Projects List

1 Milestones deadlines

Milestone	Deadline
Milestone 1	Tuesday Nov 7th
Milestone 2	Tuesday Nov 21st
Milestone 3	Friday Dec 8th

Table 1: Deadlines

All submissions must be through github with the assignment link in each project.

2 A C++ Model of the ARM AMBA AXI4 Interconnect

2.1 Focus

Interconnect (Bonus: Coherence)

2.2 General

In this project, you will get the chance to learn, and prototype one of the dominant interconnect protocols in Systems-on-Chip, which is the ARM's AMBA AXI4 protocol. You are asked to prototype a cycle-accurate model of the AXI4 protocol and test it by integrating it into an open-source simulator, Octopus. Octopus is a modular cache-coherent interconnect and cache hierarchy simulator that enables you to add extra components (such as interconnect models) fairly easily by methods of inheritance.

2.3 Pre-Requisites

- ◇ C++ experience and OOP concepts

2.4 Actions needed throughout the project

- ◇ Familiarity with the Octopus simulator
- ◇ Understand the AXI4 protocol (documentation is publicly available [2])
- ◇ Implement the AXI4 protocol interconnect class(es) that can be tested in-isolation (without the simulator)
- ◇ Integration to Octopus and testing using given set of traces/benchmarks.

2.5 Bonus

- ◇ Add ACE coherence extensions to the AXI4 protocol [1]

2.6 Project Classroom Assignment Repo

https://classroom.github.com/a/1qVQVFvh

2.7 Milestones

◇ Milestone 1: one page ieee format with the following sections:

- Introduction: high-level context of the problem
- solution: what you will do with a UML diagram for your software modelling architecture and different classes/interfaces

◇ Milestone 2: Standalone simulator with testing

◇ Milestone 3 (Final):

- Integration with Octopus
- a 5-page IEEE paper format report

3 A C++ Model of the TileLink Interconnect Specification

3.1 Focus

Interconnect (Bonus: Coherence)

3.2 General

In this project, you will get the chance to learn, and prototype one of the promising open alternatives of ARM's AMBA AXI4 SoC interconnect, which is TileLink [6] from SiFive, one of the main industry player in the RISC-V open hardware movement. TileLink is a chip-scale interconnect standard providing multiple masters with coherent memory-mapped access to memory and other slave devices. TileLink is designed for use in a System-on-Chip (SoC) to connect general-purpose multiprocessors, co-processors, accelerators, DMA engines, and simple or complex devices, using a fast scalable interconnect providing both low-latency and high-throughput transfers. TileLink: is a free and open standard for tightly coupled, low-latency SoC buses. It is designed mainly for RISC-V but supports other ISAs as well. In particular, you are required to implement the TileLink Uncached Heavyweight (TL-UH) version of the specification, which does not include the coherence specification.

3.3 Pre-Requisites

◇ C++ experience and OOP concepts

3.4 Actions needed throughout the project

◇ Familiarity with the Octopus simulator

◇ Understand the TileLink protocol (documentation is publicly available [6])

- ◇ Implement the TileLink protocol interconnect class(es) that can be tested in-isolation (without the simulator)
- ◇ Integration to Octopus and testing using given set of traces/benchmarks.

3.5 Bonus

- ◇ Add the TileLink Cached (TL-C) support to the protocol.

3.6 Project Classroom Assignment Repo

`https://classroom.github.com/a/1qVQVFvh`

3.7 Milestones

- ◇ Milestone 1: one page ieee format with the following sections:
  - Introduction: high-level context of the problem
  - solution: what you will do with a UML diagram for your software modelling architecture and different classes/interfaces
- ◇ Milestone 2: Standalone simulator with testing
- ◇ Milestone 3 (Final):
  - Integration with Octopus
  - a 5-page IEEE paper format report

4 A NoC Simulation Model

4.1 Focus

Interconnect (NoC specifically, Bonus: Ring, Mesh, or other simpler NoC formats)

4.2 General

In this project, you will get the chance to learn, and prototype A Network-on-Chip (NoC) on an open simulation platform. The NoC is a modular and scalable way of communicating among processing elements in modern SoCs. You basically have links and routers that route the packages from the agents (which are the processing elements). Inside the routers, there can be several arbitration policies to pick a message from one of the sources and map it to the destination. Arbitration is needed since several sources (input links) might be targeting the same output destination (output link).

