The Trireme Platform Manual v1.1

Adaptive and Secure Computing Systems (ASCS) Laboratory
Computer Architecture & Embedded Systems (CAES) Laboratory
Secure, Trusted, and Assured Microelectronics (STAM) Center

September 17, 2021

Contents

Chapter 1

Introduction

The Trireme Platform contains all the tools necessary for register-transfer level (RTL) architecture design space exploration. The platform includes RTL, example software, a toolchain, and configuration graphical user interface (GUI). All parts of the platform are open-source and available for download at `https://ascslab.org/research/brisc v/index.html` The platform is designed with a high degree of modularity. It provides highly-parameterized, composable RTL modules for fast and accurate exploration of different RISC-V based core complexities, multi-level caching and memory organizations, system topologies, router architectures, and routing schemes. The platform can be used for both RTL simulation and FPGA based emulation. The hardware modules are implemented in synthesizable Verilog using no vendor-specific blocks. The platform's RISC-V compiler toolchain is used to develop software for the cores. A web-based system configuration (GUI) can be used to rapidly generate different processor configurations. The interfaces between hardware components are carefully designed to allow processor subsystems such as the cache hierarchy, cores or individual pipeline stages, to be modified or replaced without impacting the rest of the system. The platform allows users to quickly instantiate complete working RISC-V multi-core systems with synthesizable RTL and make targeted modifications to fit their needs.

Chapter 2

Getting Started Guide

This chapter contains describes how to simulate and synthesize the Trireme demonstration systems as well as how to write, compile, and run software targeting these systems. Most of this chapter assumes that software is written for the "Many Poor Processor" demonstration architecture described in Section 3.0.2. The text will note when commands and instructions are not for the "Many Poor Processor" demonstration.

2.1 Simulating the Trireme Platform with Modelsim

The Trireme Platform includes scripts to simulate the included test benches with Modelsim from the command line. Simulations can also be performed with the Modelsim GUI. Explanation of simulation in the Modelsim GUI is beyond the scope of this document. See Modelsim specific documentation or tutorials for more specific information.

Directions for installing Modelsim are also beyond the scope of this document and are not provided here. The directions in this section assume Modelsim is properly installed on your machine. The following directions describe how to use the included scripts to simulate a Trireme test bench.

Adding Modelsim to your PATH

The included scripts assume that your Modelsim installation is included in your $PATH variable. To add Modelsim to your $PATH variable, execute the following command or add it to your bashrc file:

Listing 2.1: Adding Modelsim to your $PATH variable.

Note that the command above assumes you have the Altera Starter Editoin v18.1 of Modelsim installed in the default location (your home directory). If you have a different version of Modelsim or if you have changed the install location, update the command to include the correct path to the installation's bin directory.

Compiling Modules in Modelsim

A TCL script is used to compile a test bench and the modules it instantiates. the provided load.do script in the modelsim directory will compile every module and test bench in the Trireme Platform with the appropriate arguments.

By default the TCL variable comple_arg is added to each compile command to provide compiler arguments common to each module. Currently this variable is used to define a Verilog macro that points modules to an include file with more Verilog parameters. If you wish to simulate custom modules with the provided scripts, add the following TCL command to modelsim/load.do to compile the modules and add them to the modelsim library named "work".

Listing 2.2: Compie a custom module in the load.do script

Simulating Modules

You can simulate any test bench in the Trireme Platform with the run_test script in the modelsim directory. The run_test script must be executed from the modelsim directory. When the run_test script is executed, first,

it will execute the load.do TCL script to compile all of the modules. Then the script will simulate all of the test benches entered as command line arguments.

Example usage of the script is shown in the listing below:

Listing 2.3: Example usage of the run_test script

Including .vmh Files

Make sure to include any necessary memory image (.mem or .vmh) files in the modelsim/binaries directory before simulating a test bench. Modelsim assumes file paths given the to the $readmemh system task are relative to the location Modelsim was launched from. To simulate modules in the Trireme Platform, Modelsim should always be launched from the modelsim directory. The default memory image paths in the included test benches are relative to the modelsim directory.

