMENDED: *You should modify any Avalon user logic interface to be compliant with the corresponding AXI interface.*

Xilinx uses the industry standard AMBA® AXI interface protocol, while Intel® uses the proprietary Avalon interface protocol. One possibility for interfacing existing Avalon-based interfaces with AXI-4 is inserting a "logic shim" module to translate the interface handshaking and buffer the data. However, this approach requires adding FIFOs, which increase latency and consume resources. Further, the complexity of designing the shim logic is about equal to the effort in redesigning the existing interface, because in either case the user designs the interface to the AXI-4 standard.

There is a memory-mapped and streaming interface for both AXI and Avalon. For the streaming interface, it is virtually a 1:1 mapping. Avalon has a start of packet signal (SOP), as shown in Table 5-1.

Table 5-1: **Migrating Avalon to AXI4**

Avalon-ST	~AXI4
channel	TID/TUSER
data	TDATA
error	TUSER
ready	TREADY
valid	TVALID
empty	TSTRB + TKEEP
endofpacket	TLAST
startofpacket	1st TVALID + TREADY

AXI indicates SOP with the assertion of TVALID and TREADY at the same time. Looking at a 10G MAC IP core for example (see Figure 5-1 and Figure 5-2), it is easy to see the similarity in the interface timing waveforms.

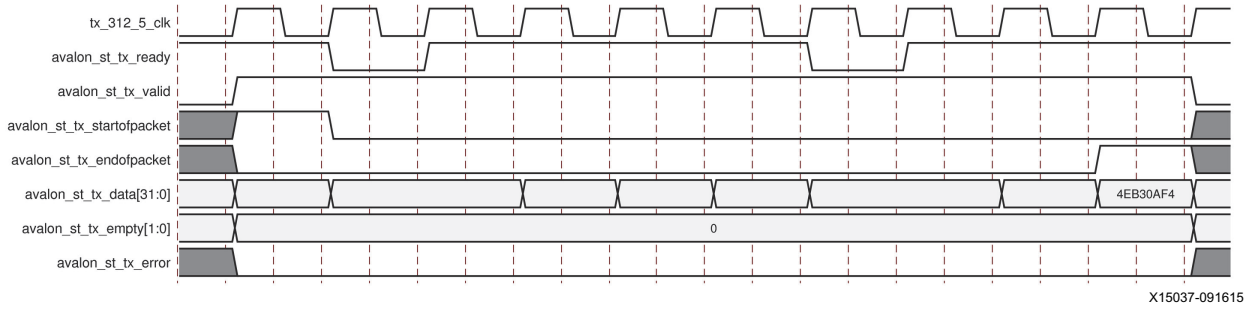

Figure 5-1: **Avalon ST Interface**

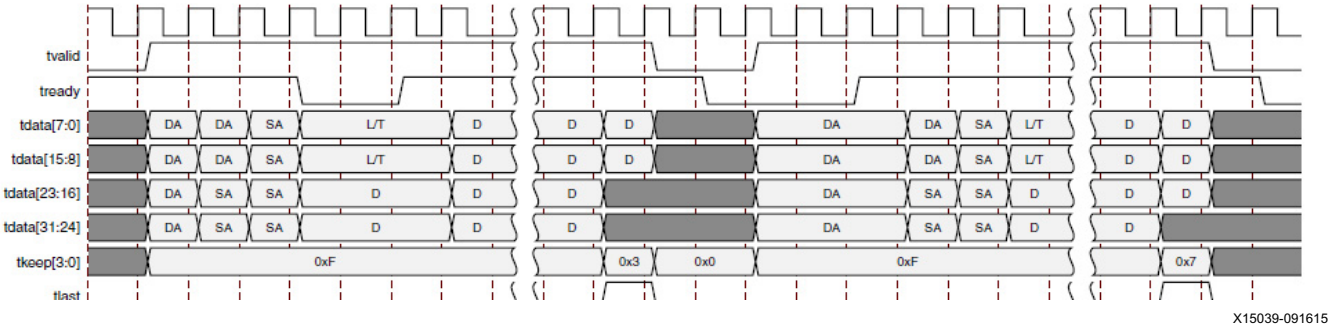

Figure 5-2: **AXI ST Interface**

However, there are some subtle differences, as listed in Table 5-2.

Table 5-2: **Avalon and AXI Comparison**

Avalon	AXI	Details
Error	TUSER	Side band signal that can be used for communicating errors.
Empty	TKEEP + TSTRB	Determines which bytes are valid in a given packet.

For the memory-mapped version, AXI separates the data, address, and control into different channels. The user logic for an Avalon interface requires some minor modifications for adaptation to the AXI standard. For example, the logic that generates an address value must also generate a valid signal with it as long as ready is asserted.

Figure 5-3 shows an example of an AXI memory-mapped interface. One channel is the address and another is the read data. Burst count is controlled by the AWLEN port, which is not shown.

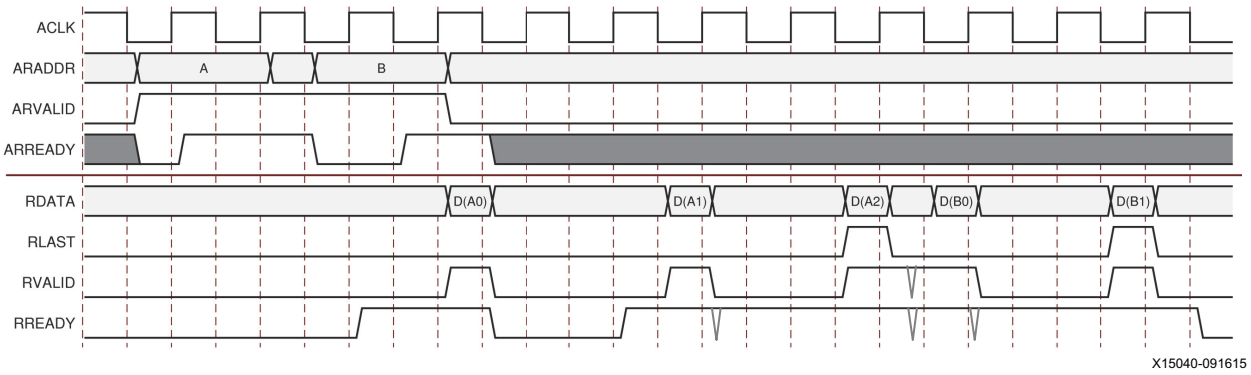

Figure 5-3: **AXI Memory-mapped Interface**

Avalon has a single channel for data, address, and control, as shown in Figure 5-4.

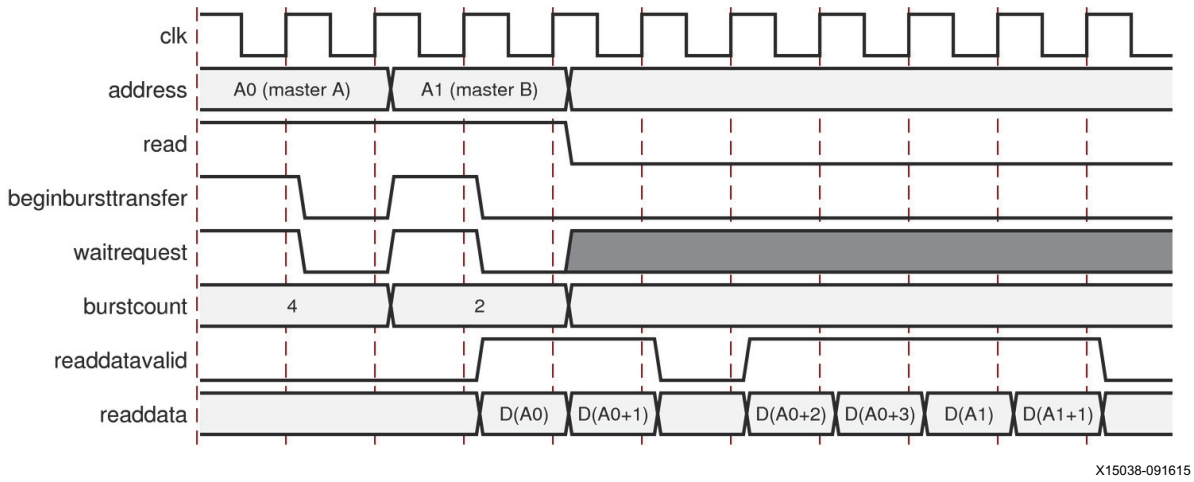

Figure 5-4: **Avalon Memory-mapped Interface**

Inspection of these timing diagrams shows that the two main differences between the interface protocols are the channels, and the lack of back-pressure with the Avalon interface. Avalon has a waitrequest signal, but this is a slave to a master only, and the master cannot throttle data. AXI separates writes and reads into different channels and the master or slave can throttle data as required. See the applicable IP core user guide on the Xilinx documentation website for detailed port configurations. After the differences

between the Avalon and AXI interfaces are understood, it is possible to modify the existing user logic to be compliant with the AXI interface. For more information about the AXI interface, see the *AXI Reference Guide* (UG761) [Ref 15].

