Multicore DLX with coherence protocol – Project Presentation

Project Name: Multicore DLX processor with MESI coherence protocol

Project Number: 3024

Project carried out at: University

Submitted by:

Yarin Koren

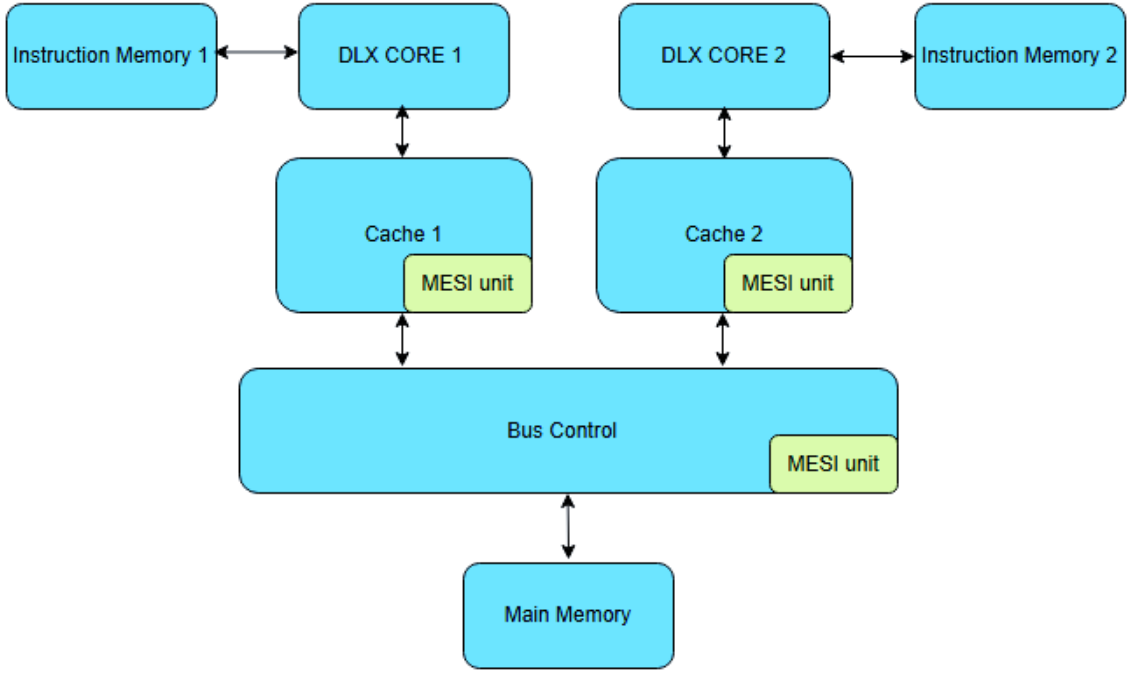
Yohai Shiloh

Project instructor approval signature:

Name: _Oren Ganon_

Introduction

Project general structure:



Motivation

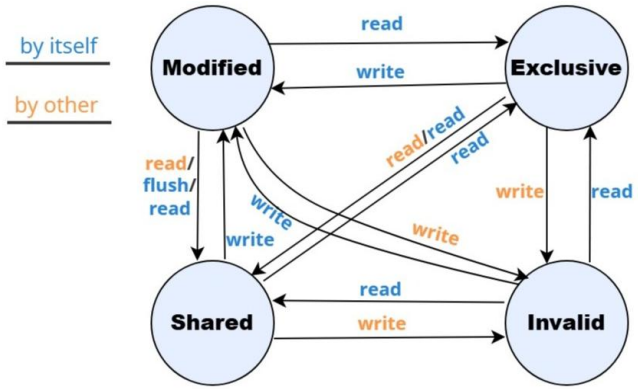
Multicore:

- ***Better performance** – Achieving speedup and high performance by dividing programs across several cores. For example, splitting the input to perform computations on each part at the same time*
- ***Multitasking** – Using parallelism to perform multiple tasks at the same time*
- ***Better usage of resources** – Higher utilization of resources like the main memory, bus, I/O devices etc.*

speed

MESI coherence protocol:

- *Ensuring data coherence and synchronization in all caches in parallel by assigning validation state for each data block*
- *The validation state saved inside the cache – always available immediately*
- *Saves time by reducing data transmission with the main memory by performing "flush" between cores*



The MESI state machine

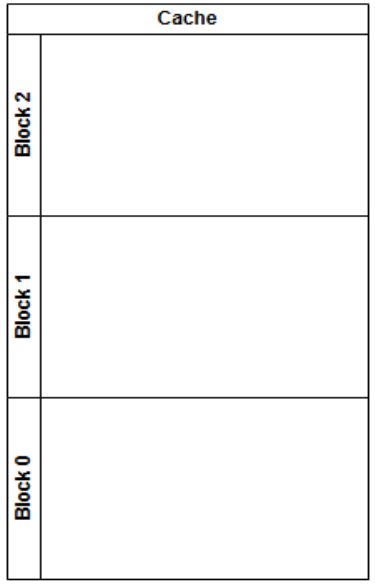
The MESI Protocol

While performing operations in parallel in multicore processor, data coherence must be kept to ensure the data used by each core is up-to-date, otherwise incorrect results might be received due to invalid inputs

The solution – using coherence protocol.