2.2 Synthesizing the Trireme Platform with Quartus

This section describes how to generate and use Quartus projects for the included demonstration systems. TCL scripts are included to generate a project for each demonstration system, removing the need to manually build a project for each one. Also included with each demonstration system is a Synopsys Design Constraint (.sdc) file to constrain the timing of each input/output pin in a system.

Adding Quartus to your PATH

The included scripts to generate a Quartus project assume that your Quartus installation is included in your $PATH variable. To add Quartus to your $PATH variable, execute the following command or add it to your bashrc file:

Listing 2.4: Adding Quartus to your $PATH variable.

Note that the command above assumes you have the Quartus version v18.1 installed in the default location (your home directory). If you have a different version of Quartus or if you have changed the install location, update the command to include the correct path to the installation's bin directory.

Generating a Project

Each sub-directory in the quartus directory is named for the demonstration system it implements and the FPGA development board it targets. An FPGA-specific top module is included in each demonstration project (in the src directory). The FPGA-specific top module handles clock and reset generation and any device specific I/O functions. To generate a Quartus project for a demonstration system, navigate to the appropriate demonstration system sub-directory in the quartus directory. The remainder of this section will assume we are generating the **many_poor_processor_DE5_net** project.

Listing 2.5: Navigate to the demonstration project directory.

The gen_project script in each demonstration project directory is hard coded to execute the project TCL script and create a Quartus project. To generate a project, simply execute:

Listing 2.6: Generate a demonstration system Quartus Project.

The script will create a Quartus Project File (.qpf) and a Quartus Settings File (.qsf). Now the project (.qpf file) can be opened with the Quartus GUI or synthesized with the synth script.

Synthesizing a Project

After creating the quartus project, execute the synth script to synthesize the project and generate a bit-stream.

Listing 2.7: Synthesize a demonstration system.

The bit-stream will be output in a newly created output_files directory with the file extension .sof. Device resource usage can be seen by executing:

Listing 2.8: Print device resource usage.

The designs maximum clock frequency (Fmax) can be found by opening the many_poor_processor_DE5_net.sta.rpt file in output_files and searching for "Fmax".

The Poor Processor and Many Poor Processor include a default bootloader program to load another program over a UART. If you change the program that is initialized in FPGA BRAM memory, be sure to include the binary in demonstration project directory. Note that the path used to initialize FPGA BRAM with the $readmemh system task is relative to the directory that the synth script is launched from. The synth script should only be launched from it's demonstration project directory, i.e., with the command in Listing 2.7.

Configuring a Bit-Stream

Currently no scripts are provided to load a bit-stream from the command line (although this is certainly possible). For now, bit-streams must be loaded with the Quartus GUI.

2.3 Writing Software for The Trireme Platform

As a complete design space exploration platform, The Trireme Platform provides several example programs and tools to compile them. After a program is written and compiled, it can be executed on the hardware architecture. Currently software must be executed "bare-metal" without an operating system. The Trireme Platform supports a limited subset of the C standard library. Complete standard library support is a work in progress. Dynamic memory allocation is supported with "malloc" and "free" functions. Other standard library features, such as file I/O, have not been implemented yet.

Programming architectures with multiple HARTs are supported with multiple "main" functions for each HART. HART (core) 0 executes the "main" function. Additional HARTs in the system will execute a "hartN_main" function where "N" is the HART number. HARTs that fetch instructions from the same memory can be grouped into a single C file with multiple main functions. However, some systems may not have cores fetch instructions from the same memory (i.e. the same physical BRAM). An example of such a system is a multi-processor system with multiple single or multi-core processors connected to a node of a Network-on-Chip. The "Many Poor Processor" can be considered three single-core processors with a shared data memory space but separate instruction memory address spaces. For such systems, separate binaries must be created for each processor. Section 2.5.1 describes how to merge multiple binaries into a single memory image for the system.