Functional Differences

Functional differences are a concern when changing or upgrading an IP core. Each IP I/O functional description should be checked against the legacy IP to ensure compatibility. Especially look for differences in polarity, handshaking, format, and latency. Review and compare timing diagrams to ensure that the cycle-to-cycle functionality is compatible, and if not, revise the logic as necessary.

Another area for attention is the block memory write mode, which can affect the clock cycle of when data is available. See the *UltraScale Architecture Memory Resources User Guide* (UG573) [Ref 5] for more information. Also important is the input or the output registers in the IP. These are set when you customize the IP, so some tuning might be required during functional verification.

After the full RTL conversion and IP conversion is complete, functional simulation can begin. As of the Vivado Design Suite 2015.3 release, there is a Tcl command `export_simulation` that creates a self-contained simulation environment. For a list of options, run `export_simulation -help`. `export_simulation` for help with setting up a complete environment for the desired simulator. All scripts, libraries, and HDL files are available in this directory to make simulation and verification easy. If you prefer to simulate the IP in a stand-alone environment, example designs are provided for many IP cores. These example designs include a simulation environment. See the documentation for the specific IP block for more information about the IP's example design.

RTL Conversion

Overview

This chapter covers the different aspects of converting low-level IP cores, such as RAMs, FIFOs, clocking, adders, multipliers, and shift-registers. In general, usage of low-level IP cores is asSoCiated with RTL-based designs. IP core conversion for higher-level design flows or interfaces is described in Chapter 5, IP Core Conversion.

Inferred Logic

In general, synthesizable HDL code written in Verilog or VHDL for use with Quartus II should be compatible with the Vivado design tools without modification. The RTL code might have additional synthesis pragmas to guide the Intel synthesis tools to behave in a certain manner. These pragmas will most likely need to be translated into the corresponding Vivado Design Suite equivalent. Chapter 8, Synthesis and Implementation provides a mapping of these pragmas.

Some HDL code might have been inferred as an IP core. Both Intel and Xilinx provide HDL templates for inferring IP cores. These templates are example segments of HDL code that are designed so that the synthesis tool recognizes a low-level IP function and implements it appropriately. There might be minor variants in some of the Quartus II core templates versus the corresponding Vivado IP templates.

You should inventory (as explained in the Analysis section) the existing IP cores in your design and ensure that each IP core instance is translated properly. The Quartus II log file lists inferred IP cores in the LPM Parameter Settings section of the compilation report.

Synthesis Verification

Minor recoding might be needed in the HDL portion where an IP core was inferred to ensure efficient realization in the converted design. Pay attention to the messages in the synthesis log file. As part of the conversion process, it might be easier to individually synthesize each module containing inferred IP cores because this provides easier readability of the log file (less clutter) and quicker turn-around time when debugging. After each module's synthesis log clean, you can take the entire design through the flow.

An example of a synthesis log file is shown in Figure 6-1. The design synthesized in this example contains only a single-port RAM HDL template, 8-bits wide and 128-bits deep. The log shows a RAMB18 being inferred from the template, along with an informational message that it might be more efficient to implement this RAM as a LUT RAM.

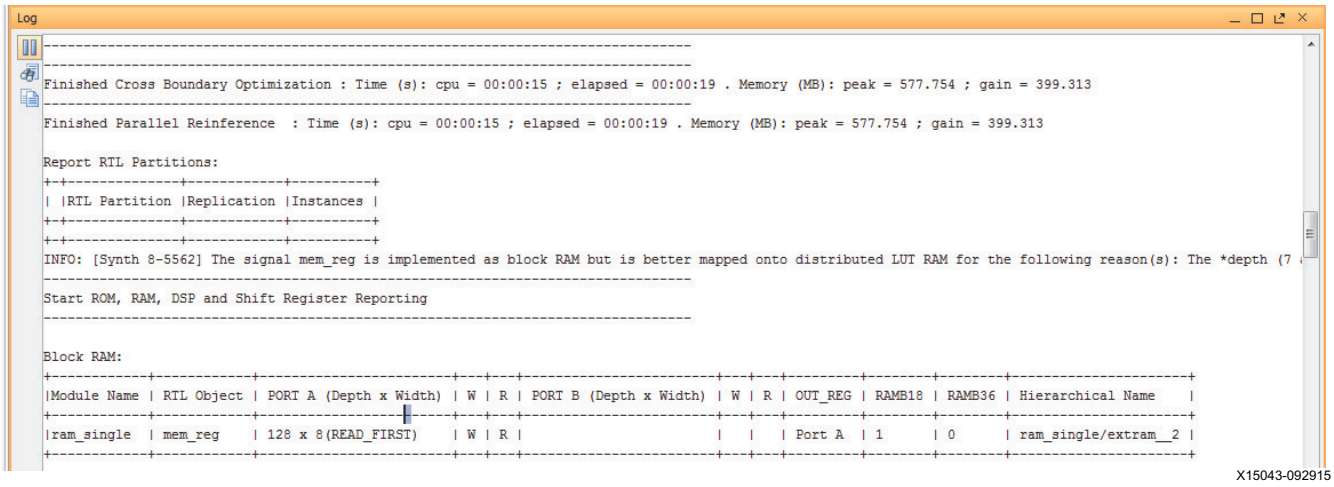

Figure 6-1: **Synthesis Log File Example**

RECOMMENDED: *Post-synthesis inferred logic should be comparable in utilization, when compared to your original design. To further understand how to compare resource utilization, see Resource Utilization in Chapter 8.*

Vivado Language Templates

See the *Vivado Design Suite User Guide: Synthesis* (UG901) [Ref 14], "Chapter 3: HDL Coding Techniques" for further information about inferring IP blocks using HDL templates.

Primitive Elements

To convert a primitive element instantiated in HDL code:

1. Locate the equivalent primitive element in the *Xilinx 7 Series and Zynq-7000 All Programmable SoC Libraries Guide for HDL Designs* (UG768) [Ref 17] or the *UltraScale Architecture Libraries Guide* (UG974) [Ref 18]. These user guides provide VHDL and Verilog instantiation templates for each Xilinx primitive and information about their usage.
2. Note any differences in the functionality or pin naming of the Intel primitive versus the equivalent Xilinx primitive, and adjust the design appropriately.

I/O

The I/O conversion method depends on the type of flow used to insert the I/O buffer:

- For converting instantiated I/O primitives, the instances must be modified to match the equivalent Xilinx I/O primitives. Refer to Primitive Elements regarding conversion of primitive elements.
- Vivado synthesis or Vivado opt_design infers I/O buffers and 3-state I/Os in cases where no pads are directly instantiated. Ensure that the proper constraints are in place for the I/O standards asSoCiated with the inferred I/O.
- For I/O asSoCiated with high-level IP functions (transceivers, ethernet, PCIe®, etc.) this I/O conversion takes place with the asSoCiated high-level IP conversion. See the appropriate sections in this guide for further information.

Intel IP Cores (Megafunctions)

Analysis

To complete any engineering task in the most efficient manner, analysis and preparation are essential and important steps. RTL design conversion is no exception to this rule. Xilinx recommends creating a list to prepare for the conversion of Intel® IP cores (megafunctions). At minimum, this IP list should contain:

- IP type
- Hierarchical name
- HDL file name that instantiates the IP
- IP core parameters (if parameterizable IPs are part of your design)

Most of the required information is provided in the “Analysis and Synthesis” section of the Quartus II project compilation report window. Consult these report sections (when present) to collect data:

- RAM summary
- DSP block usage summary
- IP cores summary
- LPM parameter settings
  - PLLs
  - RAMs
  - Multipliers

This information is also available in the map report (.map.rpt) in the "Parameter Settings by Entity Instance" section.

Table 6-1 shows an example of an Intel IP core conversion list.

Table 6-1: **Intel IP Core Conversion List**

Type	HDL File	Hierarchical Name	OPERATION_MODE	PLL_TYPE	PRIMARY_CLOCK	INCLK0_INPUT_FREQUENCY	INCLK1_INPUT_FREQUENCY	VCO_MULTIPLY_BY	VCO_DIVIDE_BY			
PLL	pll_0.v	pll0:inst_pll0|altpll:altpll_component	NORMAL	AUTO	INCLK0	16276	0	0	0			
PLL	pll_1.v	pll1:inst_pll1|altpll:altpll_component	NORMAL	AUTO	INCLK0	16276	0	0	0			
Type	**HDL File**	**Hierarchical Name**	**OPERATION_MODE**	**WIDTH_A**	**NUMWORDS_A**	**OUTDATA_REG_A**	**WIDTH_B**	**NUMWORDS_B**	**ADDRESS_REG_B**	**OUTDATA_REG_B**	**RAM_BLOCK_TYPE**	**READ_DURING_WRITE_MODE_MIXED_PORTS**
RAM	xy_ram.v	xy_ram:xy_ram|altsyncram:altsyncram_component	DUAL_PORT	26	16	UNREGISTERED	13	32	CLOCK1	CLOCK1	AUTO	DONT_CARE
RAM	yz_ram.v	yz_ram:yz_ram|altsyncram:altsyncram_component	DUAL_PORT	13	32	UNREGISTERED	26	16	CLOCK1	CLOCK1	AUTO	DONT_CARE
Type	**HDL File**	**Hierarchical Name**	**LPM_WIDTHA**	**LPM_WIDTHB**	**LPM_WIDTHP**	**LPM_REPRESENTATION**	**INPUT_A_IS_CONSTANT**	**INPUT_B_IS_CONSTANT**	**USE_EAB**	**DEDICATED_MULTIPLIER_CIRCUITRY**	**INPUT_A_FIXED_VALUE**	**INPUT_B_FIXED_VALUE**
Multiplier	filter1.v	filter1:thefilter1|LPM_MULT:u0_tree_mult1_component	9	17	26	SIGNED	NO	NO	OFF	YES	Bx	Bx
Multiplier	filter2.v	filter2:thefilter2|LPM_MULT:u0_tree_mult1_component	10	17	27	SIGNED	NO	NO	OFF	YES	Bx	Bx

The Quartus II report table contents can be copied and pasted, or directly exported to a CSV file for importing to a spreadsheet. To export to a CSV file, right-click the file, select "export" and select CSV format from the drop-down list.