4.3 Pre-Requisites

- ◇ C++ experience and OOP concepts

4.4 Actions needed throughout the project

- ◇ Familiarity with the Octopus simulator
- ◇ Understand the NoC basics.
- ◇ Implement the NoC interconnect class(es) that can be tested in-isolation (without the simulator)
- ◇ Integration to Octopus and testing using given set of traces/benchmarks.

4.5 Bonus

- ◇ Add the ability to create variations of the topologies (e.g. through inheritance). For example, a ring or a mesh interconnect is a topology that restricts which agents are connected and where routers are placed.

4.6 Project Classroom Assignment Repo

https://classroom.github.com/a/1qVQVFvh

4.7 Milestones

- ◇ Milestone 1: one page ieee format with the following sections:
  - Introduction: high-level context of the problem
  - solution: what you will do with a UML diagram for your software modelling architecture and different classes/interfaces
- ◇ Milestone 2: Standalone simulator with testing
- ◇ Milestone 3 (Final):
  - Integration with Octopus
  - a 5-page IEEE paper format report

5 Data Prefetchers

5.1 Focus

Data Prefetchers (both static and dynamic)

5.2 General

In this project, you will get the chance to learn, and prototype data prefetchers on an open simulation platform. Prefetching is a technique used by computer processors to boost execution performance by fetching instructions or data from their original storage in slower memory to a faster local memory before it is actually needed. The *Primer on Hardware Prefetching* book [4] is a small but excellent book for knowing more about hardware prefetching techniques. This seminal paper [7] is another excellent source.

5.3 Pre-Requisites

◇ C++ experience and OOP concepts

5.4 Actions needed throughout the project

◇ Familiarity with the Octopus simulator

◇ Understand the Hardware Prefetching basics.

◇ Implement Two hardware prefetchers of your choice (where one of them at least must be a dynamic prefetcher) as child classes of a general parent class that represent the basic prefetch functionality/interface. This must be able to be tested in-isolation (without the Octopus simulator)

◇ Integration to Octopus and testing using given set of traces/benchmarks.

5.5 Bonus

◇ Any third hardware prefetcher would be accepted as a bonus

5.6 Project Classroom Assignment Repo

https://classroom.github.com/a/1qVQVFvh

5.7 Milestones

◇ Milestone 1: one page ieee format with the following sections:

- Introduction: high-level context of the problem
- solution: what you will do with a UML diagram for your software modelling architecture and different classes/interfaces

◇ Milestone 2: Standalone simulator with testing

◇ Milestone 3 (Final):

- Integration with Octopus
- a 5-page IEEE paper format report

6 Hyber-Dimensional Computing (HDC) Accelerator Engine

6.1 Focus

Accelerators and Shared Memory Communication

6.2 General

In this project, you are required to write a verilog implementation of an HDC accelerator in FPGA and interface it to a microplaze processor. The microplaze processor will serve as the main CPU that initalizes the matrices and passes a pointer/memory address to the accelerator to operate on, and put the result in another memory location, and return the address of the result to the CPU. It is important that the accelerator parallelizes the computations (otherwise, it will not be in fact an accelerator). Your target FPGA in Vivado should be ZCU102. The test programs should be run baremetal on the Microplaze core. There are several algorithms to conduct several functional units of a HDC accelerator, you pick the ones of your choice.

6.3 Pre-Requisites

◇ Verilog HDL experience

◇ FPGA (Xilinx Vivado) experience is also helpful

6.4 Bonus

◇ Implement two of the algorithms and compare their effectiveness.

6.5 Project Classroom Assignment Repo

https://classroom.github.com/a/GGfIU1u0

6.6 Milestones

◇ Milestone 1: one page ieee format with the following sections:
  - Introduction: high-level context of the problem
  - solution: what you will do with a block diagram of the HDC accelerator and system architecture

◇ Milestone 2: HDC accelerator in standalone with testing

◇ Milestone 3 (Final):
  - Integration of accelerator with microplaze
  - a 5-page IEEE paper format report

7 Matrix-Matrix Multiplication Accelerator

7.1 Focus

Accelerators and Shared Memory Communication

7.2 General

In this project, you are required to write a verilog implementation of a matrix-matrix multiplication accelerator in FPGA and interface it to a microplaze processor. The microplaze processor will serve as the main CPU that initalizes the matrices and passes a pointer/memory address to the accelerator to read the two matrices, multiply them, put the result in a third matrix, and return the address of the result to the CPU. It is important that the accelerator parallelizes the computations (otherwise, it will not be in fact an accelerator). Your target FPGA in Vivado should be ZCU102. The test programs should be run baremetal on the Microplaze core. There are several algorithms to accelerate the matrix multiplication operation in the literature, you can pick any of them.