In architectures with memory mapped I/O, such as the "Poor Processor" and "Many Poor Processor" demonstrations, I/O communication is achieved by manually setting a pointer to the I/O device address and performing reads and writes to it. Currently, a UART is the only supported memory mapped I/O device. In the "Poor Processor" and "Many Poor Processor" demonstrations, the UART Tx address is 0x00003020 and the Rx address is 0x00003010.

Writing a char to address 0x00003020 will transmit that byte out the UART port. If the UART buffers are full, the write will automatically wait in the NoC buffer, so software does not need to check any Tx buffer status. Reads to the Tx address are undefined, but will probably read a 0 value.

The Rx address should be treated as a two byte value (a short). The least significant byte is data read out of the UART Rx buffer (which is 0 if the buffer is empty). The most significant byte is a valid signal, indicating that the Rx buffer contained data when the read was performed. See the uart_loopback.c program for example UART code.

2.4 Compiling Software for The Trireme Platform

This section describes how to compile software for The Trireme Platform.

2.4.1 Installing a RISC-V Cross-Compiler

Currently the Trireme Platform uses the standard RISC-V GNU toolchain with Newlib. These directions describe how to install a plain RV32I or RV64I version of the toolchain. See the RISC-V GNU toolchain GitHub page for troubleshooting help if you encounter issues.

To install the toolchain, follow these steps:

1. Make sure the required packages are installed. On Ubuntu run the following command. See the risc-v gnu toolchain git repo for directions on installing dependencies for other Linux distributions.

Listing 2.9: Install prerequisites.

2. Download the latest version of the RISC-V GNU toolchain from GitHub.

Listing 2.10: Clone the RISC-V GNU Toolchain Git repository.

3. Configure the toolchain for RV32I or RV64I and set the installation path prefix. The toolchain will be installed at the location defined by the prefix argument. The Trireme Platform assumes the toolchain is prefix is set to "/opt/riscv". If the toolchain is installed somewhere else, the correct compiler path will have to be set in the compiler wrapper with an additional argument. Section 2.4.2 describes the compiler wrapper and its arguments. Make sure the instalation path exists and that you have permission to write there, otherwise the toolchain will not build.

Listing 2.11: Configure the toolchain to build an RV32i compiler and install it in "/opt/riscv".

4. Run make to build the toolchain and install it at the specified path

Listing 2.12: Build the toolchain.

2.4.2 Compiling Software

Compiling software for the Trireme platform is accomplished through the compiler script "trireme_gcc", which is found within the software directory of the Trireme Platform. "trireme_gcc" is a *wrapper* around the RISC-V GCC (GNU Compiler Collection) . As such, the script provides the same interface one would expect from the GCC program.

The Trireme Platform executes software on "bare-metal" meaning an operating system is not used to set up the execution environment for a program. Instead, the compiled binary must include any environment setup necessary to execute the program properly. Programs compiled with trireme_gcc will automatically have the necessary environment setup needed to run on the Trireme platform.

The compiler script supports all arguments that are supported by the GCC program. These arguments are simply forwarded to GCC for compilation, assembly, and linking. In addition, the script allows the user to specify arguments to control compilation for the Trireme bare-metal environment. Executing `./trireme_gcc` or `./trireme_gcc --help` lists the various compiler script specific arguments. For convenience, GCC and compiler script arguments can be intermixed without disturbing GCC invocation.

The Trireme compiler script arguments fall into three categories: memory layout, libraries, and output files. The majority of arguments belong to the memory layout category, which controls the structure of the compiled program. Before using the compiler script, it's important to know where your tool chain is installed on your machine. By default, the Trireme compiler script assumes the RISC-V tool chain is installed in "/opt/riscv". This was specified when the tool chain configuration script was invoked. A different prefix can be specified through the `--toolchain-prefix` argument.

–start-addr Users can set the start address of the program for a Trireme processor by specifying the `--start-addr`. The start address must be in decimal and aligned on a word boundary (i.e., a multiple of 4). If the Trireme compiler script is invoked without specifying this argument, the default start address of 0x0 will be used. For compilation targeting multiple cores, it's important to note that each core will share the same start address.