From the previous analysis, it can be determined whether the original IP cores are static or parameterizable configurations. A static IP core configuration produces the identical IP core configuration each time it is instanced in a design. A parameterizable IP core configuration is given unique parameters for each instantiation. Consequently, each instanced parameterizable IP core is unique. The available parameters vary per IP core type. Some examples of these parameters are port data width, pipelining depth, and memory/FIFO depth. The subsequent sections in this chapter outline the conversion steps for static and parameterizable IP cores.

A column for effort estimate can be added to the IP core conversion list. When estimating the task effort, consider all aspects of the design conversion process:

- IP generation
- Adaptation of the interfaces and signals in the HDL code
- Verification of the synthesis and implementation results
- Simulation and hardware verification

Static Megafunctions

This section describes the conversion process for low-level IP cores originally configured with the Intel GUI. The Vivado® IP catalog can be used to generate equivalent or nearly equivalent replacements for this type of Intel IP core. This type of IP core can be converted using a side-by-side approach. This approach includes viewing the existing IP core configuration in the Intel GUI, and based on the settings, configuring the Vivado IP catalog IP core appropriately. The steps in this process are:

1. Identify the IP core to be converted:
   a. Identify the IP core configuration itself.
   b. Identify the RTL sections where the IP core is instantiated.
2. Open the existing IP core configuration in the appropriate Intel GUI.
3. Create a Vivado Design Suite project targeting the desired device.
4. Open the Vivado IP catalog and locate the equivalent IP core.
5. Examine the IP core configuration in the Intel GUI, and configure the Xilinx IP core appropriately. Many of the configuration option names and functions can be different. In this case, consult the appropriate Xilinx IP core product guide (PG) to determine the desired configuration.
6. Generate the Vivado Design Suite IP core.

7. Locate the instantiation template for the Vivado Design Suite IP core, and compare that to the existing IP core instantiation in your HDL code. Note any differences in port name and function. For reference, the differences in memory IP ports and clocking module ports are described in Clocking Primitive Port Differences.
8. Replace the existing IP core instantiation with the Vivado Design Suite IP core instantiation template. Replace the placeholder mapped port names in the template with the actual signal names, and make any necessary modifications to the surrounding logic. For a summary of memory IP and clocking pin differences, see Clocking Primitive Port Differences and Memory Primitive Differences.

Clocking Primitive Port Differences

Table 6-2 shows the mapping of the most important ports of an ALTPLL and a 7 series MMCM and PLL.

Table 6-2: **Comparison of Intel PLL and 7 Series MMCM and PLL Ports**

Intel PLL Ports	MMCM Ports	PLL Ports	Description
Inputs			
clkin0	CLKIN1	CLKIN1	Clock input 1
clkin1	CLKIN2	CLKIN2	Clock input 2
clkswitch	CLKINSEL	CLKINSEL	Selection of input clocks
areset	RST	RST	Asynchronous reset port
fbin	CLKFBIN	CLKFBIN	Clock feedback
phaseupdown	PSINCDEC	N/A	Phase shift increment or decrement
N/A	PSCLK	N/A	Dynamic phase shifting clock
phasestep	PSEN	N/A	Phase shift enable
Outputs			
fbout	CLKFBOUT CLKFBOUTB	CLKFBOUT	Dedicated feedback output
locked	LOCKED	LOCKED	Phase-locked status
clk[9:0]	CLKOUT[0:6] CLKOUT[0:3]B	CLKOUT[0:5]	Clock outputs
phasedone	PSDONE	N/A	Phase shift done output

For more information about the 7 series MMCM or PLL, see the *7 Series FPGAs Clocking Resources User Guide* (UG472) [Ref 16].

Memory Primitive Differences

For details on how to convert your HEX code to coefficient data, see the *Intel-to-Xilinx Memory Initialization File (HEX to COE) Conversion Answer Record* (AR#66015).

Table 6-3 shows a comparison of the single-port RAM port naming for Xilinx and Intel.

Table 6-3: **Single-Port RAM Port Naming Comparison**

Intel Port Name	Xilinx Port Name	Description
clock	clka	Port A clock
clken	ena	Port A clock enable
wren/byteena	wea	Port A write enable(1)
N/A	regcea	Port A clock enable for final output register
address	addra	Port A address
data	dina	Port A data input
q	douta	Port A data output
N/A	ssra	Port A synchronous reset

Notes:

1. Xilinx memory blocks support byte enables that mask the input data so that only specific bytes of data are written. The unwritten bytes or bits retain the previous written value. There are differences in the byte enables supported in Intel and Xilinx RAMs. Intel RAMs are controlled by write enable and byte enable signals, and Xilinx RAMs are controlled by the WEA[n:0] (or WEb[N:0]) signal.

Table 6-4 shows a comparison of the simple dual-port RAM port naming for Xilinx and Intel.

Table 6-4: **Simple Dual-Port RAM Port Naming Comparison**

Intel Port Name	Xilinx Port Name	Description
wrclock	clka	Port A clock
rdclock	clkb	Port B clock
wrclocken	ena	Port A clock enable
rdclocken	enb	Port B clock enable
wren/byteena	wea	Port A write enable(1)
N/A	regceb	Port B clock enable for final output register
wraddress	addra	Port A address
rdaddress	addrb	Port B address
data	dina	Port A data input
q	doutb	Port B data output
N/A	ssra	Port A synchronous reset

Table 6-4: **Simple Dual-Port RAM Port Naming Comparison *(Cont'd)***

Intel Port Name	Xilinx Port Name	Description
N/A	ssrb	Port B synchronous reset

Notes:

1. Xilinx memory blocks support byte enables that mask the input data so that only specific bytes of data are written. The unwritten bytes or bits retain the previous written value. There are differences in the byte enables supported in Intel and Xilinx RAMs. Intel RAMs are controlled by write enable and byte enable signals, and Xilinx RAMs are controlled by the WEA[n:0] (or WEb[N:0]) signal.

Table 6-5 applies to port naming differences for Intel clocking modes: `Dual clock: use separate clock for A and B ports` and `Single clock`. The `Dual clock: use separate 'input' and 'output' clocks` mode is not covered in this table.

Table 6-5: **True Dual-Port RAM Port Naming Comparison**

Intel Port Name	Xilinx port name	Description
clock_a(2)	clka	Port A clock
clock_b(2)	clkb	Port B clock
enable_a	ena	Port A clock enable
enable_b	enb	Port B clock enable
wren_a/byteena_a	wea	Port A write enable(1)
wren_b/byteena_b	web	Port B write enable(1)
N/A	regcea	Port A clock enable for final output register
N/A	regceb	Port B clock enable for final output register
address_a	addra	Port A address
address_b	addrb	Port B address
data_a	dina	Port A data input
data_b	dinb	Port B data input
q_a	douta	Port A data output
q_b	doutb	Port B data output
N/A	ssra	Port A synchronous reset
N/A	ssrb	Port B synchronous reset
aclr_a	N/A	Port A asynchronous reset
aclr_b	N/A	Port B asynchronous reset
addressstall_a	N/A	Port A address stall
addressstall_b	N/A	Port B address stall
rden_a	N/A	Port A read enable

Table 6-5: **True Dual-Port RAM Port Naming Comparison *(Cont'd)***

Intel Port Name	Xilinx port name	Description
rden_b	N/A	Port B read enable

Notes:

1. Xilinx memory blocks support byte enables that mask the input data so that only specific bytes of data are written. The unwritten bytes or bits retain the previous written value. There are differences in the byte enables supported in Intel and Xilinx RAMs. Intel RAMs are controlled by write enable and byte enable signals, and Xilinx RAMs are controlled by the WEA[n:0] (or WEB[N:0]) signal.
2. For the `Single clock` mode, the port name is `clock`.

Parameterizable Megafunctions

Definition

Parameterizable Intel IP (megafunctions) are IP cores that can represent a different configuration through Verilog parameters or VHDL generics.

For example:

- A RAM IP core instantiated in Verilog used multiple times with different data width, parameterized through a Verilog parameter.
- A VHDL FIFO IP core used multiple times with a different depth, configured through a VHDL generic.

When generated from the IP catalog and Parameter Editor (MegaWizard™), Intel IP cores (megafunctions) are not parameterizable – their configuration is static based on the settings specified in the Parameter Editor (MegaWizard). However, it is not uncommon for RAMs, FIFOs, and DSP functions to reuse the same IP core multiple times with different configurations through parameters of generics. Intel also provides a library of parameterized modules (LPM).

