VexRiscv SoC Designs

IP User Guide (Beta Release)

April 11, 2023

RapidSilicon

Contents

SoC Summary

Introduction

The reference designs included in the Raptor Design Suite utilize the bundled IPs to form a complete SoC subsystem that can both be simulated or uploaded to an FPGA. A RISC-V compliant processor sits at the heart of this SoC that is responsible for doing all the computations in transforming RISC-V instructions into valid control signals for the attached peripherals. The SoC is interconnected with all the independent IPs in Verilog HDL for easy manipulation. A ROM can be embedded with the processor that can store all the instructions required for the processor to continue its operations. Other IPs can either be embedded or connected as peripherals to make a customizable SoC that is specific to the needs of the task at hand.

Features

- Customizable design utilizing the IPs from IP Catalog.
- Customizable RISC-V processor.
- Support for bare-metal firmwares to be run on the SoC.
- Support for Linux boot.
- Support for easy manipulation and human readable interconnections.
- Customizable and Scalable SoC solutions.
- Support for RV32IM instruction set.
- Customizable IPs to be connected within the SoC.
- Support for both cached and cacheless transactions.
- Support for internal and external interrupts from IPs.

Overview

VexRiscv SoC

The IPs included in the Raptor Design Suite are written in such a manner that they can be interconnected with each other using appropriate conversion bridges to achieve greater functionality that is far beyond than that of just one individual IP. These IPs can be cohesively interconnected with a CPU at the heart forming an SoC that is a complete subsystem in itself. This SoC can be utilized to test any number of combinations of IPs by embedding them within the SoC. The test cases are written with respect to RISC-V ISA and the instructions are then stored within an embedded memory close to the processor. The VexRiscv processor is then responsible for translating the instructions into valid transactions on the attached peripherals. A block diagram of a general SoC can be seen in the figure 1.

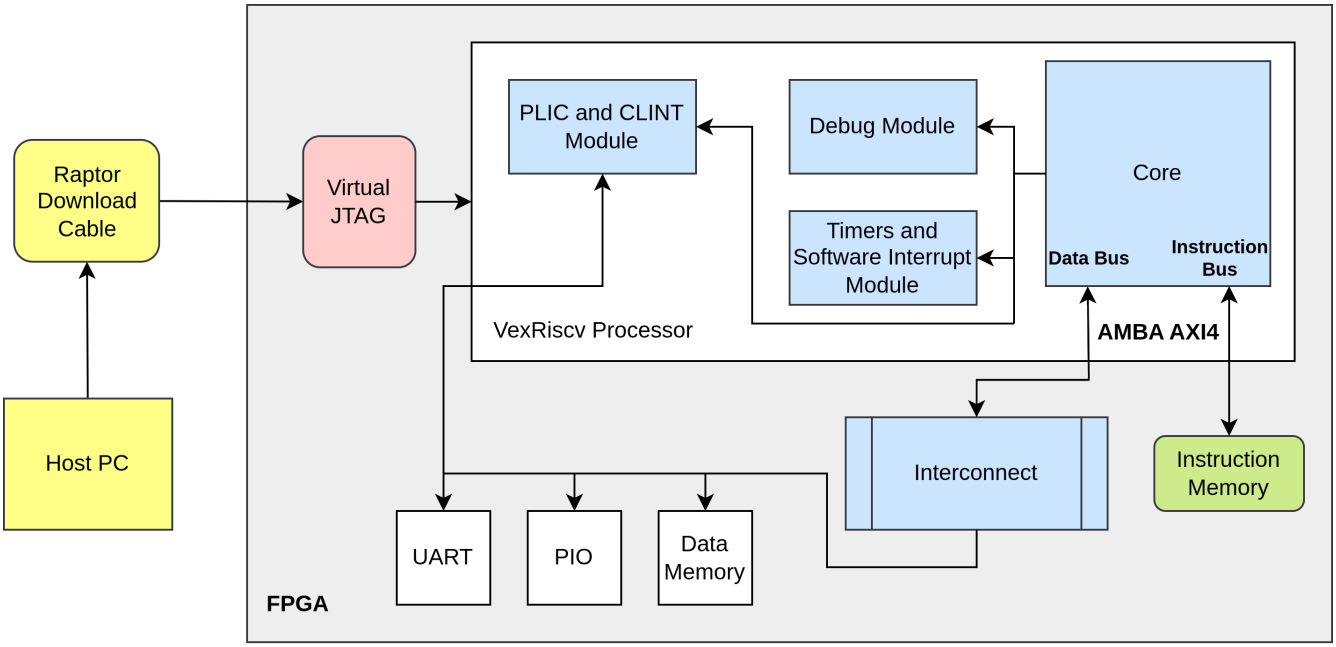

Figure 1. VexRiscv SoC Block Diagram

Licensing

COPYRIGHT TEXT:

