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
Min. Clocks per Instruction (CPI)	3
Pipeline Stages	No pipeline
Maximum Clock Speed	7.68 MHz
Resource Utilization	9,391 / 15,408 LEs (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

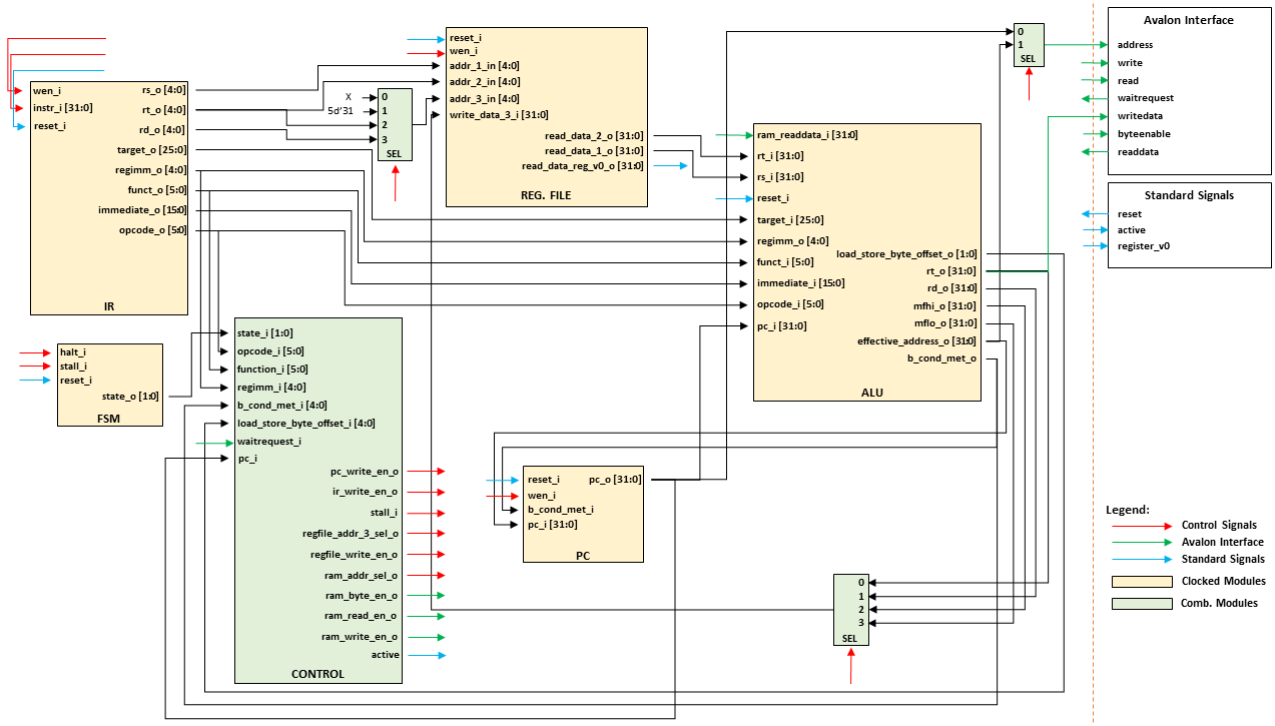

Figure 1: *Mavalon*'s Micro Architecture.

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. The CPU halts, thus entering the HALT state, when the instruction at address `0x00000000` is executed.

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 contains 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**.

- Disregarding `waitrequest`, *Mavalon* exhibits a CPI of 3. Its state machine has four states: FETCH, EXEC1, EXEC2 and HALT. During FETCH, the instruction to be executed is read from memory. This is written to the IR in EXEC1; REGFILE write-backs occur in EXEC2. Stalls are implemented by replaying the current state and forcing register-enables LOW.
- Unlike MIPS architectures outlined in literature [4], *Mavalon*'s PC contains its own 32-bit adder to compute the address of the next instruction. This simplifies the control logic as there is no resource-sharing between components.
- *Mavalon*'s ALU implements a single-cycle multiplier and divider. This avoids the need for additional CPU stalls and control logic required to implement a multi-cycle algorithm. The cost of preserving simplicity was performance - *Mavalon*'s critical path is attributed to a single-cycle divider in the ALU, which also consumes the most hardware resources.
- `Typedefs` and `Enums` are used to improve readability and leverage type safety.
- Default values for control signals with special-case overrides are used to keep the complexity of the control logic minimal, making it under 150 lines of HDL.
- Since *Mavalon* is not pipelined, all ISA instruction scheduling restrictions - such as the load delay slot - can be disregarded.