–**stack-addr** The program stack address is specified through the `--stack-addr` argument. The stack address is specified in decimal. The stack address must be a multiple of 16 Bytes. In RISC-V, the stack grows from high addresses to low addresses; therefore, this address should take into account the stack size. Trireme does not currently have run-time protection for stack overflow. Therefore, to minimize possible corruption of program data, the stack address should be selected such that there is enough of a gap between the stack end address and the stack start address. For both single core and multi-core configurations, the `--stack-addr` corresponds to the core 0 starting address for the stack. The default value of this argument is 2048. For multi-core configurations, each core is assigned its own stack segment. The compiler script uses the `--stack-size` argument to determine stack start addresses for each core after core 0. Starting from core 0's stack address, each successive core stack address is offset by stack-size bytes from the previous core's stack start address. Values for this argument should be a multiple of 16 and in decimal. The default value for this argument is 2048.

–**heap-size** The Trireme compiler script places the heap segment at the end of the data section. This cannot be changed by the user. However, the user can specify the size of the heap segment with the `--heap-size` argument. Setting this argument to 0 will tell the compiler script that no heap is needed. Values to this argument should be a multiple of four and in decimal. The default value for this argument is 0.

–**num-cores** To target a multi-core design, `--num-cores` option is provided. If the number of cores specified is more than one, the compiler script will expect the entry points defined for each core. Please refer to section 2.3 for more details. By default, the number of cores is set to 1.

–**ram-size** The main memory size of the target platform is specified through the `--ram-size` as a decimal value argument. By default this value is set to 2048. It is important that this value be large enough to contain all the data and text of the program.

–**link-libgloss** Trireme implements a Board Support Package (BSP) which acts as a low level Operating System Support Layer. Trireme BSP is in the form of a library that can be optionally linked to programs with the `--link-libgloss` flag. Linking the BSP enables support for the standard C library, including printing to stdout (e.g, puts, printf) and dynamic memory allocation (e.g, malloc, free). Trireme BSP only supports a subset of the POSIX system calls. As such, not all programs can link successfully for the Trireme platform. Before using this flag, the Trireme BSP must be built; please refer to section 2.4.2.1 for more details.

–**{vmh, dump, raw-binary}** By default, the compiler script will only generate a ".elf" file. Three other output files supported are ".vmh", ".bin" and ".dump". The ".vmh" file can be used to initialize memory in simulation or FPGA's BRAMs in synthesis. Specifying the argument `--vmh <file path>` will generate a vmh file at the provided file path. The ".dump" file contains contents of an objectdump disassembly which can be useful for debugging. Specify `--dump <file path>` to generate a ".dump" file at the given file path. Specify `--raw-binary` to create a raw binary. The elf headers and metadata are removed from this binary so that it can be directly executed when uploaded to an FPGA implementation with a bootloader program. All three arguments can be specified simultaneously.

–**trireme-lib-path** The Trireme compiler script can be invoked from directories other than "software". If the compiler script runs from another directory, the Trireme library path must be specified using the `--trireme-lib-path`. By default this points to the "software/library" directory where the BSP will be installed after being built. Note that this is only necessary if `--link-libgloss` is used.

Example 1: To compile a single core gcd.c program in the "software/applications/src" directory with an initial stack pointer of 4096, ram size of 4 KiB, vmh and raw binary output run:

