Multicore RISC-V cache controller

Master thesis specification

Menthors :

Vuk Vranjković

Tivadar Mako

Students :

Ivan Milin E1-79/2023

Petar Stamenković E1-11/2023

Contents

1. System Overview

This document is a specification of a Master thesis project that includes 2 faculty subjects:

1. Advanced microprocessor systems
2. Formal methods of verification and design

It is a cooperation between faculty of technical sciences in Novi Sad and VeriestS.

With first subject we will cover the design of individual RISCV core with its own cache controller and L1 cache alongside with a global cache controller that connects L2 caches and main memory with already mentioned cores. We will use Vivado Design Suite and code our design using Verilog HDL.

With second subject we will cover the verification using Jasper Gold formal tool and SystemVerilog language. Our approach will be *bottom-up* as we will try to verify each developed module during design phase. Our idea is to verify each module with 2 different points of view, firstly with a simple testbench during design phase and secondly using formal tool to verify complex scenarios and increase the coverage.

Figure 1 represents the top diagram of our system.

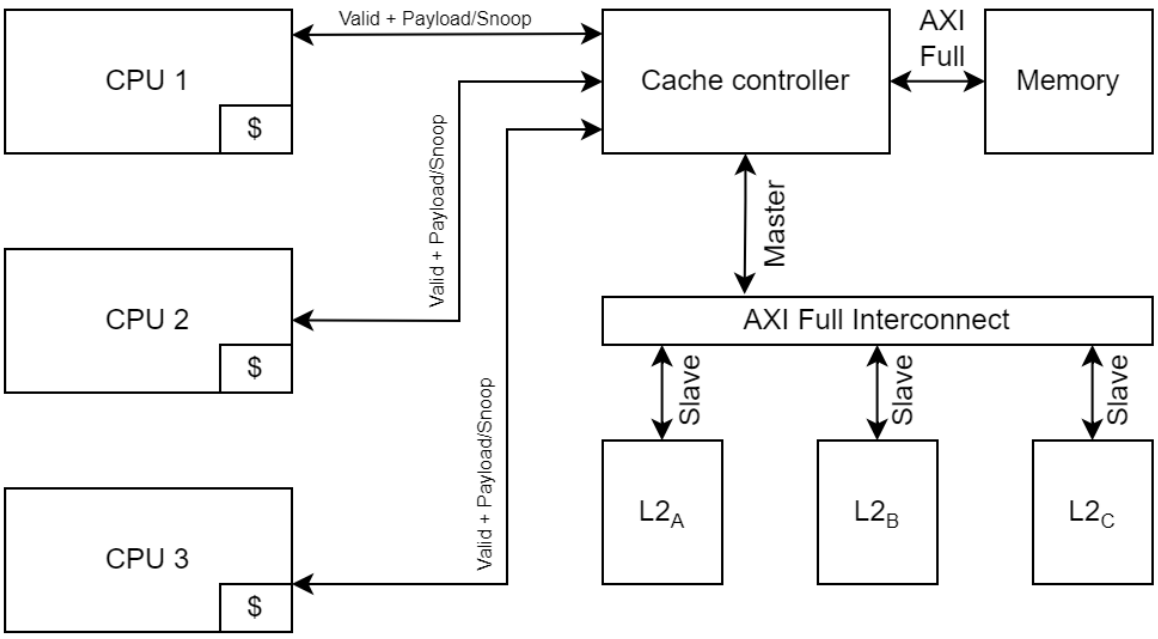

Figure 1 : System Overview

2. Components

As you can see on the figure 1, our system is composed of following components.

1. CPU cores (*CPU1*, *CPU2* and *CPU3*) with their local cache memories (*$*).
2. Global cache controller (*Cache controller*)
3. Global memory (*Memory*)
4. AXI Interconnect
5. Shared L2 cache memories ($L2_A$, $L2_B$, $L2_C$)

This paragraph will give you a more information about each component.

CPU Cores – Each core has RISCV ISA, local cache memory (L1) and cache controller. Our idea is to implement an instruction set (Atomic set – **A** and Integer multiplication and division - **M**). Then we will generate a binary text file from an assembly code using RIPES simulator [1]. We will test our system by loading contents of the binary file into design and viewing the waveforms from Vivado simulator.

Also, each CPU has a feature for accessing data from its local cache memory. If requested data does not exist in local cache memory we flag a *MISS* and send a *SNOOP Request*. When global cache controller receives this request, it has to access other L1 caches and look for a requested data there. It this also fails, global cache controller has to look for request data in L2 or main memory.

Global cache controller – This is a key component of our system and it is responsible for:

- Accepting request from CPU Core. Only one core can be served at once, others have to wait.
- In case of local cache *SNOOP MISS*, scan through L2 caches looking for a requested data. Routing logic will be implemented.
- Sends a *SNOOP Request* if requested data does not exist in local cache.
- If neither L1 nor L2 have the requested data, look for it in global memory.
- Maintain the coherence of caches.

Global memory – Lowest level of memory hierarchy. Biggest capacity but also biggest latency.

AXI Interconnect – Component that connects global cache controller and shared L2 caches. Also does the required routing.

Shared L2 cache memories – Composed of smaller L2 memories in order to reduce latency of data access.

