Imperial College London

Verilog MIPS CPU Datasheet

Nock, James Shah, Dharmil Sand, Joachim
Davis, Louise Rzepala, Bartek
Lertsakulcharoen, Sarun

Department of Electrical and Electronic Engineering

Team 13

November 2021

1 Overview

Mavalon, is a **reliable** CPU IP Core that executes a subset of the MIPS I ISA [1] as outlined in Revision 3.2 of the ISA specification [2]. It uses an Intel Avalon Memory-Mapped Interface [3] to communicate with memory-mapped devices, such as RAM.

Property	Value
Number of GPRs	32 (31 excluding `$zero`)
Clocks per Instruction (CPI)	3
Pipeline Stages	No pipeline
Maximum Clock Speed	7.68 MHz
Resource Utilization	9,391 / 15,408 (61% of logic elements)

Table 1: Key *Mavalon* Statistics

Statistics in Table 1 are derived from simulating *Mavalon* on a Cyclone IV target, using the Slow 1200mV 85C Model.

2 Limitations

In favour of reliability and stability - over speed and efficiency - *Mavalon* does not implement the following features:

- Floating Point operations
- Coprocessor instructions (SWCz, LWCz, COPz opcodes)
- Exceptions
- SYSCALL, BREAK, SWL and SWR opcodes

3 Architecture

3.1 Block Diagram

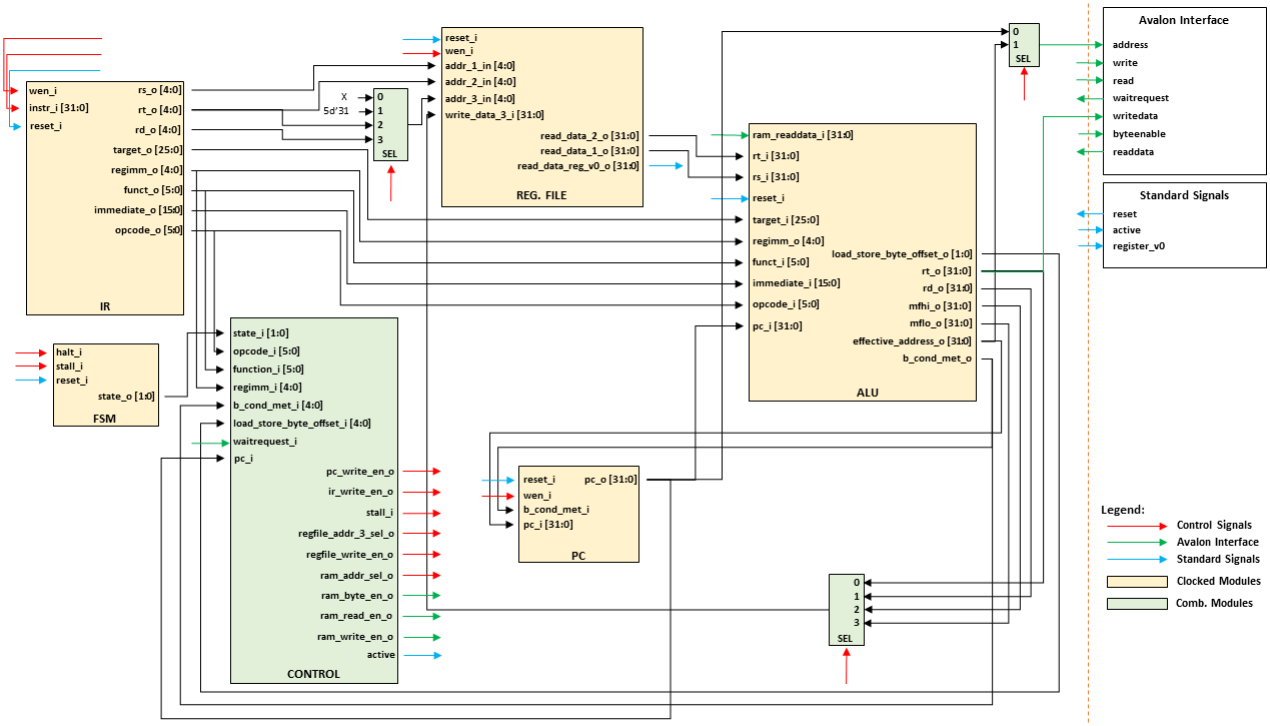

Figure 1: *Mavalon*'s Micro Architecture.

3.2 Components

Mavalon's key components are outlined below.

1. **FSM**: Outputs the state of the CPU. *Mavalon* has four main states: HALT, FETCH, EXEC1 and EXEC2 that correspond to distinct stages in instruction execution.

2. **IR**: Holds and decodes the instruction currently executed until the subsequent instruction is read.

3. **ALU**: Performs most arithmetic operations required by instructions. The ALU houses the HI/LO registers, making it clocked.

4. **REG. FILE**: Component has 1-Write and 2-Read ports and 32 general purpose registers. Reads are combinatorial; writes are clocked.

5. **PC**: Holds the address of the current instruction being executed.

6. **CONTROL**: A combinatorial component that houses the logic for all clocked-component write-enables and multiplexer selects.