Listing 2.13: Compiling a program for a single core processor

Example 2: To compile a dual-core program named gcd_fibonacci.c from the "software/applications/src" directory and generate vmh and raw binary files run the command below. This command will set the stack pointer of core 0 to 4096 and set the stack pointer of core 1 to 8192.

```
trireme: compilation summary:
 program path: gcd
 start address: 0 (hex: 0x0000)
 start address on word boundary? yes
 number of cores: 1
 program size: 440 bytes (440.00 B)
 core stack size: 2,048 bytes (2.00 KiB)
 core 0 stack start address: 4,096 (hex: 0x1000)
 core 0 stack end address: 2,052 (hex: 0x0804)
 core 0 stack start address on word boundary? yes
 core 0 stack end address on word boundary? yes
 total stack size: 2,048 bytes (2.00 KiB)
 heap start address: 440 (hex: 0x01b8)
 heap end address: 440 (hex: 0x01b8)
 heap start address on word boundary? yes
 heap end address on word boundary? yes
 heap size: 0 bytes (0.00 B)
 free space size (stack <--> heap gap): 1,612 bytes (1.57 KiB)
 heap region overlaps stack region? no
 memory image total size: 4,096 bytes (4.00 KiB)
 user selected main memory size: 4,096 bytes (4.00 KiB)
 recommended main memory size: 4,096 bytes (4.00 KiB) (bits: 12)
```

Listing 2.15: Compilation summary output

Listing 2.14: Compiling a program for a multi-core processor

After a successful compilation, the Trireme compiler script will output a "compilation summary". Much of the information within the summary aims to prevent possible runtime issues such as stack/heap corruption, instruction memory corruption and irregular program behavior. As shown in listing 2.15, the compiler script displays the start address of the processor, stack address alignment for each core. In addition to heap size and heap address alignment checks, the compiler script calculates the final program size and provides the recommend main memory size suitable for simulation and FPGA deployment.

The compiler script performs several checks on the compiled program. Compilation summary output shows the result of checks in the form of "yes" and "no" answers. The "yes" and "no" answers are colored green for "no action required" and red for "requires attention". As shown in 2.15, all checks passed for the gcd program. Although not shown here, the text color of other lines will change to "red" to signify user intervention is required.

2.4.2.1 Building The Trireme BSP

The Trireme toolchain features low level Operating System support for Bare-metal environments in the form of a board support package (BSP). Low level Operating System Support is implemented as a set of procedures implemented to intercept select POSIX system calls. This allows your application to utilize LibC procedures such as `malloc`. Bare-metal Operating System support can be enabled using the `--link-libgloss` argument.

Building the Trireme BSP is a prerequisite for using the `--link-libgloss` flag. The build process is automated with the help of the "build_bsp" script found within the "software" directory. BSPs are located in the "software/bsp/" directory. Currently only the trireme32 BSP is included. To build a BSP, execute the "build_bsp" script with the BSP name as the first argument. The BSP name is the same as the sub-directory name within the "bsp" directory. Listing 2.16 shows how to build a BSP using the script.

Listing 2.16: Building the Trireme rv32i Board Support Package

At the end of the build process, the BSP build script will automatically install the BSP library within the "software/lib" directory.

Example 3: Compiling an example program which uses `malloc`

2.5 Running Software with The Trireme Platform

This section describes how to run software in a simulation or FPGA implementation of the Trireme Platform.

2.5.1 Program Execution on FPGA Hardware

This section assumes a Trireme Platform architecture running a bootloader program (such as the "Poor Processor" or "Many Poor Processor" demonstration) is configured on FPGA hardware. The architecture is assumed to have a UART connection to a host machine with The Trireme Platform installed on it. See Section 2.2 for instructions on synthesizing and configuring an architecture on an FPGA.

After writing and compiling your program(s) according to the instructions in Sections 2.4 and 2.3 individual program binaries have been created for each processor. The remainder of this section assumes the target architecture is the "Many Poor Processor" demonstration. It is important to make sure your programs have been compiled with different stack pointers so that they do not overwrite eachother. This section assumes the uart_loopback program has been compiled with a stack pointer of 4092, the short_mandelbrot_print program has been compiled with a stack pointer of 8188 and the prime_number_counter program has been compiled with a stack pointer of 12284.

The compiler wrapper generated raw binaries for each of these programs but to create a memory image for the "Many Poor Processor" we must pad these binaries and concatenate them into a single file.

The pad_binary.py script in the software/helper_scripts directory will append zeros to each of the binaries until they reach a specified size. The first argument of the script is the binary name. The second argument is the pad size in bytes. Padding is performed in-place, on the original binary.