3. Interface

3.1. Interface between CPU Core and Global cache controller

This interface has 5 separate segments:

- Valid + Payload interface to global cache controller, requires address, opcode and data (*in case of non-write command)*
- Valid + Payload for *SNOOP Response*. If it was a *HIT* forward the request data in *payload* and flag a success (*1*), otherwise flag a fail (*0*).
- *SNOOP Request* interface goes from global cache controller to all cores (*number of cores = number of instances of this interface*)
- Data transfer interface from global cache controller to all cores. Fetched data from either L2 or global memory.
- Valid + Payload completion interface. If transaction is completed flag success, otherwise flag an error.

3.2. Interface between global cache controller and L2

For this interface, we decided to use AXI Full, because of its burst mode. We want to fetch a 64 byte cache line from one of slaves. We have one master, global cache controller.

3.3. Interface between global cache controller and main memory

For this interface, we decided to use AXI Full, because of its burst mode. We want to fetch a 64 byte data from the main memory.

4. Commands

4.1. Commands for local cache memories (L1)

- **READ** – This command reads requested data from L1. If it's a *HIT* proceed with an instruction, otherwise it's a *MISS* and CPU Core generates a command for a lower level. Global cache controller takes over the control.
- **READ + UNIQUE –** After global cache controller assures that only CPU that generated this command has a unique data, read it. Idea for this is to be an exclusive state, following the concept of a MOESI protocol for cache coherency. If this instruction is sent, check if cache line is unique and then read it.
- **WRITE –** If there is an empty location in L1, write it. If cache is full, apply LRU, write back that data to L2, and replace it with the new one from the write command.
- **CMO (Cache maintenance operation)**
    - **MI (Make invalid) –** Invalidate data without propagating it to lower level.
    - **CI (Cache invalidate) –** If data was modified (*Dirty*) before invalidating, it must be forwarded to lower level. Otherwise, just invalidate it.

4.2. Commands for L2 caches

- **WRITE –** In this scenario, global cache controller should invalidate this data in all other cores without propagation to lower levels (*SNOOP MI*). For example, if CPU1 generated this command, global cache controller should invalidate this cache line in CPU2 and CPU3 (*if they have this data*). In this situation we assume that this cache line is the newest.
- **READ –** Do a *SNOOP Read.* In this scenario, request data does not exist in neither one L1 cache memory. Read the cache line from L2 and forward it to global cache controller.
- **READ + UNIQUE –** Do a *SNOOP CI.* For example if CPU1 sent this instruction, it fetched cache line X from L2 and modified it resulting in it to become *Dirty*. If then CPU2 requests the same instruction for the same cache line, global cache controllers needs to *SNOOP* on other L1 caches to check if they have that cache line, and invalidate them (*SNOOP CI*), resulting in CPU2 having that cache line clean (*updated*).
- **CMO (Cache maintenance operation)**
    - **MI (Make invalid) –** Do *SNOOP MI.* Just invalidate cache line in L2, without propagation to the global memory.
    - **CI (Cache invalid) –** Do *SNOOP CI.* If the cache line is modified (*Dirty*), invalidate it in L2, and forward it to the global memory.

4.3. Commands for CPU Snooping (from CPU Core to global cache controller)

- **SNOOP READ –** Try to find requested cache line in other L1 memories. If cache line is found, forward it to a global cache controller.
- **SNOOP CI –** Try to find requested cache line in other L1 memories. If the data is *dirty,* invalidate it in L1 you found it in, forward it to the CPU Core that requested it and this CPU Core is obligated to propagate it to lower level.
- **SNOOP MI –** Try to find requested cache line in other L1 memories. If cache line is found, forward it to CPU Core that request it (*via global cache controller*) and invalidate this cache line in all other CPU Cores (*if they have that cache line*).

5. Unit level overview

5.1. RISCV CPU

Figure 2 represents contents of the RISCV CPU.

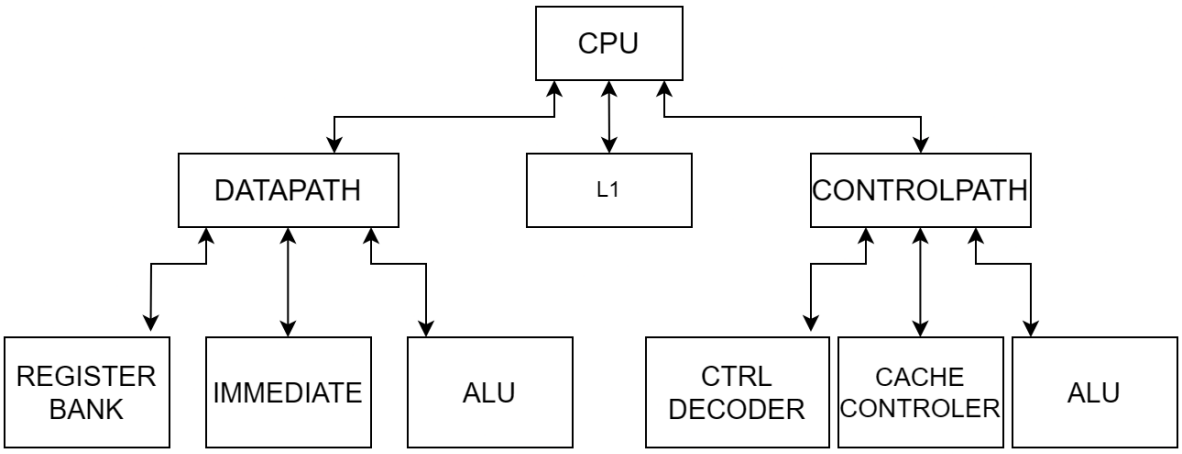

Figure 2 : RISCV CPU representation

References

[1] https://github.com/mortbopet/Ripes/releases