The MESI protocol work mechanism:

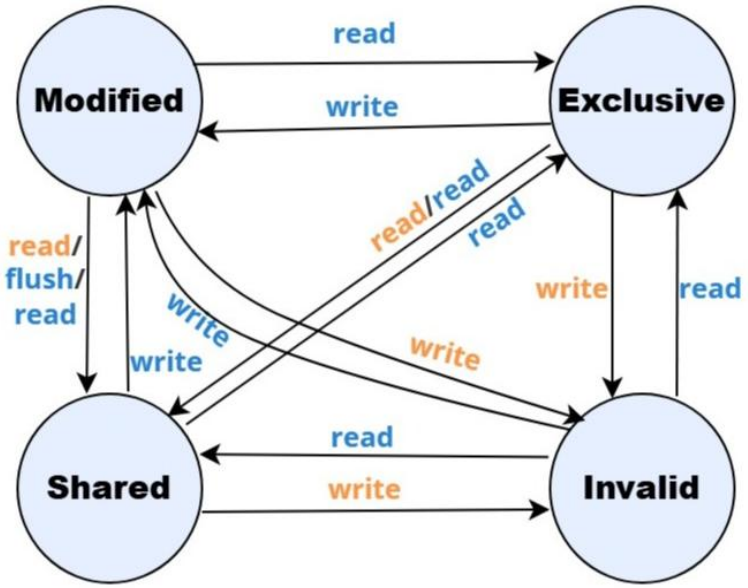
- ***Each block assigned with certain state – Modified, Exclusive, Shared or Invalid***
- ***State transition will occur as result of read/write instruction by the cores***
- ***The cores snoop the bus and communicating each other to update the MESI states if needed***
- ***When modified block is being requested by another core, the core will "flush" the modified block directly, avoiding the latency of updating through the main memory***



Cache structure illustration

by itself

by other

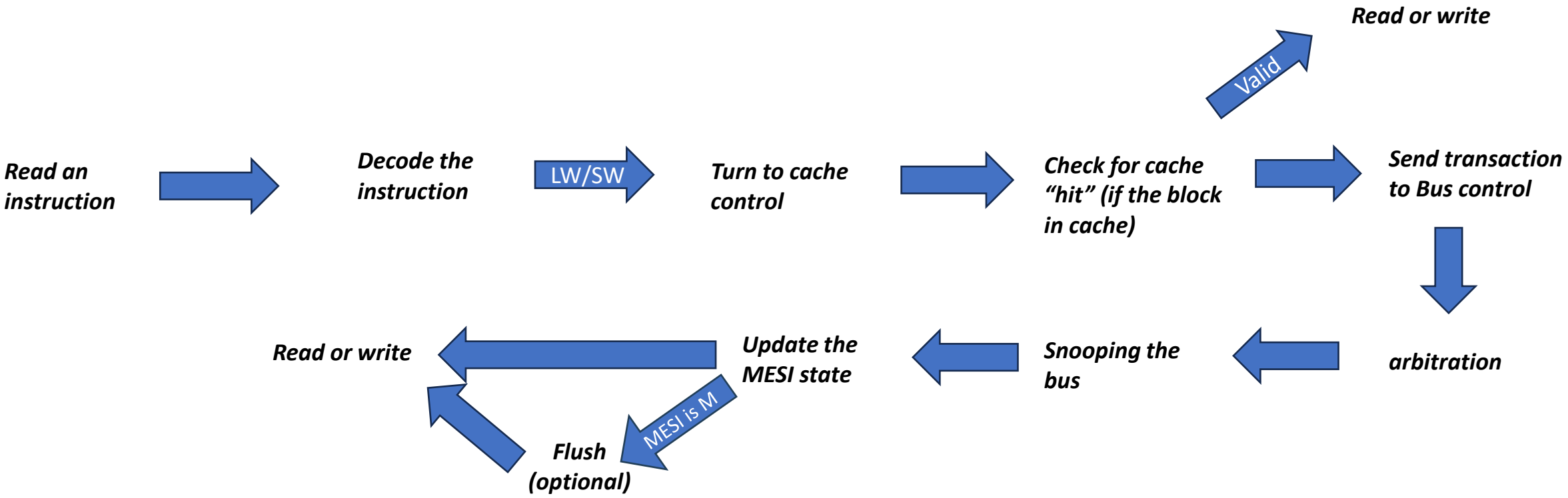


The MESI states transitions diagram

Implementation

Workflow

Below is a general workflow of the cores, both work simultaneously while using MESI protocol.
The bus control preserved correct access to the joint memory by the access queue, making sure each core receive the requested updated data (data routing) and handle signal transmitting.

