Report on 4x4 Systolic Array Design using Bluespec System Verilog (BSV)

Deepak S
MM22B011
CS6230: July-Nov '24

November 26, 2024

1 Introduction

This report presents the design and implementation of a 4x4 systolic array using the Multiply-Accumulate (MAC) module developed in Assignment 1. The systolic array is designed using Bluespec System Verilog (BSV), which is a high-level hardware description language. The project involves matrix multiplication, where two 4x4 matrices are multiplied using the systolic array architecture. The design incorporates random data values for both `int8` and `bfloat16` data types for matrix operations.

2 Design Overview

2.1 Systolic Array Architecture

A systolic array is a network of processing elements (PEs) connected in a regular grid-like fashion, where each PE computes partial results that flow through the array. In this design, each PE is a MAC module capable of performing multiplication and accumulation operations on input values `A` and `B` to produce a result `C`.

The `mat_mult_systolic` module, written in Bluespec, defines the systolic array with the following components:

- **PE Array**: A 4x4 grid of MAC modules (PES), which are responsible for computing the partial products and accumulating them.
- **Registers**: `matA`, `matB`, and `result` are registered variables that store matrices and the output result, respectively. The `matA` and `matB` matrices hold the input values, while `result` stores the output of matrix multiplication.

- **Control Logic**: The design includes logic to manage the data flow between the PEs, process data in cycles, and handle reset conditions.

2.2 PE (Processing Element) Design

The PE module, defined in `pe.bsv`, is a fundamental unit of the systolic array. Each PE consists of three registers (`reg_a`, `reg_b`, and `reg_c`) that store the operands and the result of the multiplication operation. The PE implements a Multiply-Accumulate (MAC) operation, where:

- `A` and `B` are input values fed into the PE.
- The output `C` is computed as the product of `A` and `B`, accumulated in `reg_c`.

3 Design Methodology

The systolic array design follows the steps outlined below:

1. **Matrix Initialization**: Two 4x4 matrices `matA` and `matB` are initialized with random values. These values are processed in cycles to compute the resulting matrix `result`.
2. **Input Feeding**: The values of matrix `matA` are fed into the array along the rows, and the values of matrix `matB` are fed along the columns.
3. **Computation**: In each cycle, each PE computes a partial product of the corresponding values from `matA` and `matB`. The accumulated results are stored in the `result` register.
4. **Output**: The final result of matrix multiplication is obtained after `2 * MAT_DIM` cycles, with the result matrix `result` representing the product of `matA` and `matB`.

3.1 Key Features of the Design

- **Control Signals**: The control logic manages the processing of data in each cycle, ensuring the correct feeding of input values into the PEs.
- **Data Flow**: Data flows through the PEs in a synchronized manner, ensuring that partial products are accumulated correctly.
- **Reset Functionality**: The design includes a reset function that initializes all registers in the systolic array to zero, enabling clean restarts for multiple test runs.

4 Verification Methodology

While the verification for this design was not completed due to time constraints, the intended verification methodology is as follows:

1. **Test Setup**: The two matrices `matA` and `matB` are initialized with random values for both `int8` and `bfloat16` data types. These matrices are then fed into the systolic array for matrix multiplication.

2. **Expected Results**: A Python reference model will be used to calculate the expected result of matrix multiplication for the given matrices. The result from the BSV design will be compared against the reference model's output.

3. **Coverage and Randomization**: The test suite will include various random test cases for different matrix values and verify the correctness of the output. Coverage metrics would be defined to ensure all operations within the systolic array are exercised.

4. **Cocotb Framework**: The cocotb framework would be used to drive the verification process, automating the generation of test vectors and comparison of results between the hardware model and Python reference.

5 Challenges and Issues

The primary challenges faced during the design and implementation of this project include:

- **Time Constraints**: Due to limited time, the verification process using the cocotb framework was not completed. This resulted in the inability to fully test and verify the matrix multiplication functionality.

- **Complexity of Systolic Array**: Designing and synchronizing the data flow in the systolic array required careful attention to ensure that data is properly fed into the PEs and that partial products are accumulated correctly.

- **Data Type Handling**: Handling both `int8` and `bfloat16` data types posed some challenges in terms of ensuring correct precision and accumulation in the MAC operations.

6 Conclusion

The design and implementation of the 4x4 systolic array using Bluespec System Verilog successfully followed the outlined specifications. The architecture of the systolic array was realized by interconnecting 16 MAC modules in a grid, and the matrix multiplication operation was implemented using a cycle-based data

flow approach. Although the verification process was not completed due to time constraints, the design can be extended by integrating the cocotb framework for functional and coverage-based verification.