IP Specification

The Raptor Design Suite contains three reference designs that highlight the usability and the customizability of the SoC designs. The detail of these designs can be read below: -

- **VexRiscv with AXI RAM**
  This model consists of a VexRiscv CPU in an AXI4 implementation, connected to an AXI Block RAM as a peripheral via an AXI interconnect. The CPU loads instructions from another AXI memory connected directly to the CPU instruction bus, without any interconnect, as an embedded memory serving the purpose of a ROM. The VexRiscv used here is a cacheless variant of the processor. This CPU is responsible for performing all the desired operations on connected peripherals. The AXI interconnect provides the addressing configuration for the attached peripherals and a way for traffic to flow to and fro between the CPU and the peripherals. Two instances of AXI4 Block Ram are used in this model. One connected directly to the CPU with the IBus to serve the purpose of ROM. This ROM is loaded with a hex file which contains the instructions for the CPU. The second instance is used as a peripheral for the system, the CPU attempts read and write accesses to this Ram. A pictorial representation of this model can be seen in Figure 2.

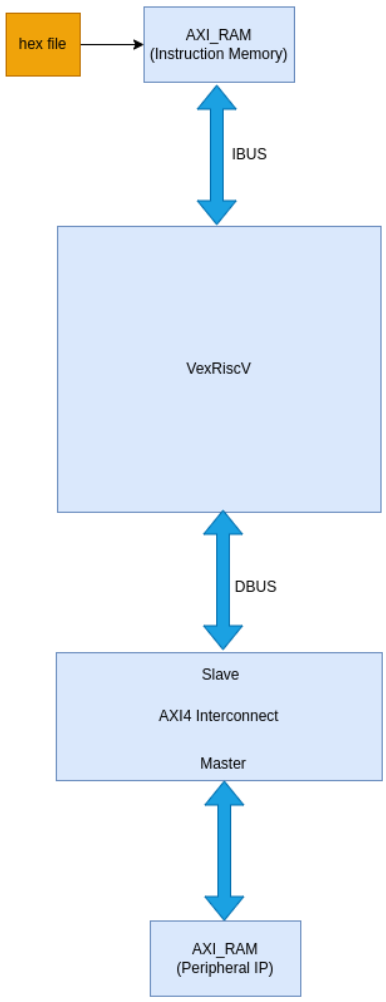

Figure 2. VexRiscv SoC with an AXI RAM

- **VexRiscv with AXI CDMA**
  This model consists of a VexRiscv CPU in an AXI4 implementation, connected to an AXICDMA as a peripheral via an AXILite interconnect communicating with the CPU with an AXI2AXILite bridge and AXICDMA connected with two different AXI BLOCK RAM via an AXI interconnect. The CPU loads instructions from another AXI memory connected directly to the CPU instruction bus, without any interconnect, as an embedded memory serving the purpose of a ROM. The VexRiscv used here is a cacheless variant of the processor. This CPU is responsible for performing all the desired operations on connected peripherals. The AXI interconnect provides the addressing configuration for the attached peripherals and a way for traffic to flow to and fro between the CPU and the peripherals. To access AXILite Peripherals that do not have support for the complete AXI4 protocol, an AXI2AXILite bridge is utilized that translates the AXI4 transactions into AXILite transactions with minimal performance overhead. It does so by utilizing a FIFO to store the calculated addresses of the burst transactions of the AXI4 protocol and executing them in an AXILite fashion. To access AXILite Peripherals that do not have support for the complete AXI4 protocol, an AXI2AXILite bridge is utilized that translates the AXI4 transactions into AXILite transactions with minimal performance overhead. It does so by utilizing a FIFO to store the calculated addresses of the burst transactions of the AXI4 protocol and executing them in an AXILite fashion. It is parameterized in a 1x2 configuration. A UART is connected as the second peripheral to the CPU and this UART generates five types of interrupts based on the state and operation of UART. This UART is connected in a loopback sense so as to check the data validity by receiving the same data as was transmitted. This serves as the Address Decoder to remove the offset of AXI interface for the CDMA connected to this Address Decoder. This serves to provide a high-speed data transfer mechanism between different memory spaces in FPGA designs, with the added benefit of being free to use, modify and distribute. AXI CDMA can improve system performance by reducing CPU overhead and enabling efficient data movement in a wide range of FPGA applications. This serves as the interconnect between CPU and peripherals, this design is based on a 1x2 Interconnect configuration that is connected ahead of the CDMA. Three instances of AXI4 Block Ram are used in this model. One connected directly to the CPU with the IBus to serve the purpose of ROM. This ROM is loaded with a hex file which contains the instructions for the CPU. The other two instance is used as a peripheral for the system, the AXICDMA attempts read and write accesses to these RAMs. A pictorial representation of this model can be seen in Figure 3.