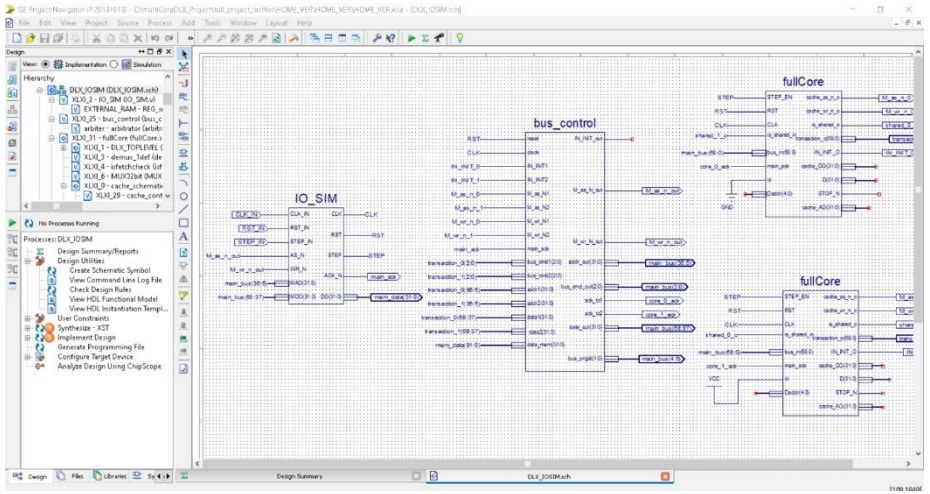


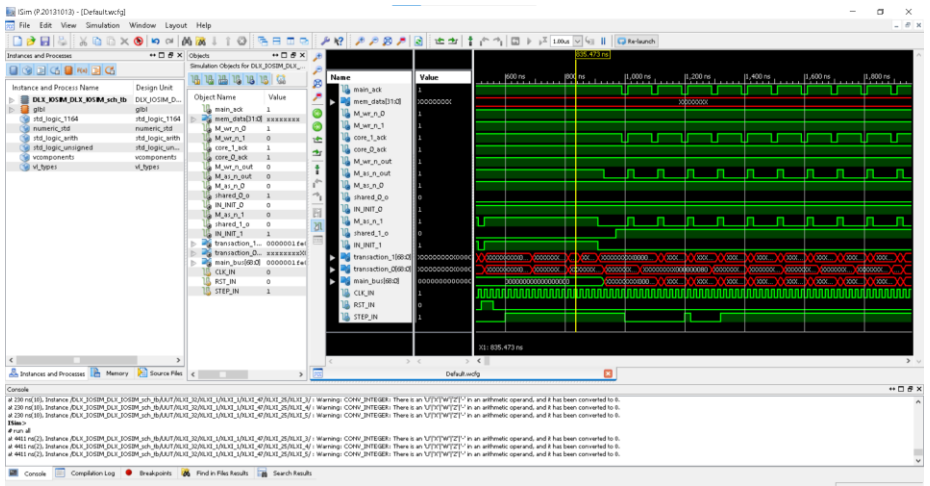
Multicore DLX with coherence protocol – Project Presentation

Implementation

Implementation guidelines and methods

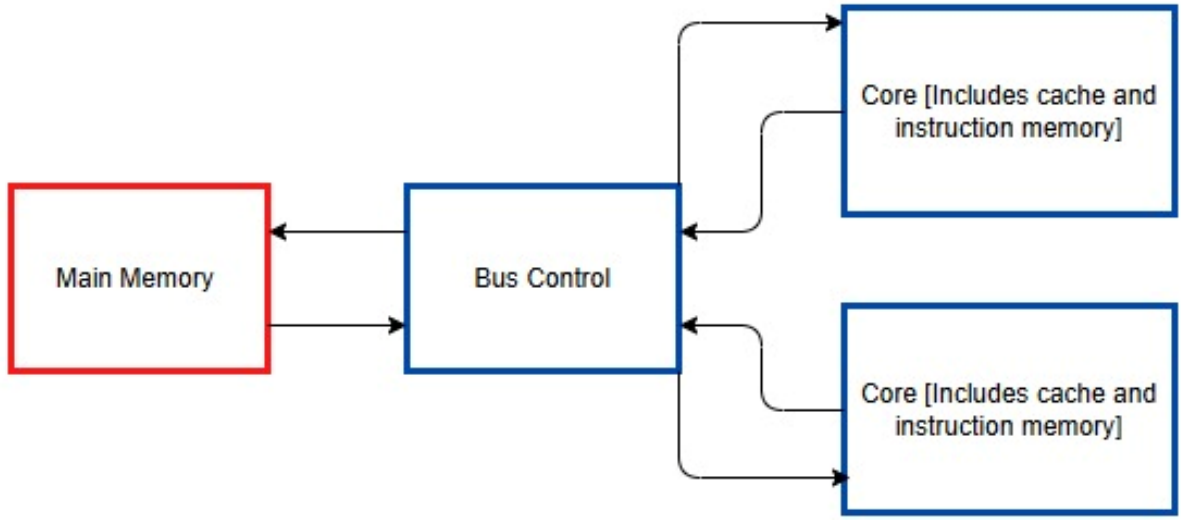
- ***All the blocks and hardware components were implemented using Verilog HDL or schematic drawing over Xilinx's ISE software (shown in the upper picture on the left) and Microsoft's VScode EDA (for Verilog only)***
- ***The programs wrote in Assembly language for DLX architecture, as it was defined in the ACSL course, and compiled to machine code using the RESA software***
- ***Testing scripts and testbenches were written in Verilog and simulated using Xilinx's Isim (shown in the lower picture) platform***
- ***The Main Memory and additional components were provided to us by the staff of the ACSL course and used as black boxes***
- ***Also, we used basic components and logic gate, which were included in the ISE built-in library***