Flow

The first step in converting these parameterizable IP cores is to identify all configurations. The Analysis section showed how to find all configurations present in the design and build up a list. When there are parameterizable IP core in a design, Xilinx recommends adding the IP core parameters to this list.

The conversion of parameterizable IP core depends on the amount of variants of an IP core. You should consider short-term effort and long-term portability needs as you decide among one of the options described in this section:

- Replace Parameterizable IP With Static IP Core
- Replace Parameterizable IP With Inferred IP
- Replace Parameterizable IP With Instantiated Primitives

- Replace Parameterizable IP with Unmanaged Parameterizable Vivado IP
- Replace Parameterizable IP with XPM

Replace Parameterizable IP With Static IP Core

Use this flow when there are only a few variants of a specific IP core file used in your design.

1. Generate several static IP cores with the Vivado IP catalog to cover all variants.
2. Use generate statements to conditionally instantiate the corresponding Vivado IP core.

This is an example of the content of the wrapper that conditionally instantiates the correct configurations of the RAM:

```
entity RAM_A is
   generic (DATAWIDTH: integer := 8);
   port    (...);
end RAM_A;

architecture RTL of RAM_A is
begin

    8_BIT: if DATAWIDTH=8 generate
      RAM_A8_i: entity work.RAM_A8
      port map
         (...);
    end generate 8_BIT;

    16_BIT: if DATAWIDTH=16 generate
      RAM_A16_i: entity work.RAM_A16
      port map
         (...);
    end generate 16_BIT;
```

Typically, this method is used for FIFOs and memories with ECC, which cannot be inferred.

Replace Parameterizable IP With Inferred IP

Whenever possible, write HDL to infer memories and DSP functions. Inferring offers many advantages, such as architecture independence, fully parameterizable, and lowest simulation runtime.

Inferable code examples are available in the Vivado Design Suite language templates. These templates are for non-parameterizable code, however parameterization can be added if required.

Examples of advanced module templates that are parameterizable can be downloaded here:

www.xilinx.com/support/documentation/sw_manuals/ug1192-adv-templ.zip

These include:

- Accumulate
- Addsub
- Asynchronous simple FIFO
- Counter
- DDIO_IN
- DDIO_OUT
- Mult
- Multaccum
- Mult_add
- Shiftreg
- Synchronous simple FIFO

Replace Parameterizable IP With Instantiated Primitives

Certain IP cores cannot be inferred, such as FIFOs that use the hard FIFO or memories that use ECC. In these cases, one possible solution is to instantiate the hard primitives, and based on the parameterization possibilities, use a generate statement to cover all cases. The benefit of this method is that the results are very area-optimized. The downside is that it is laborious to create more complex variants that cover a wide range of variants.

Replace Parameterizable IP with Unmanaged Parameterizable Vivado IP

When none of the previous methods are feasible, the generated Vivado IP core can be made parameterizable. Typically, this flow is used for FIFOs and memories with ECC. For example, to add parameterization to a Vivado Design Suite-generated block memory, as shown in Figure 6-2.

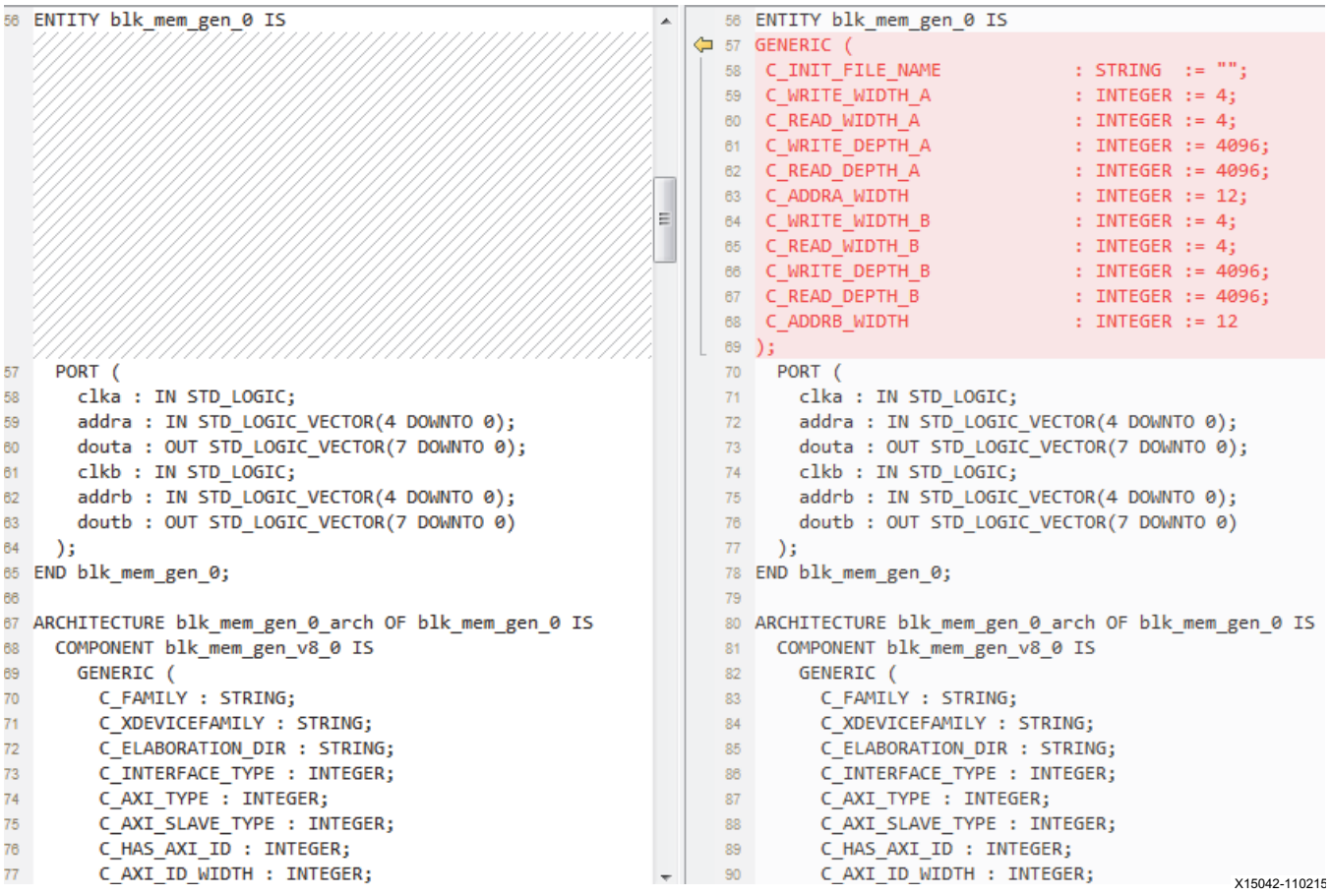

Figure 6-2: **Adding Parameterization to a Vivado Design Suite-Generated Block Memory**

IMPORTANT: *This flow must be applied with great care: 1. Parameter values are not verified during synthesis, only during IP generation. 2. Constraints files (i.e., FIFO) depend on the IP configuration – only use limited parameterization!*

This process involves generating the IP core from the Vivado IP catalog and then editing the IP core files. This process is described step-by-step in Answer Record 57546.

By referencing the Xilinx core instance (XCI) file (recommended), you have access to the IP core source files for simulation and synthesis, as well as for implementation.

Replace Parameterizable IP with XPM

Xilinx offers a Xilinx parameterized modules library (XPM) that is available as an early access solution (as of the 2015.3 release of the Vivado Design Suite) for memories. Additional XPMs are planned for the future. Contact your Xilinx FAE for more information on the latest Xilinx offerings.

RTL Design Constraints

Overview

This chapter describes how to convert Intel® Synopsys® Design Constraint (SDC) files to Xilinx® Design Constraints (XDC) files. This chapter identifies which timing constraints can be used as is, which timing constraints need a slight modification, and which timing constraints need a complete rewrite. It is assumed that you are familiar with static timing analysis, as well as the syntax used for timing constraints in the SDC format.

Review of Intel SDC File

Typically an SDC file for an Intel design includes constraints for:

- Primary clock constraints with the `create_clock` constraint.
- Input and output delay constraints with the `set_input_delay` and `set_output_delay` constraints.
- User clock uncertainty constraints with the `set_clock_uncertainty` constraint (if any).
- `derive_pll_clocks` command for MMCM/PLL generated clocks.
- `derive_clock_uncertainty` command for interclock and intraclock uncertainty constraints.
- Clock group constraints with the `set_clock_groups` command.
- Generated clock constraints with `create_generated_clock` constraint.
- Other exception constraints, such as `set_multicycle_path` and `set_false_path` commands.

The SDC commands for an Intel design might need to be modified to change the names of the individual cells, instances, or pins being referred. These changes are needed due to the difference in pin names and primitive names in the Xilinx tools and the Intel tools. The individual instance names created by the Vivado® synthesis tool might be different from the names created by the Intel tools.