4 Testing

Mavalon is tested with Verilog module testbenches and instruction test-cases, with a RAM that emulates random Avalon-compliant `waitrequests`.

1. Automated test-cases are written in both Assembly and C. Generally C test-cases are only written to implement recursive algorithms such as Factorial and Binary Search [5, 6], as these make heavy use of load and store instructions - via stack accesses and writes - as well as jump and link instructions.
2. Cases are compiled using a MIPS specific `gcc` cross-compiler or assembled using `as`; they are relocated to the reset vector with `ld` and the `.text` and `.data` sections are extracted with `objcopy`.
3. Cases are sequentially loaded into our test-bench RAM and run for a maximum of 65535 cycles before timing out. Additionally, each case is guarded with a 15 second timeout, in case the simulation hangs due to a combinational loop.
4. Special test-cases are built with different `iverilog` flags to change the simulation behaviour. This allows us to insert complex logic which only applies to these specific test-cases.
5. Each test-case defines the expected output at the top of the source-file and the testbench asserts that this value is present in the `$v0` register when the CPU halts.

Mavalon's testbench contains 297 automated tests. Instruction tests were targeted to check for potential edge-cases defined in the ISA and bugs caught during *Mavalon*'s development process. Some edge cases that are tested for include,

- Assessing whether PC-region jumps are performed from the branch delay slot, rather than the `J` or `JAL` instructions themselves [7].
- Assessing whether the CPU issues correct `byteenables` during loads. To do this, tests [8] were built with a special flag that corrupts data in RAM if the byte enable was set, after reading from it. By detecting this corruption, we can infer whether the correct `byteenables` were issued.
- Assessing whether branch and link instructions write unconditionally to the `$ra` register [9].

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.

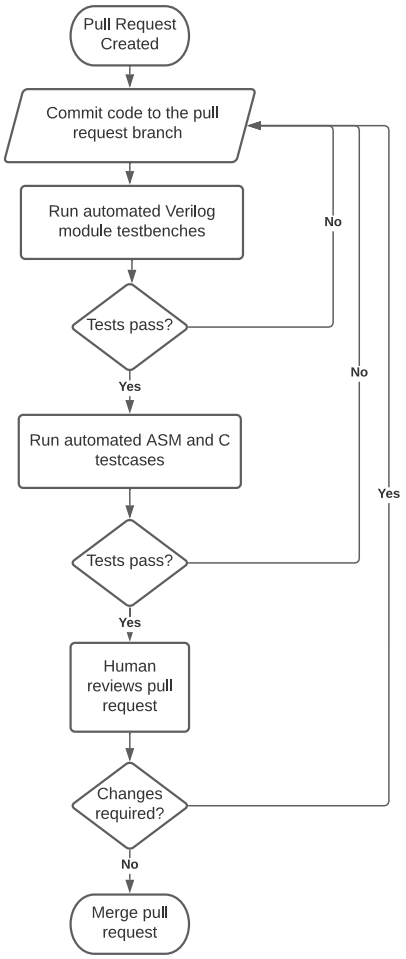

Figure 2: Continuous-integration automated testing flow

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]. Available: 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] IACTeam13. 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

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

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

[8] ——. Verilog cpu - byte enable test. Imperial College London. [Online]. Available: https://github.com/Jpnock/verilog-cpu/blob/master/test/mips/lw/05_check_read_byte_en_lw.asm

[9] ——. Verilog cpu - always link test. Imperial College London. [Online]. Available: https://github.com/Jpnock/verilog-cpu/blob/master/test/mips/bltzal/06_always_link.asm