Implementation

Detailed architecture – top level



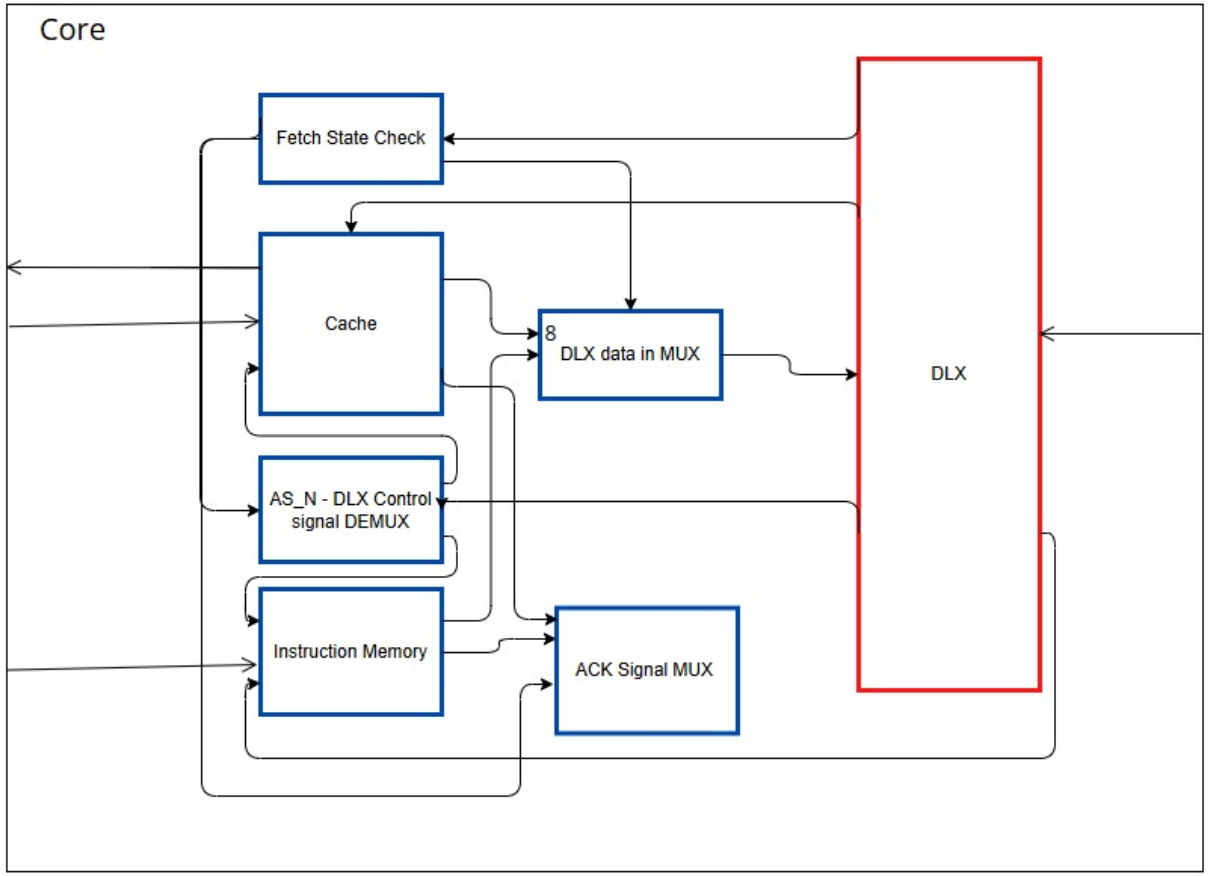
- ***Main Memory** – joint for both core*
- ***Bus control** – performing arbitration, data routing and signal transaction handling*
- ***Core** – includes a DLX simple processor, instruction memory unit and cache.*

** Red blocks are black boxes – used for the project but wasn't made by us. Blue blocks were made by us*

Implementation

Detailed architecture – processor core structure

- ***DLX** – simplified processor*
- ***Cache** – fast access memory of limited local memory blocks. Includes MESI protocol*
- ***Instruction memory** – separated memory unit for instruction only*
- ***Fetch State check** – Indicates if DLX read instruction is in progress.*