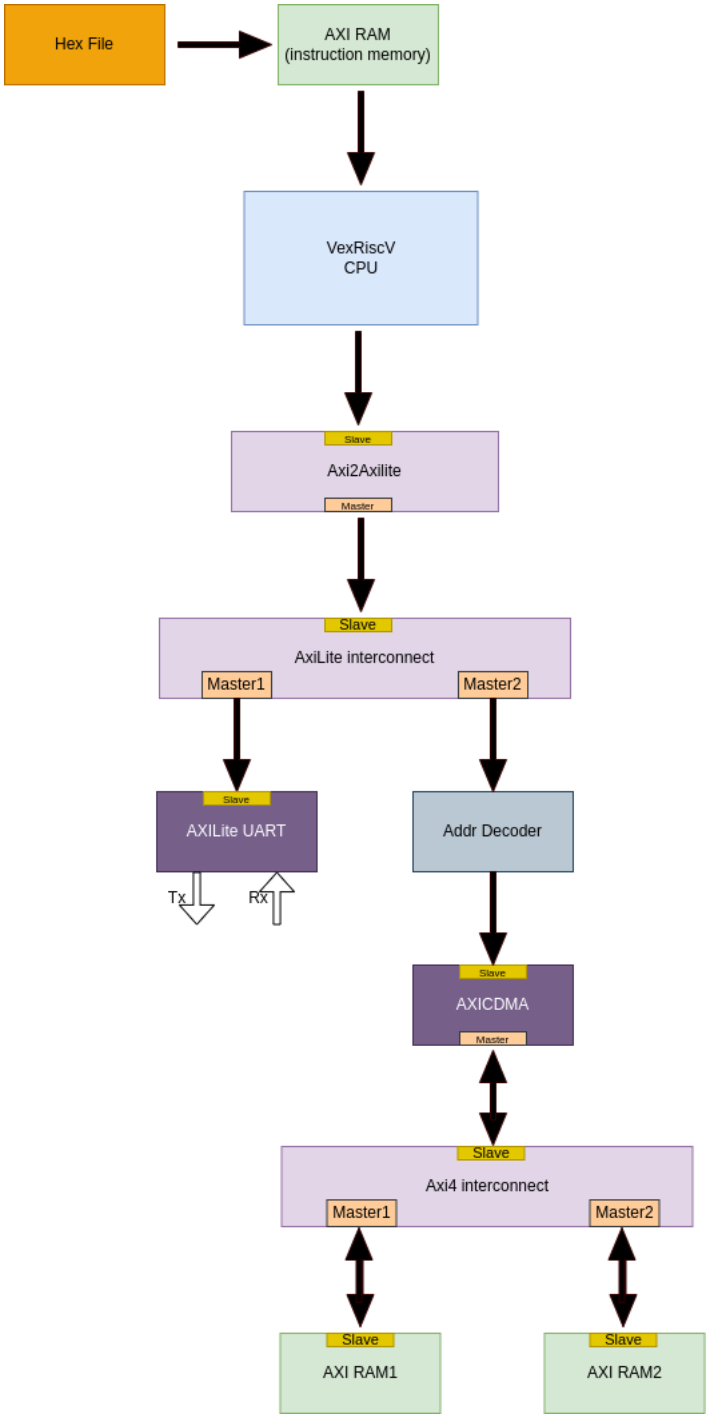

Figure 3. VexRiscv SoC with an AXI CDMA

- **VexRiscv with PLIC and CLINT Interrupts**
  This model consists of a VexRiscv CPU in an AXI4 implementation with PLIC and CLINT modules, connected to an AXI Block RAM and and AXILite UART as peripherals via an AXILite interconnect communicating with the CPU with an AXI2AXILite bridge. The CPU loads instructions from another AXI memory connected directly to the CPU instruction bus, without any interconnect, as an embedded memory serving the purpose of a ROM. The CPU then executes the instructions on the attached IPs handling interrupts according to the written ISR. The VexRiscv used here is a cached variant of the processor with MMU, PLIC and CLINT interrupt routines. The PLIC is responsible for handling external IP interrupts while the CLINT is responsible for