Each node in the "Many Poor Processor" has 4096 bytes of memory. Addresses 0x00000000-0x00000300 of each node are reserved for the bootloader application (remember that the node is indexed by address bits 12 and 13). That leaves 3328 bytes of memory at each node for additional program instructions and data memory. To pad each of the binaries to 3328 bytes run the following commands from the software directory:

Listing 2.18: Pad binaries.

The img_cat.sh helper script concatenates each program argument and writes a memory image to a file name generated based on the input program names. To generate a memory image from the padded binaries, run the following from the software directory. The script will write a memory image to "C0_uart_C1_mandelbrot_C2_primes.img".

Listing 2.19: Build a Many Poor Processor Memory Image.

Now that a memory image has been created, it can be uploaded to a compatible architecture FPGA implementation. These directions use the minicom program to communicate with the architecture implementation over a serial port. The serial port connection must be opened with the correct BAUD rate. The default BAUD rate for the "Many Poor Processor" is 115200 bits/second. Generally when a serial port is connected to a host computer it will appear in the /dev directory. These directions assume the serial port device is named /dev/ttyUSB0. Assuming the FPGA is configured with the "Many Poor Processor" architecture running the bootloader program, open minicom with the following command:

Listing 2.20: Start Minicom.

This command will open a console to send and receive data with the FPGA. When the "Many Poor Processor" is reset, the bootloader will print a message indicating that it is waiting for a program to be uploaded. Figure 2.1 shows the default boot message printed on a minicom terminal.

Note: Instead of using sudo to start minicom, you can add yourself to "dialout" group with:

Listing 2.21: Add the current user to the dialout group.

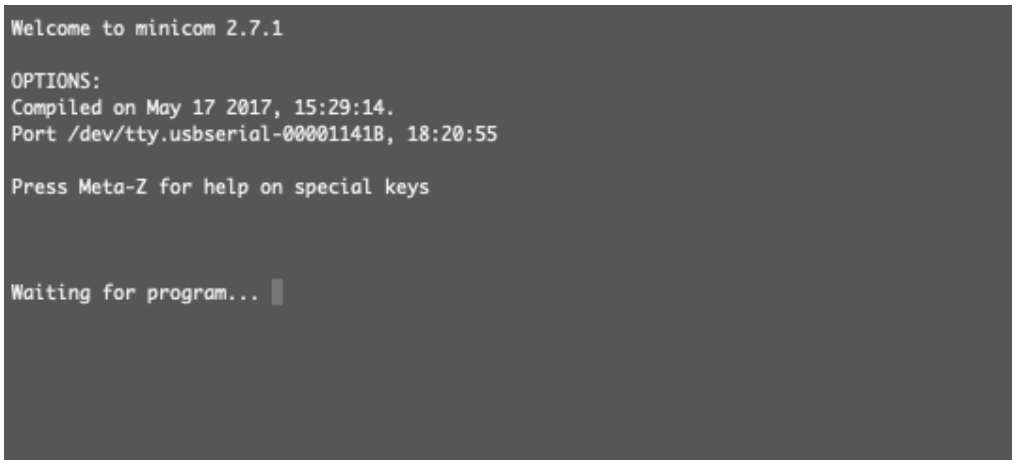

Figure 2.1: The Minicom "send file" file selector. Change directories by selecting the "Goto" menu button, hitting enter and typing the path to your desired directory.

To upload a memory image in minicom, pres CTRL and A at the same time (written as CTRL-A from now on), then press S. This key combination will open the menu shown in Figure 2.2 to send a file. Use arrow keys to highlight the ASCII protocol option from the list, select it by hitting spacebar, then hit enter.