Constraint Changes Not Required

The constraints for primary clocks with the `create_clock` constraint do not need any changes. The input and output delay constraints also do not need to change if the board design has not changed. If the board delay has changed, then the value of the maximum and minimum numbers must be changed to reflect the new board design.

If the design includes specific user clock uncertainty constraints with the `set_clock_uncertainty` constraint, they too can remain as is, provided the clock names have not changed.

Constraints That Are Not Required

The Intel SDC file uses the `derive_pll_clocks` constraint for deriving phase-locked loop (PLL) generated clocks, and the `derive_clock_uncertainty` constraint for interclock and intraclock uncertainties. These `derive_pll_clocks` and `derive_clock_uncertainty` commands are not needed in the XDC file. The Vivado design tools auto-derive the mixed-mode clock manager (MMCM) generated clocks.

Constraints Requiring Change

User-generated Clocks

User-generated clocks in the FPGA fabric are typically divided clocks. If an Intel SDC file includes a multiply-by option used with a value of 1, the option and the value are not needed and should be removed. These examples show the removal of the `multiply_by` option and the value 1.

Intel SDC File:

```
create_generated_clock -name div_by_2_CLK -source [get_pins
{div_by_2|FDIV|clk}] -divide_by 2 -multiply_by 1 -master_clock
{clkdiv} [get_pins {div_by_2|FDIV|q}]
```

XDC File:

```
create_generated_clock -name div_by_2_CLK -source [get_pins
{div_by_2/FDCE/C}] -divide_by 2 -master_clock {clkdiv} [get_pins
{div_by_2/FDCE/Q}]
```

Clock Group Constraints

In the Intel SDC file, the `set_clock_groups` constraint needs to specify auto-derived generated clocks from PLLs. These auto-derived clocks might have a different name in the XDC file. To include generated clocks in the clock group constraint in the Vivado design tools, use the `-include_generated_clocks` option in the constraint as shown:

From Intel SDC File:

```
set_clock_groups -asynchronous -group [get_clocks {sys_clk}] -group
[get_clocks {clk100
pll0|altpll_component|auto_generated|pll1|clk[0]}]
```

To Xilinx XDC File:

```
set_clock_groups -asynchronous -group [get_clocks {sys_clk}] -group
[get_clocks -include_generated_clocks clk100]
```

For auto-derived clocks, you can specify the clock name using the `create_generated_clock -name <your name> <object>` command. Remaining arguments to the generated clock are auto-derived by the tool.

Exception Constraints

For the exception constraints, such as false path, multicycle path, and max delay datapath only constraints, Intel has implemented non-SDC extensions, such as `get_keepers` and `get_registers`. These are not standard SDC extensions and cannot be used in the Vivado design tools. Use get_pins or get_cells instead, as shown:

Intel Constraint:

```
set_false_path -from [get_registers
{*instance1_str:*|in_data_buffer*}] -to [get_registers
{*instance2_str:*|out_data_buffer*}]

set_false_path -to [get_keepers {*std_synchronizer:*|din_s1}]
```

Xilinx Constraint:

```
set_false_path -from [get_cells -hier -filter {name ==
top/instance1/in_data_buffer && ref_name =~ FD*}] -to [get_cells
-hier -filter {name == top/instance2/out_data_buffer && ref_name =~
FD*}

set_false_path -to [get_pins  {top/modA/std_synchronizer/inst1/Q} ]
```

IMPORTANT: *For exception constraints, it is important to minimize the usage of wild cards. Try to be as specific as possible and specify valid start and end points.*

Set/Report Max Skew

Intel supports non-SDC commands, such as `set_max_skew` and `report_max_skew`. These commands are not supported in the Vivado design tools.

To constrain a skew requirement on the clock-domain-crossing (CDC) paths, use the `set_max_delay` with the `-datapath_only` option. The Xilinx Tcl Store has a `report_bus_skew` Tcl script for detecting any skew violations.

Validating Timing Constraints

After you have converted your constraints, load your design and constraints into the Vivado tool and run these commands to confirm that the constraints are correct:

```
check_timing

report_clock_interaction

write_xdc -force -constraints invalid <file name>
```

After running synthesis, check the synthesis log file for any warnings regarding timing constraints.

XILINX
ALL PROGRAMMABLE™

Chapter 8

Synthesis and Implementation

Overview

This chapter details the differences between Intel® and Xilinx synthesis and implementation commands and settings.

Synthesis

Synthesis is the process of transforming an RTL-specified design into a gate-level representation.

Language Support

The synthesis-supported language constructs for VHDL, Verilog, and SystemVerilog are roughly equivalent between the Intel Quartus II design tools and the Xilinx® Vivado® design tools. There is a small subset of Verilog features supported by Quartus II synthesis, but not by Vivado synthesis:

- Hierarchical path name references.
- Net data types of wand, wor, triand, and trior.
- Config/endconfig.

The Vivado Design Suite supports many more VHDL-2008 constructs than Quartus II. For a complete list of VHDL-2008 constructs supported by the Vivado Design Suite, see the *VHDL-2008 support in 2014.3 Answer Record* (AR#62005)

SystemVerilog files with a `*.v` extension are handled differently in the Vivado Design Suite and Quartus II design tools. In Quartus II, there is a global option to treat all `*.v` files as SystemVerilog files. In the Vivado tools, the `file_type` property is set on the `*.v` files to treat them as SystemVerilog files.

Unlike Quartus II, the Vivado synthesis tool compiles the `*.v` files with the Verilog 2005 syntax and the `*.sv` files with the SystemVerilog syntax, leading to macro (define) visibility issues between the two groups of files.

RECOMMENDED: *Xilinx recommends assigning all the files in a SystemVerilog project to the SystemVerilog file type to ensure that the header files and their macros are visible and read as intended.*

Directives

Xilinx recommends the following flow to convert directives:

1. Analyze and identify all directives. Directives can be identified using:

   ```
   "find in files" for "(*" (verilog) or "attribute" (VHDL)
   ```

2. Create a table of directives with their name and location.
3. Find the equivalent directive using Table 8-1.
4. Modify the HDL source code.

Note: Vivado synthesis does not report unsupported directives. If the specified attribute is not recognized by the tool, Vivado synthesis passes the attribute and its value to the generated netlist.

Table 8-1 lists the Quartus II directives and their equivalent in the Vivado Design Suite.

Table 8-1: **Mapping Quartus II and Vivado Attributes**

Quartus II Attribute	Equivalent Vivado Attribute(1)	Description
altera_attribute	*Use XDC or Tcl*	Make assignments to objects
chip_pin	PACKAGE_PIN	Assign device pins to a port
direct_enable	DIRECT_ENABLE	Force signal as enable of flop
dont_replicate	DONT_TOUCH	Prevent register replication
dont_retime	DONT_TOUCH	Prevent re-timing
dont_merge	KEEP/DONT_TOUCH	Prevent register merging
enum_encoding (VHDL)	fsm_encoding	Specify FSM enumeration type
full_case (verilog)	full_case (verilog)	Indicate that all possible case values are specified
keep	KEEP	Prevent synthesis optimizations
library	*Use Tcl or project settings*	Set destination library for design unit
max_depth	*Not supported*	Specify maximum depth of inferred memory blocks
maxfan	max_fanout	Drive replication to respect maximum fanout
multstyle	mult_style	Multiplier style

Table 8-1: **Mapping Quartus II and Vivado Attributes *(Cont'd)***

Quartus II Attribute	Equivalent Vivado Attribute[1]	Description
noprune	DONT_TOUCH	Prevent floating logic optimization
parallel_case	(* parallel_case *)	Build case statement as parallel structure
preserve	DONT_TOUCH	Prevent redundant or constant driver optimization
ram_init_file	*See "Specifying RAM Initial Contents in an External Data File" in UG901*	Specify initial content of memory
ramstyle	ram_style	Specify type of inferred RAM
read_comments_as_HDL	*Not supported*	Perform synthesis on portions code in comments
romstyle	rom_style	Specify type of inferred ROM
syn_encoding (verilog)	fsm_encoding	Specify FSM enumeration type
synthesis translate_off	synthesis translate_off	Synthesis ignore portion of code
synthesis translate_on	synthesis translate_on	Synthesis enable again
useioff	IOB	Implement I/O connected registers in I/O block
verilog_input_version	*Use HDL file property in Vivado project*	Specify verilog version (VERILOG_1995, VERILOG_2001, or SYSTEMVERILOG_2005)
vhdl_input_version	*Use HDL file property in Vivado project*	Specify VHDL version (VHDL_1987, VHDL_1993 or VHDL_2008)

Notes:

1. For more information on the Vivado attributes, see the *Vivado Design Suite User Guide: Synthesis* (UG901) [Ref 14] and the *Vivado Design Suite Properties Reference Guide* (UG912) [Ref 19].

Settings

Table 8-2 lists the Quartus II compiler optimization modes and their equivalent Vivado synthesis strategies.

Table 8-2: **Mapping Quartus II Compiler Optimization Mode and Vivado Synthesis Strategies**

Quartus II Compiler Optimization Mode	Vivado Synthesis Strategy
Balanced (normal flow)	Vivado synthesis defaults
Performance (high effort – increases runtime)	Flow_PerfOptimized_High
Performance (aggressive – increases runtime and area)	Flow_PerfThresholdCarry
Power (high effort – increases runtime)	N/A for synthesis
Power (aggressive – increases runtime, reduces performance)	N/A for synthesis
Area (aggressive – reduces performance)	Flow_AreaOptimized_high Flow_AreaOptimized_medium Flow_AreaMultThresholdDSP

Table 8-3 lists the Quartus II register optimizations and their equivalent Vivado synthesis settings.

