AXI-Based Dot Product Accelerator

GitHub

Naga Vardhan Maddala

maddalanagavardhan@gmail.com

Design and implemented a hardware accelerator that computes the dot product of two vectors stored in memory, Using an AXI LITE Interface and AXI Interface for efficient data Transfer.

Accelerator
In this Accelerator, if the Start was triggered, I'll Calculate the dot product of my 2 inputs Vector A and Vector B and store the calculated Result in Vector C.

Register Block Configuration (reg_block_fsm)

In this design, a register block contains Six key registers configured using an AXI-Lite slave interface. The AXI-Lite slave interface allows the master (testbench) to initiate transactions and write data to these registers. Each register serves a specific role in controlling and monitoring the functionality of the module.

1. **Control Register (REG0):**
   - This 32-bit register holds control bits to configure the operation. It includes a "start" bit that triggers the accelerator to begin its operation. This Register will be configured using a 32-bit address (32'h0000).
2. **Vector A Base Address (REG1):**
   - A 32-bit register that holds the base memory address of the first vector (Vector A) in memory. Where the data in the memory will be the input for Vector A. This Register will be configured using a 32-bit address (32'h0004).
3. **Vector B Base Address (REG2):**
   - This 32-bit register holds the base memory address of the second vector (Vector B). Where the data in the memory will be the input for Vector B. This Register will be configured using a 32-bit address (32'h0008).
4. **Vector Length (REG3):**
   - A 32-bit register that specifies the number of elements in each vector. This will Specify the total number of times we need to fetch the data from the Memory and pass to Accelerator for Memory. This Register will be configured using a 32-bit address (32'h000C).
5. **Output Address (REG4):**
   - A 32-bit register that holds the destination memory address for the result of the dot product calculation. This register is used to store the result in memory. This Register will be configured using a 32-bit address (32'h0010).

6. **Status Register (REG5):**
    - A 32-bit register that communicates the status of the operation. It includes flags such as "busy," "done," and error indicators, providing feedback on the progress or completion of the operation. This Register will be configured using a 32-bit address (32'h0014).

CORE ACCELERATOR

A core accelerator will fetch the data from the reg_block_fsm to perform the operation. The fetched data will have all the necessary data about the control flow. Based on this fetched data we need to communicate with the test bench using an AXI Master Interface and then we need to pass the collected data into the Accelerator as Inputs (Vector A & Vector B).
Here, I've designed an FSM for the entire operation to be performed. I've Designed an FSM using 9 States.

These are the following 9 States:
State 1: idle_state = 4'b0000
State 2: read_request_state_1 = 4'b0001
State 3: read_data_state_1 = 4'b0010
State 4: read_request_state_2 = 4'b0011
State 5: read_data_state_2 =4'b0100
State 6: accelerator_operation_state = 4'b0101
State 7: write_address_state = 4'b0110
State 8: write_data_state = 4'b0111
State 9: write_response_state = 4'b1000

State 1: idle_state = 4'b0000
In this state, if the data in my REG0 is 1 then I'll start my operation and move to my next state read_request_state_1.

State 2: read_request_state_1 = 4'b0001
In this State, I'll use the data in REG1 as base address of Vector A. Using AXI interface I'll fetch the data from the Memory and stored it in the Temporary Register and move to next state read_data_state_1.

State 3: read_data_state_1 = 4'b0010
In this State, I'll Collect the RDATA using AXI Interface and stored it in the Temporary Register as vector A input and move to next state read_request_state_2.

State 4: read_request_state_2 = 4'b0011

In this State, I'll use the data in REG2 as base address of Vector B. Using AXI interface ARADDR I'll pass my address to testbench which has local memory and move to next state read_data_state_2.