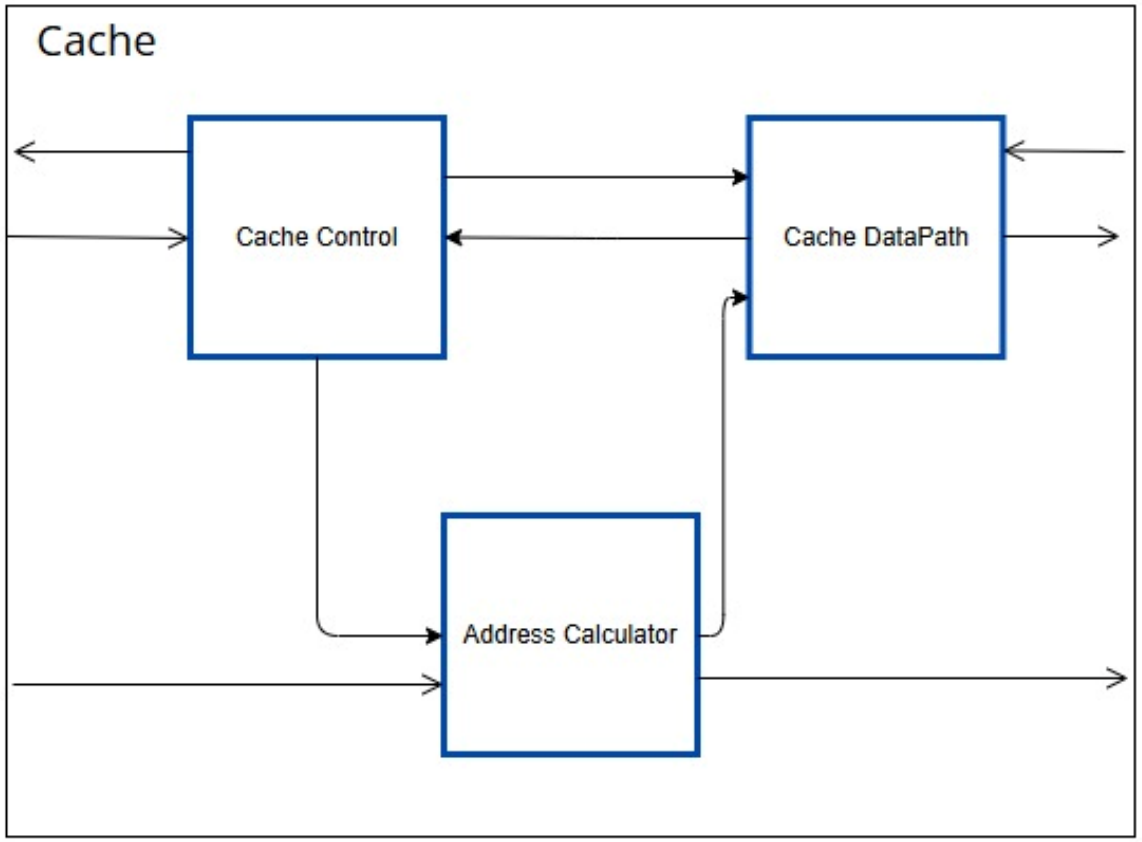


The DLX was implemented during the ASCL lab.

Implementation

Detailed architecture – cache internal structure

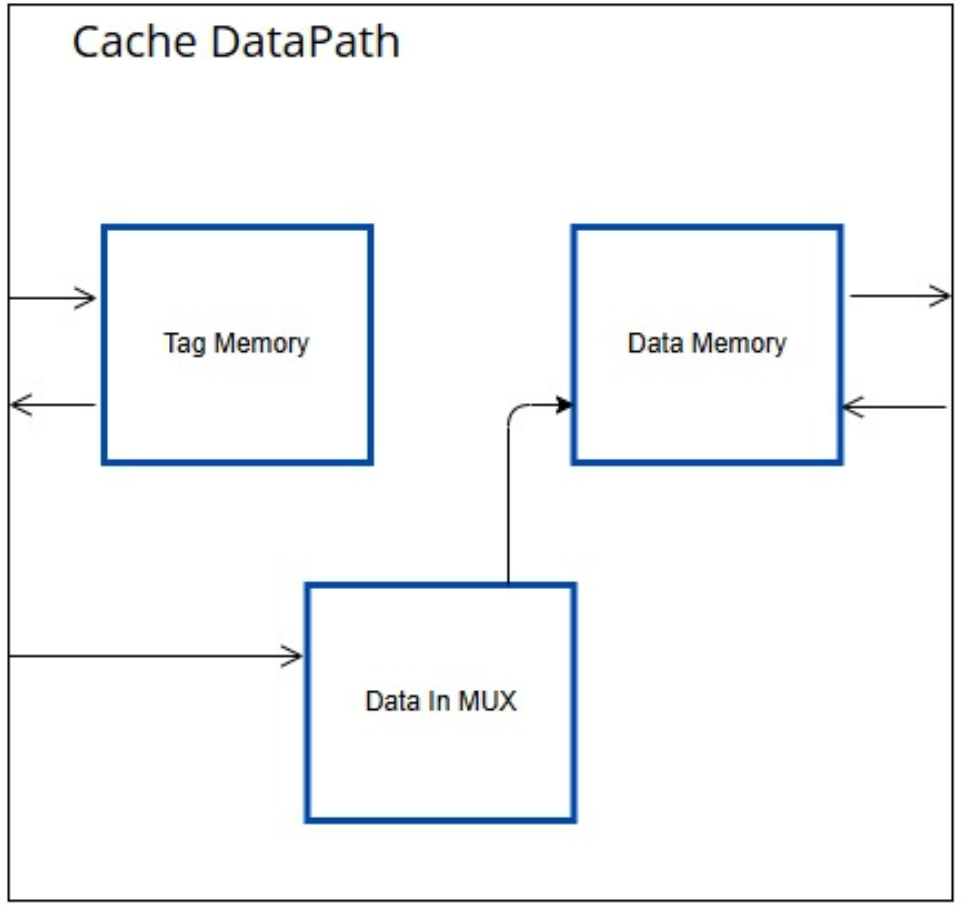
- ***Cache control*** *- managing the cache and Initiate memory transactions*
- ***Cache DATAPATH*** *– Holds data and includes a bus snooping mechanism.*
- ***Address Calculator*** *– calculates the requested address based on the various situations*