Table 8-3: **Mapping Quartus II Register Optimization and Vivado Synthesis Settings**

Quartus II Prevent Register Optimization	Vivado Synthesis Options
Prevent register merging	`-keep_equivalent_registers`
Prevent register duplication	`-fanout_limit<n>`, where `<n>` is a high numerical value. For example, `100000`.
Prevent register retiming	N/A for synthesis

Resource Utilization

The verification of the post-synthesis resource utilization is the key to correct design conversion. It can be difficult to compare the resource utilization of the converted complex IP to the original IP. However, RTL and inferred logic should be very comparable in utilization, when compared to the original design. First, a comparison of the top-level utilization metrics should be performed (see Table 8-4). Depending on the result of this comparison, a more in-depth hierarchical comparison might need to be completed.

For an overall comparison, you can populate a utilization comparison table such as Table 8-4.

Table 8-4: **Sample Utilization Comparison Table**

	Synthesis Resource Estimate						
	Logic LUTs	Mem LUTs	Total LUTs	FFs	M20K/ RAMB18	RAMB36	DSPs
Quartus		N/A			N/A		
Vivado							
	Implementation Resource Utilization						
Quartus							
Vivado							

A quick mapping of the terminology, derived from the reports generated by the tools, should be used to fill the utilization table shown in Table 8-4.

- Quartus

  Utilization figures can be gathered from these locations:

  - Analysis and synthesis - "Resource Usage Summary" report
    - Logic LUTs = total LUTs: "Combinational ALUT Usage for Logic"
    - FFs: "Dedicated Logic Registers"
    - DSP: "Total DSP Blocks"

- Fitter - “Resource Usage Summary” report
  - Logic LUTs : “Combinational ALUT Usage for Logic”
  - Mem LUTs: “Memory ALUT Usage”
  - Total LUTs: “Logic LUTs + Mem LUTs”
  - FFs: “Dedicated Logic Registers”
  - RAMB18/M20K: “M20K Blocks”
  - DSP: “Total DSP Blocks”
- Vivado synth_design - “Utilization” report
  - Logic LUTs: "LUT as Logic"
  - Mem LUTs: "LUT as Memory"
  - Total LUTs: "Slice LUTs"
  - FFs: "Slice Registers"
  - DSP: "DSPs"
  - RAMB18: "RAMB18"
  - RAMB36: "RAMB38/FIFO"

Note: The synth design utilization report does not include any out-of-context (OOC) IP utilization figures. To circumvent this difference, either set the synthesis options for IP to global or regenerate the utilization report (report_utilization) from the "synthesized design" view.

When the table is populated, check for large anomalies:

- Logic LUTs:
  - Xilinx LUT6 count might be about 10% to 20% less than the Intel ALUT count.
  - Differences can be due to IP, otherwise hierarchical utilization comparisons are required.
- Registers:
  - Should be comparable, differences should be further investigated.
  - Differences can be due to asynchronous resets: Xilinx DSPs, SRLs, and block RAMs do not support asynchronous resets.
  - Unrecognized RAM templates can cause RAMs inferred as large register-arrays.
- Mem LUTs + RAMB18/M20K+RAMB36
  - Differences can be:
    - Due to data-width: for 20-bit wide RAMs, RAMB36 are required.

  - Due to difference in inference threshold between distributed RAM versus block RAM.
  - Unrecognized RAM templates can cause RAMs inferred as large register-arrays.

- DSPs:
  - Investigate any differences, see DSP in Chapter 2 for differences in the architecture.

A much more detailed comparison can be performed using hierarchical utilization reports. For the purpose of this hierarchial utilization, you should retain the hierarchies, by setting the appropriate option in the Vivado synthesis. However, retaining hierarchies results in a sub-optimal netlist. Consequently, after you are sure of your recoding, run again without retaining the hierarchies to generate an optimal netlist.

RECOMMENDED: *Xilinx recommends exporting the reports to a* `*.csv` *file format. The* `*.csv` *files can be imported into spreadsheets and can be easily compared, or can be merged together for more advanced comparison.*

To gather the hierarchical utilization report from Quartus II, export the "Resource Utilization by Entity" by right-clicking on the report in the "Compilation Report" - "Fitter" section. It is not recommended to use the "resource utilization" report from "analysis & synthesis" because it is less detailed and misses information (i.e., memories).

A hierarchical utilization report is not available by default in the Vivado Design Suite, but it can be generated by opening the synthesized design and issuing this command:

```
report_utilization -hierarchical -file <filename>
```

This report is in text format and can be imported into a spreadsheet by using the '|' character as a separator. Alternatively, the Vivado Design Suite utilization report can be generated by exporting the hierarchy table by first disabling percentages (when enabled) and then right-clicking in the table and selecting **Export to Spreadsheet**.

Implementation

Implementation is the process of optimizing, placing, and routing the primitives of a synthesized netlist. This section details the differences between Intel and Xilinx implementation commands and settings.

Implementation Commands

Table 8-5 lists the Quartus II and the equivalent Vivado Design Suite implementation commands.

Table 8-5: **Mapping Quartus II and Vivado Implementation Commands**

Quartus II Command	Vivado Command	Task/ Activity(1)
quartus_map	synth_design	Synthesis
N/A	opt_design (optional)	Optimization
quartus_fit	place_design	Placement
	phys_opt_design (optional)	Post-placement physical optimization
	route_design	Routing
N/A	phys_opt_design (optional)	Post-route physical optimization

Notes:
1. For more information, see *Vivado Design Suite User Guide: Implementation* (UG904) [Ref 20].

Settings

The Vivado tools support a pushbutton flow, as well as the sweeping of strategies. Runs can be set per your requirements and can be launched in parallel or sequentially, based on the number of compute resources available.

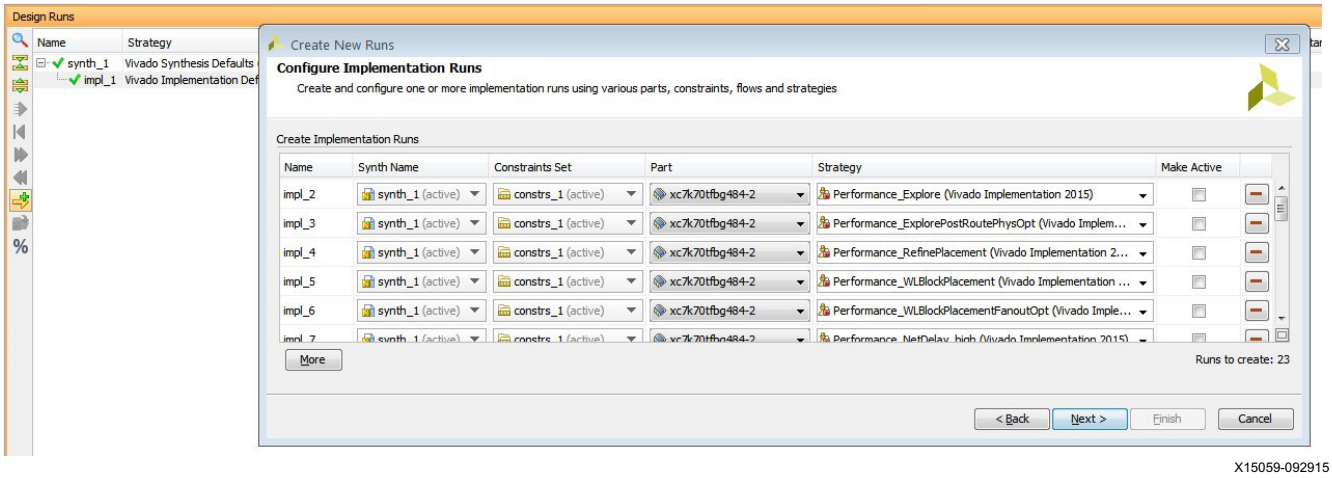

Figure 8-1: **Avalon ST Interface**

Table 8-6 lists the Quartus II compiler optimization modes and the equivalent Vivado Design Suite implementation strategies.

Table 8-6: **Mapping Quartus II Compiler Optimization Mode and Vivado Implementation Strategies**

Quartus II Compiler Optimization Mode	Vivado Implementation Strategy
Balanced (Normal flow)	Vivado implementation defaults
Performance (High effort – increases runtime)	Performance_Explore
Performance (Aggressive – increases runtime and area)	Performance_ExplorePostRoutePhysOpt
Power (High effort – increases runtime)	Power_DefaultOpt
Power (Aggressive – increases runtime, reduces performance)	
Area (Aggressive – reduces performance)	Area_Explore

Table 8-7 lists the Quartus II register optimizations and the equivalent Vivado Design Suite implementation settings.