3.3 Design Choices

Design decisions were made to preserve *Mavalon*'s **simplicity**, **reliability** and **functionality**.

Mavalon exhibits a CPI of 3. All instructions are executed in three distinct phases: FETCH, EXEC1 and EXEC2. During FETCH, the next instruction to be executed is read from memory. This is written to the IR in EXEC1; REGFILE write-backs occur in EXEC2. Breaking instruction execution into distinct states made the control logic for preserving the state registers during `waitrequest` stalls simpler.

Unlike many MIPS architectures outlined in literature [4], *Mavalon*'s PC contains its own 32-bit adder to compute the next FETCH address in EXEC2. This frees up the ALU in EXEC2 to perform arithmetic operations, reducing the need for additional control logic that would prevent multiple operations in a state requiring the same hardware.

Mavalon's ALU implements a single-cycle multiplier and divider. This avoids the need for additional CPU stalls and control logic required to integrate a multi-cycle algorithm at the expense of performance. *Mavalon*'s critical path is attributed to a single-cycle divider in the ALU, which also consumes the most hardware resources.

Smart design decisions were made, such as using `Typedefs` and `Enums` to improve HDL readability and to leverage type safety. Additionally, default control signals were used in order to decrease the complexity of the control logic, keeping it under ~ 150 lines of code. By only stating what is necessary and falling back to defaults, it is much easier to manually verify the correctness of the logic.

4 Testing

Mavalon was tested with Verilog module testbenches and instruction test-cases, with a RAM that emulates random Avalon `waitrequests`. Apart from ensuring functionality, tests were targeted to check potential edge-cases defined in the ISA and bugs caught during *Mavalon*'s development process. For example, one test [5] accurately assesses if PC-region jumps are performed from the branch delay slot, rather than the `J` or `JAL` instructions themselves.

Additionally, C programs were written and compiled to MIPS. In particular, recursive Factorial and Binary Search algorithms [6, 7] are tested as they make heavy use of the stack - thus testing testing load, store and link instructions.

Continuous integration practices were also followed during development and testing: every code submission required a set of automated tests to pass, as well as code review. The flowchart in Figure 2 summarises how testing was performed.

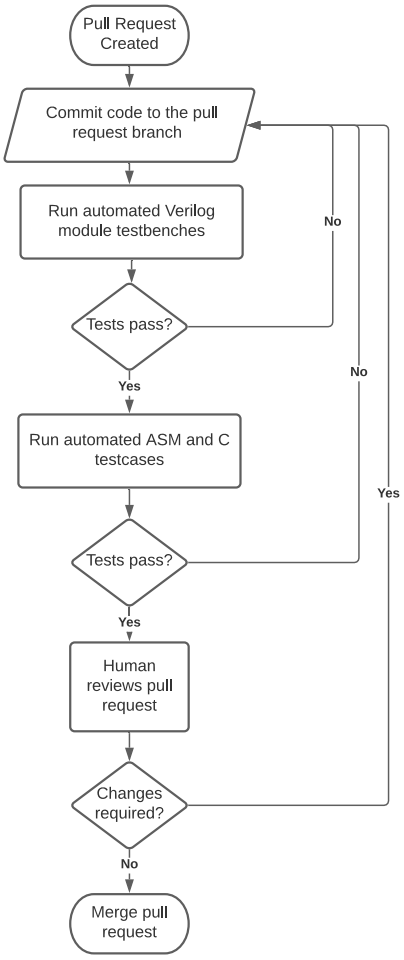

Figure 2: Continuous-integration automated testing flow

5 Future Work

Fuzz Testing the CPU could be explored: one could generate a random series of instructions and run them against a reference simulator and the CPU. *Fuzz Testing* can be restricted to simply generating known instructions with random values, as the aim is not to uncover hidden instructions. Additionally, one could increase the CPI of *Mavalon* by conditionally skipping the EXEC1 state for instructions that do not perform loads or stores.

References

[1] T. David. Elec50010 instr. arch + comp. : Cpu coursework. Imperial College London. [Online]. Available: https://github.com/m8pple/elec50010-2021-cpu-cw#instruction-set

[2] P. Charles. Mips iv instruction set. MIPS Technologies, Inc. [Online]. Avail-

able: https://www.cs.cmu.edu/afs/cs/academic/class/15740-f97/public/doc/mips-isa.pdf

[3] Intel. Avalon interface specifications. Intel. [Online]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/manual/mnl_avalon_spec.pdf

[4] C. Kruskal and S. Scolnik. Multicycle datapath. University of Maryland. [Online]. Available: https://www.cs.umd.edu/~meesh/cmsc311/clin-cmsc311/Lectures/lecture33/multi_cycle.pdf

[5] T. 13. Verilog cpu. Imperial College London. [Online]. Available: https://github.com/Jpnock/verilog-cpu/blob/master/test/mips/jal/03_jal_correct_pc_plus_4_msb.asm

[6] ——. Verilog cpu - binary search test. Imperial College London. [Online]. Available: https://github.com/Jpnock/verilog-cpu/blob/master/test/mips/all/22_binary_search_1.c

[7] ——. Verilog cpu - factorial test. Imperial College London. [Online]. Available: https://github.com/Jpnock/verilog-cpu/blob/master/test/mips/all/03_factorial.c