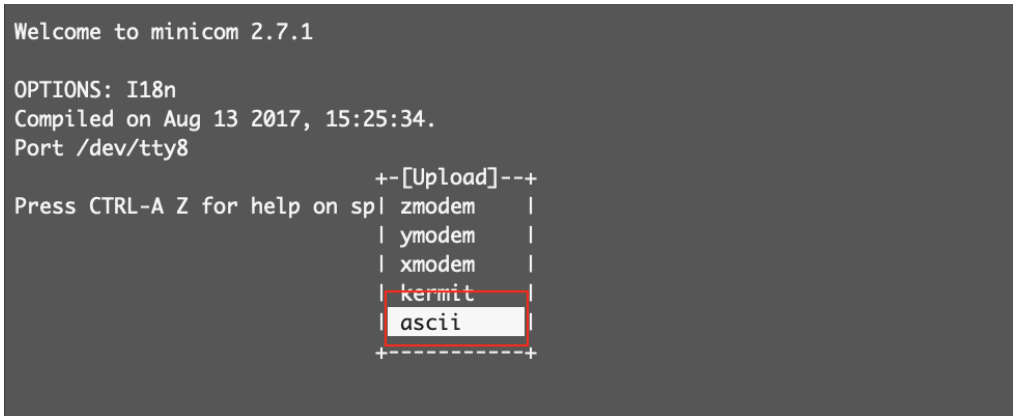

Figure 2.2: The Minicom "send file" menu opened with CTRL-A, S.

Navigate to the directory with the memory image using the GOTO menu option. Select the memory image by highlighting it and hitting spacebar.

Figure 2.3: The Minicom "send file" file selector. Change directories by selecting the "Goto" menu button, hitting enter and typing the path to your desired directory.

Select the memory image to be uploaded, and hit enter to begin the upload.

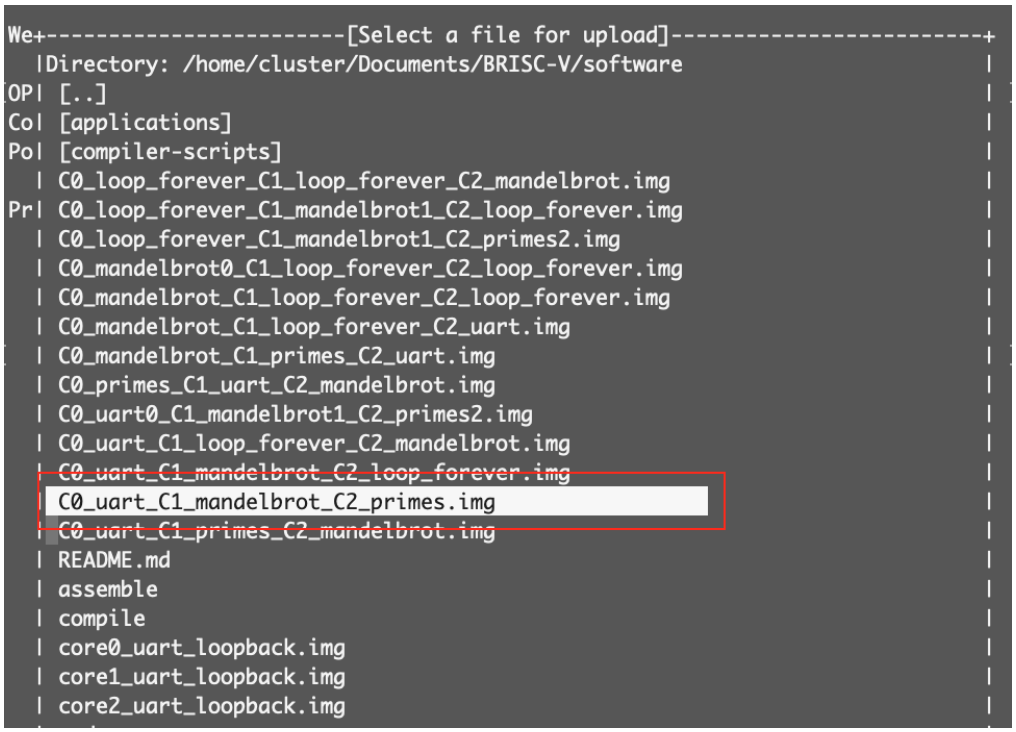

Figure 2.4: Upload the desired image file by using the arrow keys to navigate to the file name. Hit spacebar to select it. Hit enter to begin the upload.

