Why I rewrote my capstone project in BSV

Author: Sai Govardhan M C
Email: sai.govardhan@incoresemi.com

Introduction

This document outlines the motivation to using Bluespec System Verilog which provides high levels of abstraction to rapidly design hardware microarchitecture.

As it happens with most Electronics students, I started out designing digital design logic, in Verilog, with reference to coding guidelines from the Sunburst Papers and careful guidance from my UG professor and course textbooks.

Specifying designs with parallelism and concurrency in Verilog was always a challenge due to lower levels of abstraction and regular rework to fix synthesis-simulation mismatches.

At InCore, the use of BSV is one of the superpowers that enabled a small teams like us to specify complex hardware intuitively, correctly and efficiently.

During my internship at InCore in 2024, as a novice BSV designer, I decided it would be meaningful to contrast the efforts that my team and I spent at college implementing the Multi Dimensional Sorting Algorithm(MDSA) in Verilog - by rewriting the MDSA Bitonic variant in BSV. This blog collates these insights, to establish a strong use-case of BSV at universities.

I shall be diving into the BSV implementation by explaining essential parts of the micro-architecture, and corresponding snippets from the code base.

More of our work on the taxonomy of sorters, low power methodologies, other variants (Hybrid and Odd-Even sorters) and our ASIC implementation results can be referred to in our published paper Low Power Multidimensional Sorters using Clock Gating and Index Sorting.

My complete MDSA Bitonic Implementation in BSV, along with our legacy Verilog implementation can be found in my GitHub repository.

The Compare And Exchange Block

The Compare And Exchange (CAE) block is a fundamental building block of systolic array based hardware sorters. It simply compares two inputs and presents an ascending order output.

In the following code snippet, the CAE block checks if cae_in[0] is greated that cae_in[1] and uses the Vector to Vector `reverse` function to swap the values.

- Specify the CAE typedef:

typedef Vector#(2, Bit#(WordLength)) CAE;

`bsv/MDSA_bitonic/mdsa_types.bsv[lines:9]`

- Declare the ActionValue mav_get_sort:

```
method ActionValue#(CAE) mav_get_sort (CAE cae_in);
   if(cae_in[0] > cae_in[1]) begin
      cae_in = reverse(cae_in);
   end
   return(cae_in);
endmethod
```

bsv/MDSA_bitonic/cae.bsv[lines:38..43]

The Bitonic Sorting Unit

The MDSA Algorithm

The MDSA Architecture Implementation

References

1 2 3 4 5 6 7 8

Acknowledgements

The block diagrams and drawings to aid the explaination of the CAE, Bitonic and MDSA are from the paper[1], and the legacy Verilog codebase from the team (names) guided by my UG Professor -(name) at PES Universtiy, Bangalore.

The BSV language [] [] compiler and training material

Initial Work on MDSA and its predecessors[][].

Much thanks to my manager (name) and peers (names) for reviewing the blogpost and InCore for permitting to write on this topic.