handling of the internal local inter-processor-interrupts. To access AXILite Peripherals that do not have support for the complete AXI4 protocol, an AXI2AXILite bridge is utilized that translates the AXI4 transactions into AXILite transactions with minimal performance overhead. It does so by utilizing a FIFO to store the calculated addresses of the burst transactions of the AXI4 protocol and executing them in an AXILite fashion. This serves as the interconnect between the CPU, via the AXI2AXILite bridge, and peripherals. This design is based on a 1x4 Interconnect configuration. Two instances of AXI4 Block Ram are used in this model. One connected directly to the CPU with the IBus to serve the purpose of ROM. This ROM is loaded with a hex file which contains the instructions for the CPU. The second instance is used as a peripheral for the system, the CPU attempts read and write accesses to this Ram. A UART is connected as the second peripheral to the CPU and this UART generates five types of interrupts based on the state and operation of UART. This UART is connected in a loopback sense so as to check the data validity by receiving the same data as was written. The PLIC and CLINT modules are connected as the fourth and third masters to the interconnect. CLINT handles all the inter-processor timer interrupts based on the number of cores and HARTs in the RISCV CPU. Whereas the PLIC is responsible for handling all the interrupts of the external IPs, such as UART in this example design. UART raises an interrupt and it flows to the CPU via the PLIC module which then handles the interrupt before resuming the normal operation. A pictorial representation of this model can be seen in Figure 4.

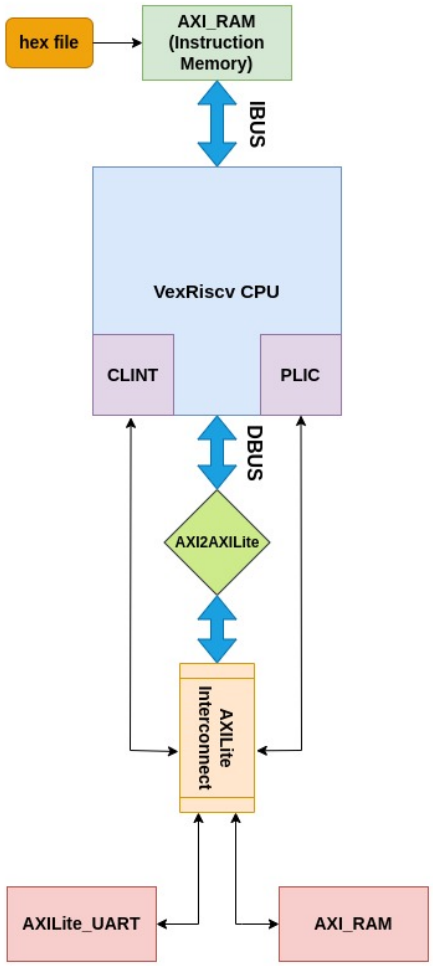

Figure 4. VexRiscv SoC with PLIC and CLINT

IP Support Details

The Table 1 gives the support details for Digital Signal Processor.

Compliance		IP Resources					Tool Flow		
Device	**Interface**	**Source Files**	**Constraint File**	**Testbench**	**Simulation Model**	**Software Driver**	**Analyze and Elaboration**	**Simulation**	**Synthesis**
GEMINI	Standard	Verilog	SDC	C / C++ / Python	Verilog	Icarus	Raptor	Raptor	Raptor

Table 1. IP Details

Resource Utilization

The parameters for computing the maximum and minimum resource utilization are given in Table **??**, remaining parameters have been kept at their default values.

Tool	Raptor Design Suite		
FPGA Device	GEMINI		
Configuration		**Resource Utilization**	
Minumum Resource	CPU: Cacheless VexRiscv Peripherals: AXI RAM	**Resource**	**Utilized**
		LUTs	2589
		DSPs	4
		Registers	2325
		Carry Chain	194
		BRAM	32
Maximum Resource	CPU: Cacheless VexRiscv Peripherals: AXI CDMA AXI RAM AXILite UART	**Resource**	**Utilized**
		LUTs	4828
		DSPs	4
		Registers	2925
		Carry Chain	486
		BRAM	16
Median Resource	CPU: Cached VexRiscv with MMU, PLIC and CLINT Peripherals: AXI RAM AXILite UART	**Resource**	**Utilized**
		LUTs	2777
		DSPs	4
		Registers	2363
		Carry Chain	284
		BRAM	8

Table 2. SoC Resource Utilization

Ports

Table 3 lists the top interface ports of the Digital Signal Processor.