Table 8-7: **Mapping Quartus II Register Optimization and Vivado Implementation Settings**

Quartus II Prevent Register Optimization	Vivado Implementation Options
Prevent register merging	• Avoid `opt_design` with sequential resynthesis • Default is without sequential resynthesis
Prevent register duplication	Disable `Phys_opt_design`
Prevent register re-timing	• Avoid `Phys_opt_design` with re-timing • Default is without re-timing

Note: There is no equivalence for the "Fitter Initial Placement Seed" in the Vivado Design Suite. Only the ISE® design tools offer the similar cost-tables for MAP and PAR. The Vivado Design Suite counterpart of the Quartus "Design Space Explore" is the "Design Runs" window, where multiple runs can be set up.

Verification

Overview

This chapter describes the Xilinx verification tools and compares them to the Intel® tools.

Simulation

The Xilinx® Vivado® design suite offers the most comprehensive solution for simulation in the FPGA industry. All of the Vivado Design Suite editions (including the WebPACK™ edition) come with an integrated full-featured mixed language simulator to address your verification needs.

The Vivado simulator has no line or instance limitations. It supports these third-party simulators for both integrated (pushbutton flow) and script-based flow:

- Mentor Graphics® Questa® Simulator (QuestaSim) or ModelSim
- Cadence® Incisive Enterprise Simulator (IES)
- Synopsys® VCS®
- Aldec® Active-HDL™ or Riviera-PRO™

Vivado Simulator

The Vivado simulator is a hardware description language (HDL) event-driven simulator that supports behavioral and timing simulations for:

- VHDL
- Verilog
- SystemVerilog
- Mixed VHDL/Verilog or VHDL/SystemVerilog designs

Feature Highlights

- SystemVerilog
- Verilog-2001
- VHDL-93
- Standard delay format (SDF) 3.0 for timing simulation
- Switching activity interchange format (SAIF) for power analysis
- Value change dump (VCD)

Licensing

There is a fundamental difference between the Quartus II simulator licensing and the Vivado simulator licensing. The Vivado (including WebPACK) simulator features do not change based on the license you have.

The Vivado simulator is not only used for unit and system-level simulation. The Vivado simulator is widely deployed by verification engineers for regressions because it enables the user to run multiple simulation combinations allowed by their infrastructure using a single license.

Table 9-1: **Product Comparison**

Product Details	Xilinx Vivado Simulator	ModelSim-Intel Started Edition	ModelSim-Intel Edition
Price	Included with Vivado IDE	Free no license required	Paid
Design size support	No limitations	Small designs 10,000 executable line limit	Up to 3,000 instances
Mixed language support	Yes	Yes	Yes
Operating system support	Windows, Linux	Windows, Linux	Windows, Linux
SystemVerilog DPI	Supported	Not supported	Not supported
SystemVerilog dynamic types	Supported	Not supported	Not supported
Download and setup	No additional setup needed	Download a separate simulator	Download a separate simulator

These Xilinx user guides and videos provide detailed information about the Vivado simulator:

- For more information on the simulation process, and the simulation options in the Vivado Design Suite, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 21].
- For more details on the simulator, see the Vivado Simulator website.

Vivado Simulation Flow

The Vivado Design Suite enables you to launch any of the four supported third-party simulators from the design cockpit (Vivado IDE), and also provides Tcl APIs to generate scripts to run simulation.

Feature Highlights

- Simulate using all five supported simulators using a single click in the Vivado IDE
- Xilinx employs IEEE P1735 encryption flow for RTL IP source
  - The same file is used for synthesis and simulation
  - Accurate and fast simulation with fewer files
- Generate simulation scripts at IP generation
- Unified support across all supported simulators
  - Can move from one simulator to another easily
- Support for Cadence® irun flow

Bitstream Generation and Flash Creation

The Vivado Design Suite provides `write_bitstream` and `write_cfgmem` commands to generate programming files and flash files, respectively. These commands are equivalent to the `quartus_asm` executable.

These Xilinx user guides and videos provide detailed information about the `write_bitstream` and the `write_cfgmem` commands:

- For more information on what Tcl commands are available in the Vivado Design Suite, see the *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 12].
- To get started learning about the Vivado Design Suite, see the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 22].

VIDEO: *For more information on how to set, list, or report device configuration properties for a bitstream using the Vivado IDE and Tcl commands, see the* How to Use the "write_bitstream" Command in Vivado *video.*

VIDEO: *For more information on how to use the new configuration dialog box to set and edit device properties, see the* Setting and Editing Device Properties in Vivado *video.*

Note: When using the Vivado design tools, the bitstream (`*.bit`) files are generated automatically as part of the flow.

Programming the Device

The Vivado Design Suite includes the Vivado Hardware Manager in all editions. The Vivado Lab Edition is a standalone collection of all the tools required to program the Xilinx devices.

These Xilinx user guides and videos provide detailed information about the Hardware Manager used for programming:

- To learn about the Vivado Design Suite, see the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 22].

VIDEO: *For more information about the features and benefits of Vivado Lab Edition, and to become familiar with its installation and typical use flows, see the* Using Vivado Lab Edition *video.*

VIDEO: *For more information on how to use the Vivado device programmer to create and configure a configuration memory device, see the* Indirectly Program an FPGA using Vivado Device Programmer *video.*

VIDEO: *For more information on how to set correct properties for generating an encrypted bitfile, see the* Using Advanced Encryption Standard Keys with the Battery-Backed (BBR) RAM *video.*

Hardware Debug

The Vivado Design Suite offers many solutions for debugging your designs quickly and effectively while they are running in the hardware. These solutions include tools, IP cores, and flows that mostly match the same features and capabilities provided in Quartus II.

The Vivado Design Suite provides a unified design environment that enables users to perform different debug tasks within the same IDE, in which interfaces look consistent and features communicate well between each other.

The sections listed here describe the main Quartus II verification tools and their equivalent features in Vivado.

- System Console/Vivado Hardware Manager
- Logic Debug
- Transceiver Debug
- Memory Calibration Debug
- Other Intel Tools

System Console/Vivado Hardware Manager

The Vivado Hardware Manager facilitates device programming and debugging after a bitstream (`*.bit`) file is generated. Similar to the System Console, the hardware manager allows you to connect and program hardware targets containing one or more FPGA devices, and then interact with the debug IP cores within the FPGA designs via the Tcl console or other GUI interfaces, including the logic analyzer, serial I/O analyzer, and memory calibration debug tool.

The Hardware Manager can be used in conjunction with the JTAG-to-AXI IP core for transaction-level interaction with the AXI interfaces within a system. The JTAG-to-AXI IP core can be used in the Vivado IP Integrator (IPI) block diagram design or directly instantiated in design HDL (from the IP core catalog). The JTAG-to-AXI IP core is a customizable core that can generate the AXI transactions and drive the AXI signals internal to the FPGA in the system.

Other features in the Hardware Manager include:

- Device programmer for FPGA, configuration memory devices, eFUSE AES key/registers
- Access to system monitor (SYSMON) – ADC and on-chip sensors
- Scripting support via the Tcl console for debug automation
- Remote debugging over the network
- Support for Xilinx virtual cable (XVC)

These Xilinx user guides and videos provide detailed information about the Hardware Manager:

- For more information on available Tcl commands in the Vivado Design Suite, see the *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 12].
- To get started learning about the Vivado Design Suite, see the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 22].
- For more information on the JTAG to AXI Master customizable core, see the *JTAG AXI Master v1.0 Product Guide* (PG174) [Ref 23].

VIDEO: *For more information on how to set correct properties for generating an encrypted bitfile, see the* Using Advanced Encryption Standard Keys with the Battery-Backed (BBR) RAM *video.*

VIDEO: *For more information on how to set, list or report device configuration properties for a bitstream using the Vivado IDE and Tcl commands, see the* How to Use the "write_bitstream" Command in Vivado *video.*

VIDEO: *For more information on how to use Vivado device programmer to create and configure a configuration memory device, see the* Indirectly Program an FPGA using Vivado Device Programmer *video.*

VIDEO: *For more information on how to use the new JTAG to AXI Master feature in Vivado, see the* Using JTAG to AXI Master in Vivado *video.*

Logic Debug

Xilinx provides various tools and IP cores to help you debug your hardware system while the system is operating.

- **Intel SignalTap™ II Logic Analyzer/Xilinx Integrated Logic Analyzer (ILA):** used for basic/advanced trigger and capture of internal signals. Using HDL instantiation or netlist insertion (after synthesis), you can add ILA to your design. Netlist insertion for ILA is supported via the IDE (setup debug wizard, debug window) or Tcl.
- **Intel In-system Sources and Probes Editor/Xilinx Virtual Input Output (VIO):** used for simple monitoring and driving of internal signals. Using HDL instantiation only, you can add VIO to your design. After device programming, you can then interact with ILA or VIO by using the Vivado logic analyzer tool in the Hardware Manager. The Hardware Manager offers customizable dashboards to display all status and control information pertaining to a given debug IP. For taking measurements and capturing data, you can configure and define triggers and analyze the captured data in the waveform viewer.

