AXI-Based Dot Product Accelerator

Project Overview

The implementation of a hardware accelerator for computing the dot product of two vectors stored in memory was carried out. The design consists of:

- AXI Master Interface: Used to read the input data from BRAM and to write the computed 32-bit dot product back to memory.
- AXI BRAM Controller and BRAM: The AXI master interacts with the AXI BRAM controller to read and write data into Block RAM (BRAM).
- AXI-Lite Slave Interface: Provides register configuration for vector addresses, vector length, and output address.
- **Dot Product Computation Module:** This module receives the input data, performs element-wise multiplication and accumulation, and generates the final result.

The accelerator computes the dot product by multiplying corresponding elements of the two vectors and summing the results. The final 32-bit result is stored in BRAM, and the done signal is asserted high to indicate computation completion.

Implementation Details

Start Signal Control:

- The **AXI-Lite Slave Interface** provides a control register that contains a **start bit** to initiate the accelerator operation.
- Once the start bit is set via AXI-Lite, the accelerator begins execution and transitions through different states.

AXI-Lite Register Configuration:

- The AXI-Lite interface is responsible for setting up the computation parameters by writing to the following registers:
 - Control Register: Contains a start bit to trigger the operation.
 - Vector A Base Address Register: Specifies the starting memory address for vector A.
 - Vector B Base Address Register: Specifies the starting memory address for vector B.
 - Vector Length Register: Defines the number of elements in each vector.

- Output Address Register: Holds the memory address where the computed dot product will be stored.
- Status Register: Reflects the current state of execution (e.g., busy, done, or error).

AXI Master and BRAM Handling:

- The AXI Master retrieves base addresses for vectors A and B, and vector length from the AXI-Lite configuration registers.
- The AXI Master then interacts with the **AXI BRAM Controller**, which manages read/write operations to Block RAM (**BRAM**).
- Using the configured base addresses, the AXI Master loads values from BRAM into the dot product computation module.

Dot Product Computation:

- The computation module receives vector elements from BRAM, performs **element-wise multiplication**, and accumulates the results.
- Intermediate results are stored in a higher precision format to prevent overflow.
- The final **32-bit dot product result** is stored in BRAM at the location specified by the **Output Address Register**.

Control Flow:

- 1. The **host processor writes to the AXI-Lite registers**, configuring vector addresses, length, and setting the start bit.
- 2. The AXI Master reads vector data from BRAM using the provided base addresses.
- 3. The **dot product computation module** processes the elements and computes the final result.
- 4. The computed **dot product is stored back in BRAM** at the output address.
- 5. The status register is updated to indicate completion by setting the done flag.

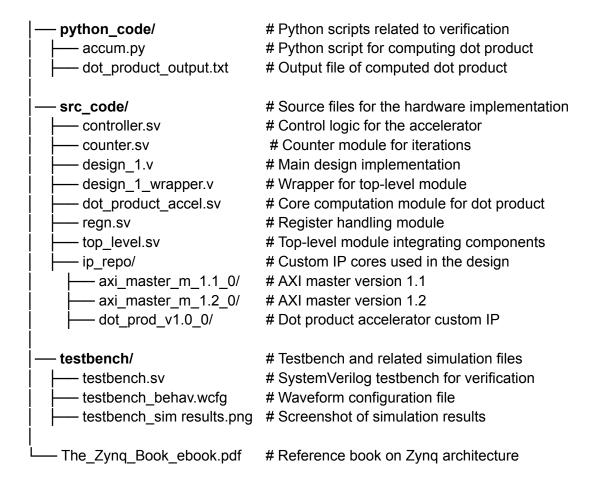
0

Project Directory Structure

Below is the structured hierarchy of files and directories in the project:

AXI-Dot-Product-Accelerator - No AXI Lite/

	— diagrams/	# Block diagrams and testbench simulation results
ĺ	Block Diagram.pdf	# Architectural overview of the design
	testbench_sim results.png	# Screenshot of simulation results
	 documentation/ 	# Contains design documentation



Verification and Testing

A **testbench** was developed to validate the **AXI-based dot product accelerator** by performing the following steps:

- 1. Configuration via AXI-Lite Interface:
 - The testbench writes the base addresses of vectors A and B, vector length, and output address into the AXI-Lite configuration registers.
 - The **start bit** is set in the **control register** to initiate the computation.
- 2. Memory Modeling and Data Initialization:
 - Two BRAM instances (BRAM A and BRAM B) were modeled to store vector A and vector B values.
 - The testbench assumes that vector A and B values are preloaded into these BRAMs.
 - Known 8-bit values were used for verification:
 - First 256 values of vector A and B were assigned incrementally from 0 to 255.
- 3. Execution and Computation Verification:

- The AXI Master fetches the input values from BRAM using the configured base addresses.
- The accelerator performs the element-wise multiplication and accumulation.
- The computed dot product is stored in BRAM at the output address.

4. Result Readback and Verification via AXI-Lite:

- The testbench reads back the computed result using the AXI-Lite interface from the output address.
- The read value is compared against the expected dot product result to verify correctness.

5. Control Flow Validation:

- The testbench ensures proper state transitions:
 - Idle → Busy → Done upon completion of computation.
- The status register is monitored to confirm that the done flag is asserted upon successful execution.

6. Simulation Environment:

 The implementation was simulated using Vivado Simulator to confirm correct behavior and performance.

•

Conclusion

The hardware accelerator was successfully implemented with an **AXI Master interface** handling BRAM access and a **dot product computation module** performing the required operations. The inclusion of an **AXI-Lite interface** enhanced the design by providing a structured method for **configuring control parameters**, including the **start signal**, **vector base addresses**, and **vector length**.

Using **AXI-Lite**, the accelerator's output result was read back and verified against the expected dot product, ensuring correctness. The status register provided clear feedback on computation progress by indicating idle, busy, and done states.

This implementation effectively computed the dot product, stored the result in BRAM, and asserted the done signal upon completion, while the AXI-Lite interface facilitated seamless interaction with the system processor for both configuration and result retrieval.