Signal Name	I/O	Description
clock	I	System Clock
reset	I	Active High System Reset

Table 3. VexRiscv SoC Interface

Design Flow

IP Customization and Generation

These reference designs are a part of the Raptor Design Suite Software. A customized SoC design can be generated from the Raptor's IP configurator window utilizing the required IPs as shown in Figure 5.

Figure 5. IP Selection Window

SoC Design: After generating the required IPs with the required parameters, their interfaces can be joined together from within the Raptor Suite to form the required SoC. A snapshot of the window for connecting the IPs together is shown in Figure 6.

vex_soc.v

Search | Save | Undo | Redo | Cut | Copy | Paste | Delete | Select

```
1   /*
2   Top-level module for the Vexriscv SoC with AXI RAM and AXI Interconnect
3   */
4   `timescale 1ns / 1ps
5
6   module vex_soc (
7       input wire      reset,
8       input wire      clk
9   );
10      wire            vexriscv_dBusAxi_ar_ready;
11      wire            vexriscv_dBusAxi_aw_ready;
12      wire    [7:0]   vexriscv_dBusAxi_b_payload_id;
13      wire    [1:0]   vexriscv_dBusAxi_b_payload_resp;
14      wire            vexriscv_dBusAxi_b_valid;
15      reg     [31:0]vexriscv_dBusAxi_rf_payload_data;
16      wire    [7:0]   vexriscv_dBusAxi_r_payload_id;
17      wire            vexriscv_dBusAxi_r_payload_last;
18      wire    [1:0]   vexriscv_dBusAxi_r_payload_resp;
19      wire            vexriscv_dBusAxi_r_valid;
20      wire            vexriscv_dBusAxi_w_ready;
21      wire            vexriscv_debugReset = 1'd0;
22      wire            vexriscv_externalInterrupt  = 1'd0;
23      wire            vexriscv_iBusAxi_ar_ready;
24      wire    [31:0]vexriscv_iBusAxi_r_payload_data;
25      wire    [7:0]   vexriscv_iBusAxi_r_payload_id;
26      wire            vexriscv_iBusAxi_r_payload_last;
27      wire    [1:0]   vexriscv_iBusAxi_r_payload_resp;
```

Figure 6. IP Configuration

Example Design

Overview

Example designs for all three reference models are also available in the Raptor Design Suite the detail for which are given below: -

- **VexRiscv with AXI RAM**
  Read and Write operations on the peripheral RAM via the CPU.
- **VexRiscv with CDMA**
  Read and Write operations on the peripheral RAM via the CDMA while CPU performs trasmit and receive operations on the UART.
- **VexRiscv with PLIC and CLINT**
  Read and Write operations on the peripheral RAM and the peripheral UART while an ISR running on the CPU handles the interrupts generated by the UART.

Simulating the Example Design

All these reference designs can be simulated via any simulator of choice among which Verilator and Icarus are bundled within the Raptor Design Suite. Since the top module port list only consists of a clock and a reset, a simple testbench generating a clock should suffice enough for the simulation since the rest of the testcases are written in RISCV ISA and are directly loaded on the embedded AXI RAM as the instructions for the CPU.

Synthesis and PnR

Raptor Suite is armed with tools for Synthesis along with Post and Route capabilities and the generated post-synthesis and post-route and place netlists can be viewed and analyzed from within the Raptor. The generated bitstream can then be uploaded on an FPGA device to be utilized in hardware applications.

Test Bench

The testbench for all these reference designs are bare-metal firmware written in RISCV ISA loaded directly on the CPU in .hex format. The CPU then performs some specific operations on an attached IO device. This bare-metal firmware can either be written in C or Assembly and compiled via the riscv toolchain that generates an executable in .elf format. This .elf executable in turn is used to generate a 32 bit .hex file which is loaded direclty on to the AXI RAM that is embedded close to the processor. These testcases are already packed with the Raptor Suite and can be simulated at ease woth the bundled simulator. The bitstream for all of these designs, and much more, can be generated via the Raptor Suite and uploaded on an FPGA device to be used in hardware applications.

Revision History

Date	Version	Revisions
April 11, 2023	0.01	Initial version SoC Reference Designs User Guide Document