These Xilinx user guides and videos provide detailed information about the Vivado logic analyzer tool:

- For more information on programming and debugging options in the Vivado Design Suite, see the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 22].
- For more information on available Tcl commands in the Vivado Design Suite, see the *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 12].
- For more information on the customizable Integrated Logic Analyzer (ILA) IP core, see the *Integrated Logic Analyzer (ILA) v5.1 Product Guide* (PG172) [Ref 24].
- For more information on the virtual input/output (VIO) core that can both monitor and drive internal FPGA signals in real time, see the *Virtual Input/Output (VIO) v3.0 Product Guide* (PG159) [Ref 25].
- For more information on the JTAG to AXI master customizable core, see the *JTAG AXI Master v1.0 Product Guide* (PG174) [Ref 23].

VIDEO: *For more information about the logic debug features in the Vivado tools, how to add logic debug IP to a design, and how to use the Vivado logic analyzer to interact with logic debug IP, see the* Logic Debug in Vivado *video.*

VIDEO: *For more information about the new dashboard improvements introduced in Vivado Design Suite 2015.1, how to use them in the Vivado logic analyzer, and the benefits of using them, see the* Using New Dashboards in Vivado Logic Analyzer *video.*

VIDEO: *For more information on how to use the Vivado tools to debug at and around device startup, see the* Debugging at Device Startup *video.*

Transceiver Debug

The FPGA transceivers are highly customizable. The transceiver parameters must be set correctly to attain the desired characteristics and performance.

Intel Transceiver Toolkit/Xilinx IBERT: offers a fast and easy solution for debugging and optimizing FPGA transceivers. This solution includes a customizable debug IP (IBERT) and the Vivado serial I/O analyzer tool. Used together, you can take bit-error ratio (BER) measurements on multiple channels, perform 2D eye scans, and adjust transceiver parameters in real-time while your serial I/O channels interact with the rest of the system.

Designed for PMA evaluation and demonstration of transceivers, IBERT also includes data pattern generators and checkers, as well as access to transceivers DRP ports. After IBERT is implemented within the FPGA, the Vivado serial I/O analyzer interacts with the IP and allows you to create links (analogous to a channel on a board) and analyze the margin of the links by running scans and viewing the results graphically.

These Xilinx user guides and videos provide detailed information about the Vivado logic analyzer tool:

- For more information on Programming and Debugging options in the Vivado Design Suite, see the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 22].
- For more information on what Tcl commands are available in the Vivado Design Suite, see the *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 12].
- For more information on the customizable integrated bit error ratio tester (IBERT) core for 7 series FPGAs GTP transceivers, see the *IBERT for 7 Series GTP Transceivers v3.0 Product Guide* (PG133) [Ref 26].
- For more information on the customizable integrated bit error ratio tester (IBERT) core for 7 series FPGAs GTX transceivers, see the *IBERT for 7 Series GTX Transceivers v3.0 Product Guide* (PG132) [Ref 27].
- For more information on the customizable integrated bit error ratio tester (IBERT) core for 7 series FPGAs GTH transceivers, see the *IBERT for 7 Series GTH Transceivers v3.0 Product Guide* (PG152) [Ref 28].
- For more information on the customizable integrated bit error ratio tester (IBERT) core for 7 series FPGAs GTZ transceivers, see the *IBERT for 7 Series GTZ Transceivers v3.0 Product Guide* (PG171) [Ref 29].

VIDEO: *For more information on how to use the new integrated Vivado serial I/O analyzer, see the* Using Vivado Serial I/O Analyzer *video.*

Memory Calibration Debug

The delay through board traces is not fixed due to the changing physical environment of the board. Consequently, a calibration step is required for the memory controller (on the FPGA) to be able to exchange data reliably with the external memory.

Intel External Memory Interface Toolkit/Xilinx Memory Calibration Debug Tool: allows users to quickly debug calibration or data errors in UltraScale devices memory interfaces (DDR4/3, RLDRAM3, and QDRII+). You can always view and analyze core configuration, calibration status, and data margin of the memory interfaces for read and write at any time throughout operation in hardware.

These Xilinx user guides, answer record, and videos provide detailed information about the Vivado logic analyzer tool:

- To get started learning about the Vivado Design Suite, see the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 22].
- For more information on what Tcl commands are available in the Vivado Design Suite, see the *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 12].
- For more information on how to use the MIG example design and enable the debug feature, see the *UltraScale MIG DDR4/DDR3 - Hardware Debug Guide* (AR60305).
- For more information about the memory interface solutions (MIS) core that is a combined pre-engineered controller and physical layer (PHY) for interfacing UltraScale architecture-based FPGAs user designs to DDR3 and DDR4 SDRAM, QDR II+ SRAM, and RLDRAM 3 devices, see the *UltraScale Memory Interface Solutions v6.1 Product Guide* (PG150) [Ref 30].

Other Intel Tools

- **SignalProbe** - no equivalent tool; however, by using Tcl commands in the Vivado Design Suite, you can easily connect any selected signal in the design to an external pin.
- **In-System Memory Content Editor** - no equivalent tool.
- **Logic Analyzer Interface Editor** (for connection to an external logic analyzer) - no equivalent tool.

Appendix A

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

Solution Centers

See the Xilinx Solution Centers for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

Documentation Navigator and Design Hubs

Xilinx Documentation Navigator provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open the Xilinx Documentation Navigator (DocNav):

- From the Vivado IDE, select **Help > Documentation and Tutorials**.
- On Windows, select **Start > All Programs > Xilinx Design Tools > DocNav**.
- At the Linux command prompt, enter `docnav`.

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In the Xilinx Documentation Navigator, click the **Design Hubs View** tab.
- On the Xilinx website, see the Design Hubs page.

Note: For more information on Documentation Navigator, see the Documentation Navigator page on the Xilinx website.

References

These Xilinx documents provide supplemental material useful with this guide:

1. *UltraScale Architecture SelectIO Resources User Guide* (UG571)
2. *IBERT for UltraScale GTH Transceivers LogiCORE IP Product Guide* (PG173)
3. *IBERT for UltraScale GTY Transceivers LogiCORE IP Product Guide* (PG196)
4. *UltraScale Architecture Clocking Resources User Guide* (UG572)
5. *UltraScale Architecture Memory Resources User Guide* (UG573)
6. *Kintex UltraScale and Virtex UltraScale FPGAs Packaging and Pinouts Product Specification User Guide* (UG575)
7. UltraScale Architecture PCB Design User Guide (UG583)
8. *Vivado Design Suite User Guide: Getting Started* (UG910)
9. *Vivado Design Suite User Guide: Design Flows Overview* (UG892)
10. *Vivado Design Suite Tutorial: Design Flows Overview* (UG888)
11. *Vivado Design Suite User Guide: Using the Vivado IDE* (UG893)
12. *Vivado Design Suite Tcl Command Reference Guide* (UG835)
13. *UltraFast Design Methodology Guide for the Vivado Design Suite* (UG949)
14. *Vivado Design Suite User Guide: Synthesis* (UG901)
15. *AXI Reference Guide* (UG761)
16. *7 Series FPGAs Clocking Resources User Guide* (UG472)
17. *Xilinx 7 Series and Zynq-7000 All Programmable SoC Libraries Guide for HDL Designs* (UG768)
18. *UltraScale Architecture Libraries Guide* (UG974)
19. *Vivado Design Suite Properties Reference Guide (*UG912)
20. *Vivado Design Suite User Guide: Implementation (*UG904)
21. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
22. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
23. *JTAG AXI Master v1.0 Product Guide* (PG174)
24. *Integrated Logic Analyzer (ILA) v5.1 Product Guide* (PG172)
25. *Virtual Input/Output (VIO) v3.0 Product Guide* (PG159)
26. *IBERT for 7 Series GTP Transceivers v3.0 Product Guide* (PG133)

27. *IBERT for 7 Series GTX Transceivers v3.0 Product Guide* (PG132)
28. *IBERT for 7 Series GTH Transceivers v3.0 Product Guide* (PG152)
29. *IBERT for 7 Series GTZ Transceivers v3.0 Product Guide* (PG171)
30. *UltraScale Memory Interface Solutions Product Guide* (PG150)
31. *Zynq-7000 All Programmable SoC Packaging and Pinout Product Specification* (UG865)
32. *7 Series FPGAs SelectIO Resources User Guide* (UG471)
33. *7 Series FPGAs Packaging and Pinout Product Specification* (UG475)
34. *Zynq-7000 All Programmable SoC Overview* (DS190)
35. *Zynq-7000 All Programmable SoC Technical Reference Guide* (UG585)
36. *UltraScale Architecture System Monitor User Guide* (UG580)
37. *7 Series FPGAs and Zynq-7000 All Programmable SoC XADC Dual 12-Bit 1 MSPS Analog-to-Digital Converter User Guide* (UG480)
38. *7 Series FPGAs PCB Design Guide* (UG483)
39. *Zynq-7000 All Programmable SoC PCB Design Guide* (UG933)
40. *PetaLinux SDK User Guide: Getting Started Guide* (UG977)
41. *PetaLinux Tools Documentation: PetaLinux Command Line Reference* (UG1157)
42. *Zynq-7000 All Programmable SoC: Embedded Design Tutorial* (UG1165)
43. *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994)
44. *Xilinx Software Development Kit (SDK) User Guide: System Performance Analysis* (UG1145)
45. *System Performance Analysis of an All Programmable SoC Application Note* (XAPP1219)
46. *Zynq-7000 All Programmable SoC Software Developers Guide* (UG821)

Please Read: Important Legal Notices

in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos.

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.