Implementation

Detailed architecture – the cache datapath

- ***Tag memory*** *- Contains overhead data like MESI states and bus snooper responsibilities*
- ***Data memory*** *– Holds the data*
- ***Data in MUX*** *– Selects the data source/destination – the DLX data or the Bus data*



Validation

Verification coverage

To ensure the correctness of the designed multicore, several tests were conducted using targeted program to achieve full coverage. The tests' results summarized in the next table:

Unit under test	Test/Tests	Tested by -	Notes
Instruction memory unit	Small input handling	Very short program	
	Very large input handling	Very long program	
	Parallel execution by multiple cores	Parallel programs	
Cache unit	Data transmission from the main memory	Read/Write instructions on missing blocks ("cache miss")	On single-core DLX with cache extension
	Data transmission to the core	Read/Write instructions on existing blocks ("cache hit")	On single-core DLX with cache extension
	Updating data in the main memory	Read/Write instructions on missing blocks, while its place in the cache is occupied ("clearing "dirty" block")	On single-core DLX with cache extension
MESI Unit	Transition I to S, E, M	Read/Write instructions	
	Transition S to I	Read/Write was performed by another core	
	Transition S to E, M	Read/Write instructions	
	Transition E, M to I, S	Read/Write was performed by another core	
	Transition E, M to E, M	Read/Write instructions	Both by read/write to the same block and by calling to another block
Bus controller	Correct arbitration between the cores	Various combination of bus usage request from the cores in every possible order	
	Data transmission		
	Signal transmission	IN_INIT, ack_N, etc. from/to each MESI unit and main memory	
	Snooping	Read/Write instructions	This test is for both bus control and MESI units

Validation

Correctness example

The design maintains correctness through parallelism thanks to the MESI protocol.

Example workload – bubble sort program:

The unsorted array - before the execution

Memory Dump

ADDRESS	VALUES							
0x00000000	0x000000a2	0x000000c4	0x000000e9	0x000000b3	0x00000071	0x00000087	0x000000f7	0x00000034
0x00000008	0x00000035	0x00000079	0x00000096	0x000000ea	0x000000d1	0x00000075	0x0000000b	0x000000e0
0x00000010	0x000000dd	0x0000005b	0x0000004f	0x000000ec	0x000000de	0x00000081	0x000000c9	0x000000af
0x00000018	0x000000d9	0x000000c3	0x00000062	0x0000009a	0x000000a5	0x0000001a	0x000000b8	0x00000044

The sorted array – after the execution

Memory Dump

ADDRESS	VALUES							
0x00000030	0x0000000b	0x0000001a	0x00000034	0x00000035	0x00000044	0x0000004f	0x0000005b	0x00000062
0x00000038	0x00000071	0x00000075	0x00000079	0x00000081	0x00000087	0x00000096	0x0000009a	0x000000a2
0x00000040	0x000000a5	0x000000af	0x000000b3	0x000000b8	0x000000c3	0x000000c4	0x000000c9	0x000000d1
0x00000048	0x000000d9	0x000000dd	0x000000de	0x000000e0	0x000000e9	0x000000ea	0x000000ec	0x000000f7