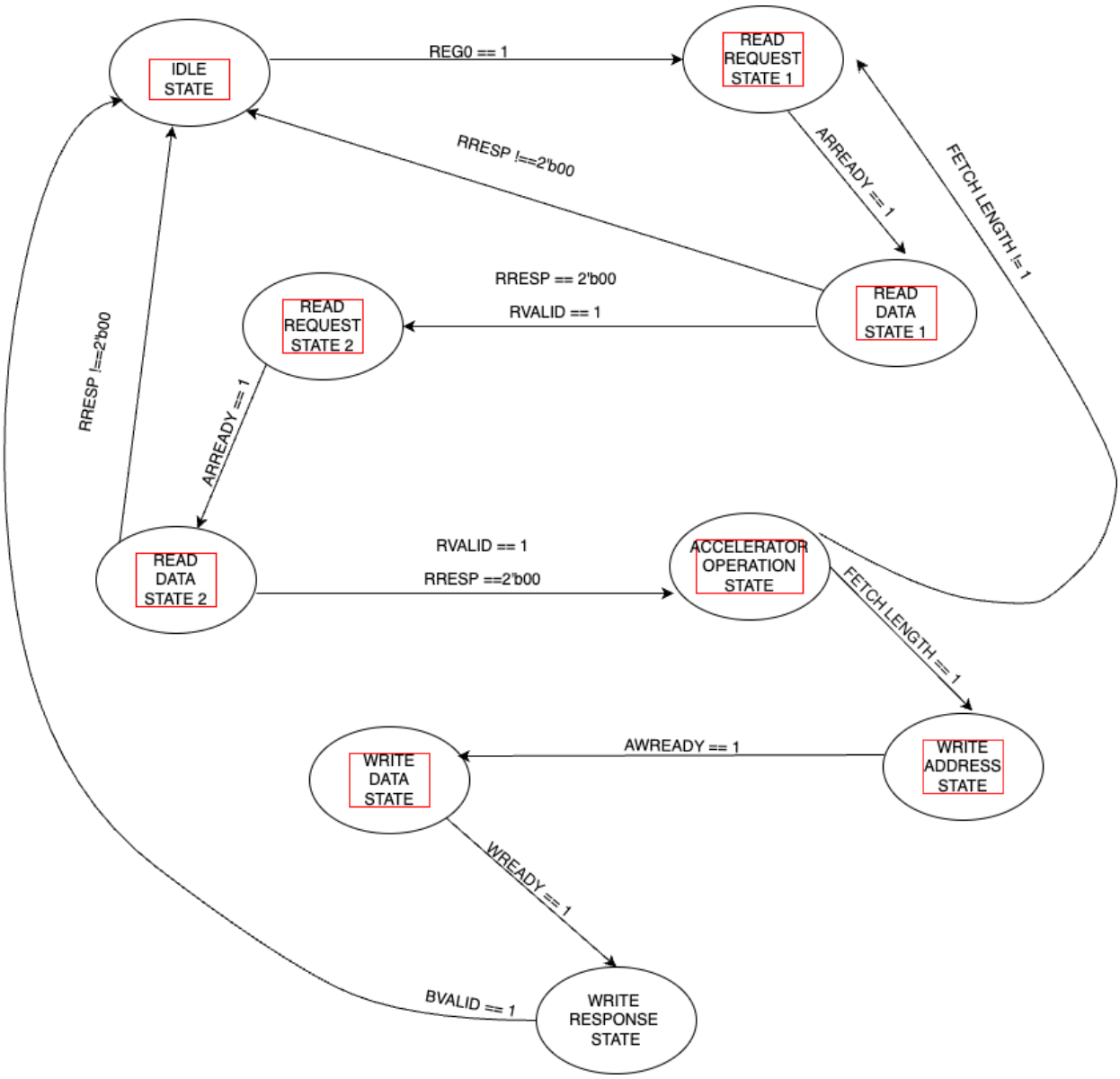
State 5: read_data_state_2 =4'b0100
In this State, I'll Collect the RDATA using AXI Interface and stored it in the Temporary Register as vector B input and move to accelerator_operation_state.

State 6: accelerator_operation_state = 4'b0101
In this State, I'll trigger the start operation in the Accelerator. If provided based on fetch length I'll repeat my operation continuously until it reach my fetch length requirements and move to next state write_address_state.
State 7: write_address_state = 4'b0110
In this State, I'll use the data in REG4 as base address of Vector C (output of Accelerator). Using AXI interface AWADDR I'll pass the address to Testbench and move to next state write_data_state_1.

State 8: write_data_state = 4'b0111
In this state, I'll Send the Base the output data of Accelerator Vector C using a AXI Master Interface WDATA and move to next state write_response_state.

State 9: write_response_state = 4'b1000
In this state, I'll wait for the Response and once the response received I'll update the status in REG5 and clear the Vector C and I'll move to next state idle_state.

Finite State Machine

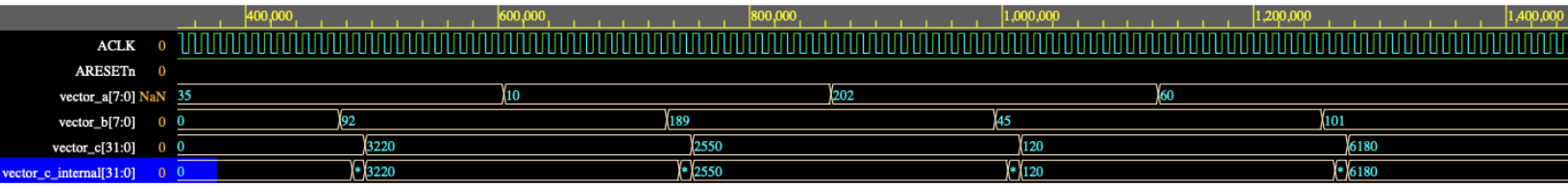


Challenges Faced:

- Comprehending and adhering to the AXI Lite and AXI protocol requirements.
- Effectively managing control signals across different states posed a significant challenge.
- Designing an optimal finite state machine that accounts for all possible scenarios.
- Ensuring proper handshaking management throughout the process.

Result:

Waveform



Here my Vector A and Vector B are Signed Values. So, when there is a value which is above the range (-128 to 127) while performing the Dot Product the value will be considered as 2's Complement and then the Calculation will happen.

Explanation:

- Here in my Waveform, the 1st Set of Vector A & B are in range of (-128 to 127) the Dot Product for two positive numbers are Calculated and Stored in temporary variable for Accumulation.
- The 2nd Set of Vector B is out of range (189), So the 2's Complement of 189 and Vector A Dot Product Was Calculated and Accumulated with previous Result.
- The 3rd Set of Vector A is out of range (202), So the 2's Complement of 202 and Vector B Dot Product Was Calculated and Accumulated with previous Result.
- The 4th Set of Vector A & B are in range of (-128 to 127) the Dot Product for two positive numbers are Calculated and Accumulated with previous Result.

Conclusion:

Successfully Implemented the Dot Product Calculation of 2 input Vectors. Implemented AXI Lite and AXI interface for efficient data transfer.