7.3 Pre-Requisites

- ◇ Verilog HDL experience
- ◇ FPGA (Xilinx Vivado) experience is also helpful

7.4 Bonus

- ◇ Implement two of the algorithms and compare their effectiveness.

7.5 Project Classroom Assignment Repo

`https://classroom.github.com/a/GGfIU1u0`

7.6 Milestones

- ◇ Milestone 1: one page ieee format with the following sections:
  - Introduction: high-level context of the problem
  - solution: what you will do with a block diagram of the MM Multiplication accelerator and system architecture
- ◇ Milestone 2: matrix-matrix multiplication accelerator in standalone with testing
- ◇ Milestone 3 (Final):
  - Integration of accelerator with microplaze
  - a 5-page IEEE paper format report

8 Analyzing the TileLink Coherence on the RocketChip SoC

In this project, you will use the Firesim [1] [5] infrastructure to deploy, and simulate the RocketChip [2] [3] on FPGA. The main purpose is to analyze the operation and latencies of the TileLink interconnect [6] on the Rocket chip SoC.

[1] `https://docs.fires.im/en/stable/index.html`
[2] `https://chipyard.readthedocs.io/en/stable/Generators/Rocket-Chip.html`

8.1 Pre-Requisites

◇ Verilog HDL experience

◇ FPGA (Xilinx Vivado) experience is also helpful

8.2 Bonus

◇ TBD

8.3 Project Classroom Assignment Repo

https://classroom.github.com/a/GGfIU1u0

8.4 Milestones

◇ Milestone 1: one page ieee format with the following sections:

- Introduction: high-level context of the problem
- solution: what you will do with a plan on how to analyze the protocol latencies in the architecture

◇ Milestone 2: Having a working simulation and onboard implementation of the platform

◇ Milestone 3 (Final):

- Analysis of different latencies of TileLink protocol in the RocketChip.
- a 5-page IEEE paper format report

9 Inter-VM communication in Heterogeneous SoCs

In this project, you will explore one of the important layers in modern SoCs, which is Hypervisors. In particular, you will work with the Xvisor hypervisor [3]. Xvisor works both for ARM and RISC-V platforms. In the project you are asked to run Xvisor on the ARM PS side of the FPGA SoC and also run it on a CVA6 RISC-V core on the FPGA side and enable them to talk to each other through shared memory.

9.1 Pre-Requisites

◇ Verilog HDL experience

◇ FPGA (Xilinx Vivado) experience is also helpful

9.2 Bonus

◇ TBD

[3] https://github.com/xvisor/xvisor

9.3 Project Classroom Assignment Repo

https://classroom.github.com/a/GGfIU1u0

9.4 Milestones

◇ Milestone 1: one page ieee format with the following sections:

  - Introduction: high-level context of the problem
  - solution: what you will do with high-level diagram of the xvisor and modules you need to touch.

◇ Milestone 2: Xvisor in cva6 standalone (without ps side communication)

◇ Milestone 3 (Final):

  - The system with xvisor in both arm ps and cva6 pl and communicate together
  - a 5-page IEEE paper format report

References

[1] ARM. AMBA AXI and ACE Protocol Specification. https://documentation-service.arm.com/static/5f915905f86e16515cdc333b, 2023.

[2] ARM. AMBA AXI Protocol Specification. https://developer.arm.com/documentation/dui0534/b/?lang=en, 2023.

[3] Krste Asanovic, Rimas Avizienis, Jonathan Bachrach, Scott Beamer, David Biancolin, Christopher Celio, Henry Cook, Daniel Dabbelt, John Hauser, Adam Izraelevitz, et al. The rocket chip generator. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17*, 4:6–2, 2016.

[4] Babak Falsafi and Thomas F Wenisch. *A primer on hardware prefetching.* Springer Nature, 2022.

[5] Sagar Karandikar, Howard Mao, Donggyu Kim, David Biancolin, Alon Amid, Dayeol Lee, Nathan Pemberton, Emmanuel Amaro, Colin Schmidt, Aditya Chopra, et al. Firesim: Fpga-accelerated cycle-exact scale-out system simulation in the public cloud. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 29–42. IEEE, 2018.

[6] SiFive. SiFive TileLink Specification. https://static.dev.sifive.com/docs/tilelink/tilelink-spec-1.7-draft.pdf, 2023.

[7] Steven P Vanderwiel and David J Lilja. Data prefetch mechanisms. *ACM Computing Surveys (CSUR)*, 32(2):174–199, 2000.