Performed on the FPGA via RESA software

Results

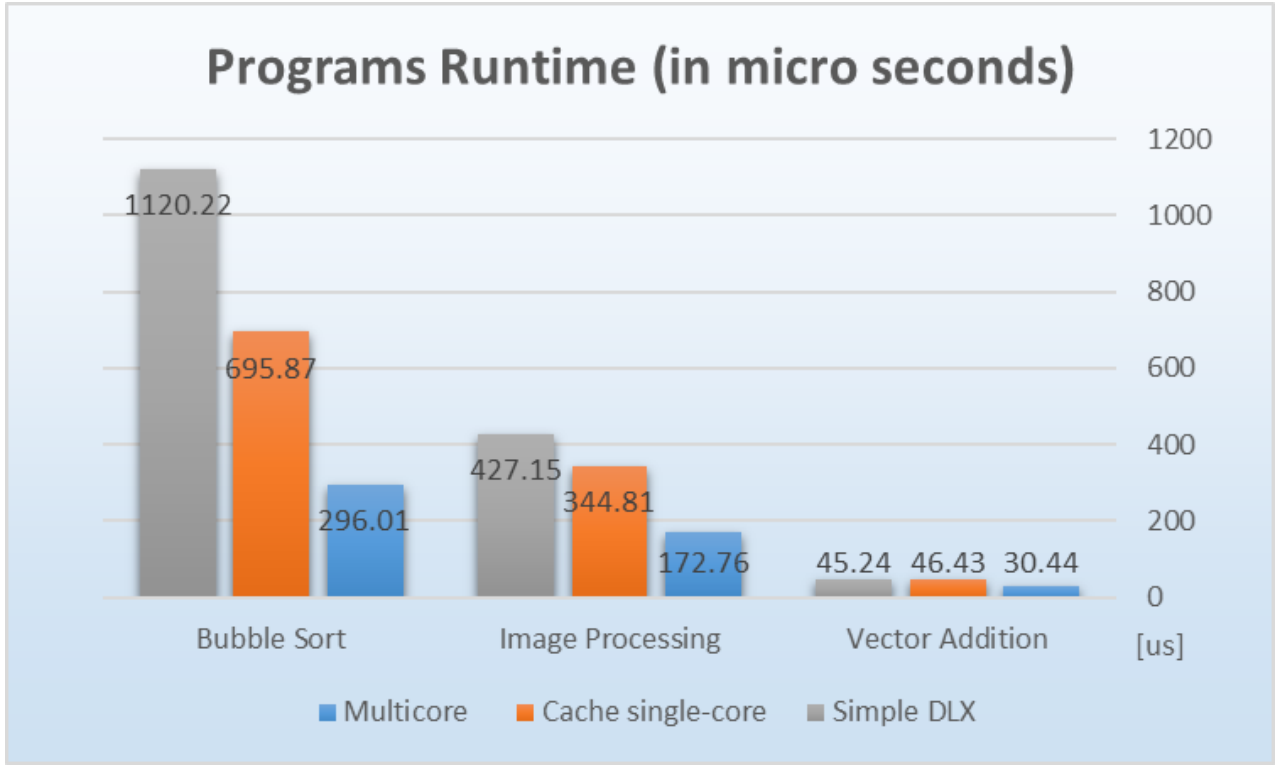
Benchmark

To examine the design improvement, compared to the original DLX processor, a benchmark was made, testing various workloads. As a control tester, the programs performed by a cache single core processor, to verify the impact of parallelism in the results.

The benchmark contained three main programs:

- ***Bubble sort program***
- ***Simple image processing task (segmentation)***
- ***Vector summing***

The runtime of those programs is summarized in the graph on the left, showing major improvement by the multicore compared to both cached single core processor and the original DLX, demonstrating the significancy of the parallelism in the speedup the system achieved

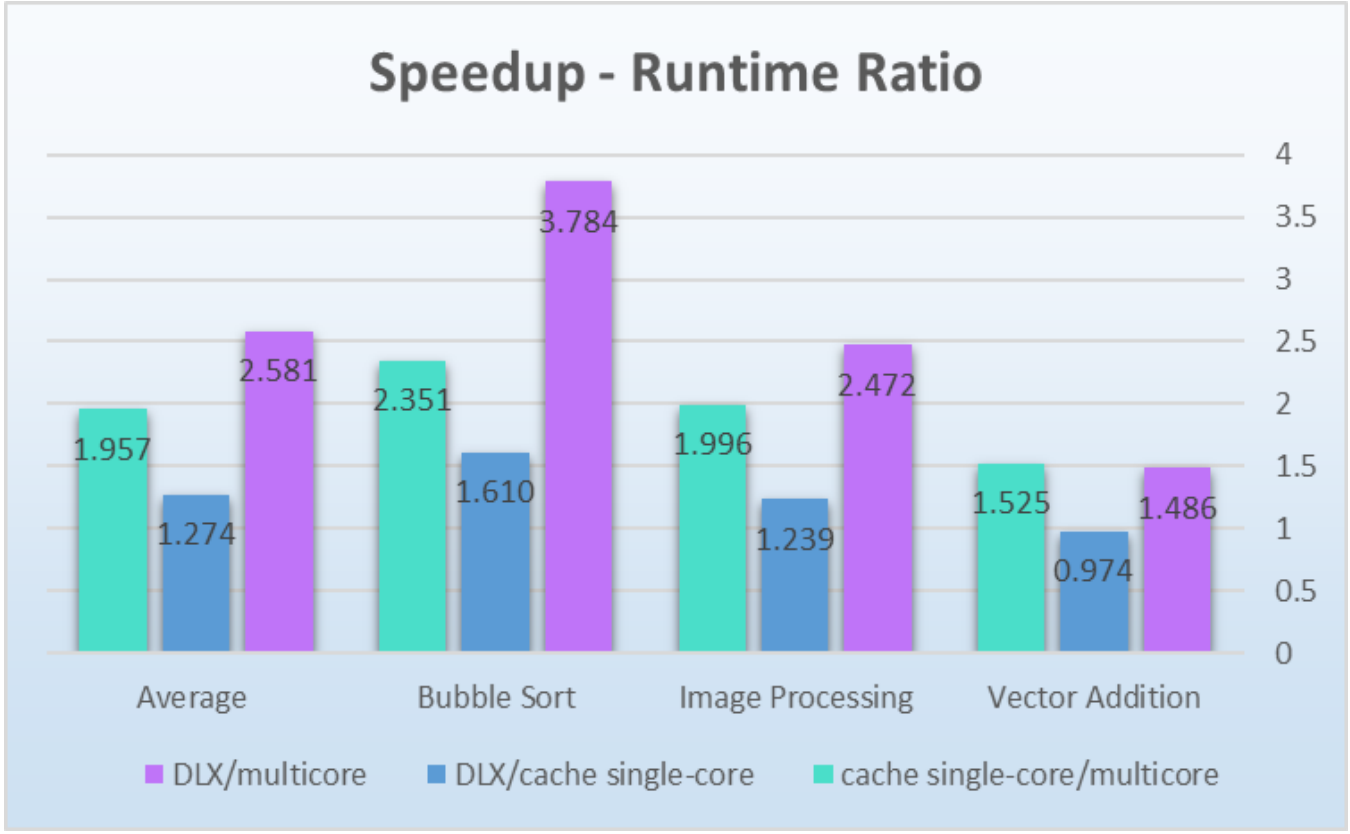


Multicore DLX with coherence protocol – Project Presentation

Results

Performance analysis:

- ***The benchmark shows significant improvement, up to x3.78 in runtime by the multicore processor.***
- ***The mild improvement of the cached single core shows that while other improvements (caches, instruction memory) also contributed, the parallelism was the most significant factor in the superiority of the system***
- ***Averaging speedup of x2.58, the multicore design undoubtedly achieved the original goal of the project in performance improvement***



Further work

Multiprocessor

Improvements in multicore level:

- *Higher level of parallelism – more than two cores in the system*
- *Exploring the advantages of more advanced protocols, such as “Dragon" protocol*
- *Integrate MOESI for reduced memory traffic.*
- *Scale, Coordination and synchronization*

Single processor

Improvements in single-core level:

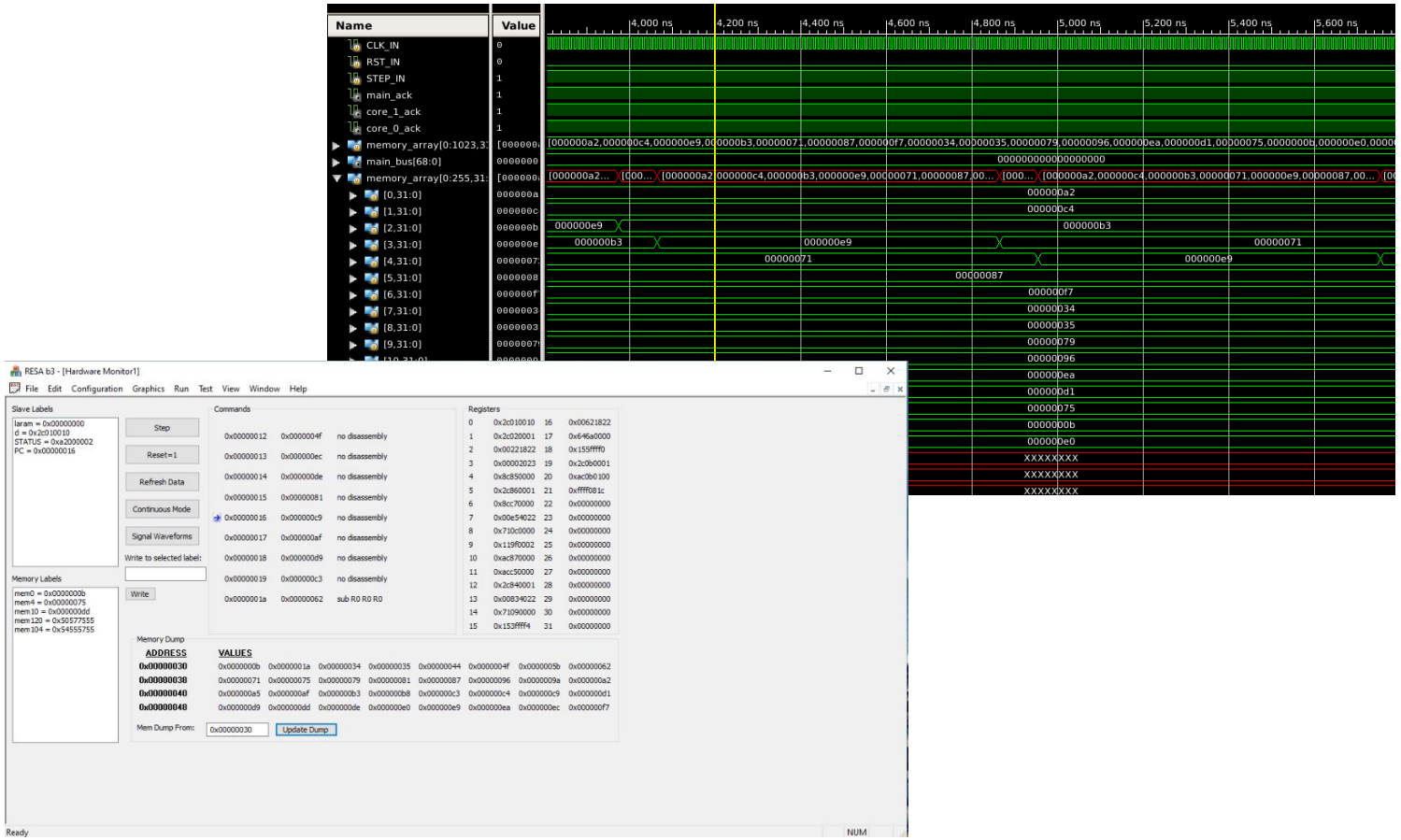
- *Pipelining processors for higher throughput*
- *Out-of-order execution to reduce stalls*
- *Branch prediction to reduce loss of pipelined execution through branch instructions*
- *ILP – instruction level parallelism*

Demonstration

Bubble sort live execution

In simulation

on the FPGA board



Isim and RESA (illustration)

Documentation

GitHub