When the upload is complete, hit any key to close the menu.

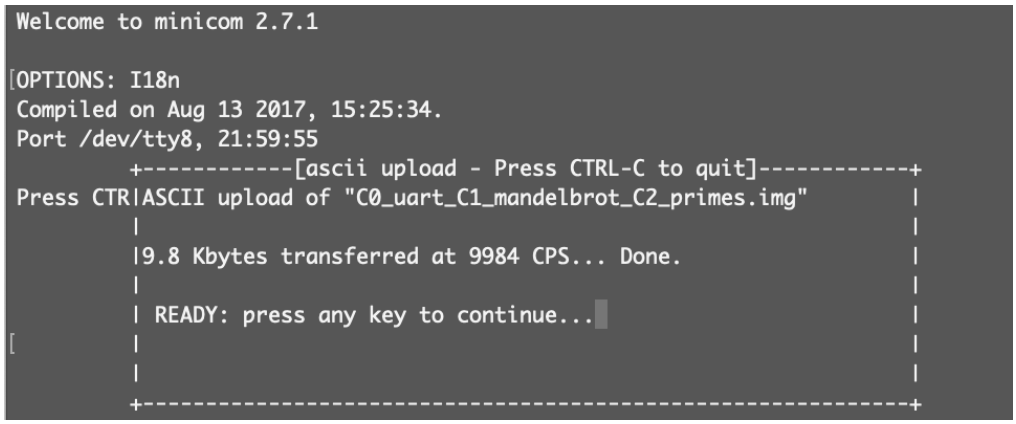

Figure 2.5: After the upload starts, a widow will popup to display the transfer rate. When the transfer is complete, hit any key to exit the window and return to the minicom console.

The bootloader program should print a message indicating the completion of the file upload. After the message prints, the bootloader begins executing the uploaded program.

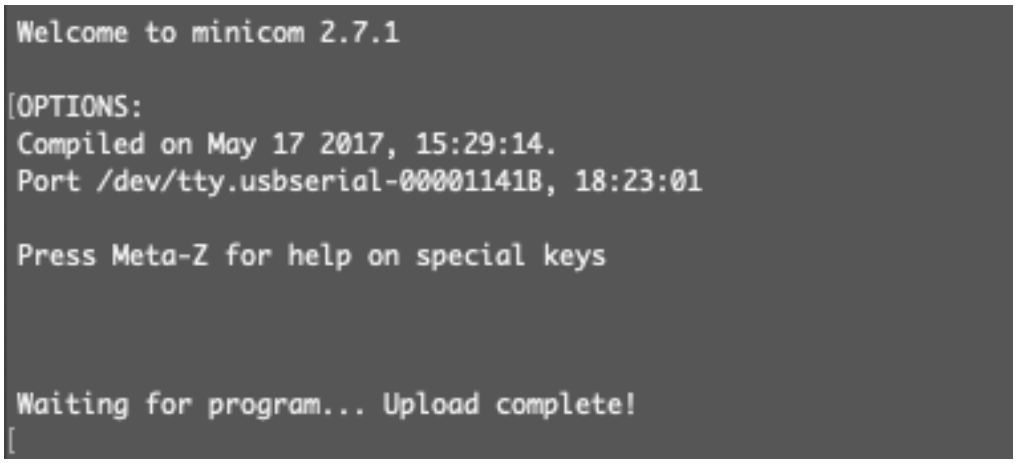

Figure 2.6: The bootloader upload completion message.

2.5.1.0.1 Troubleshooting If you try to upload a file or type in the console but no data is sent over the serial port (usually made visible by LEDs on the FPGA board), minicom is probably using hardware flow-control to prevent data from being dropped. The Trireme Platform UART currently does not support hardware flow-control.

To turn off hardware flow control, press CTRL-A, then Z to open the minicom help menu shown in Figure 2.7. Hit O inside the help menu to open the mincom configuration menu shown in Figure 2.8. Alternatively, just hit CTRL-A, O to directly enter the minicom configuration menu.

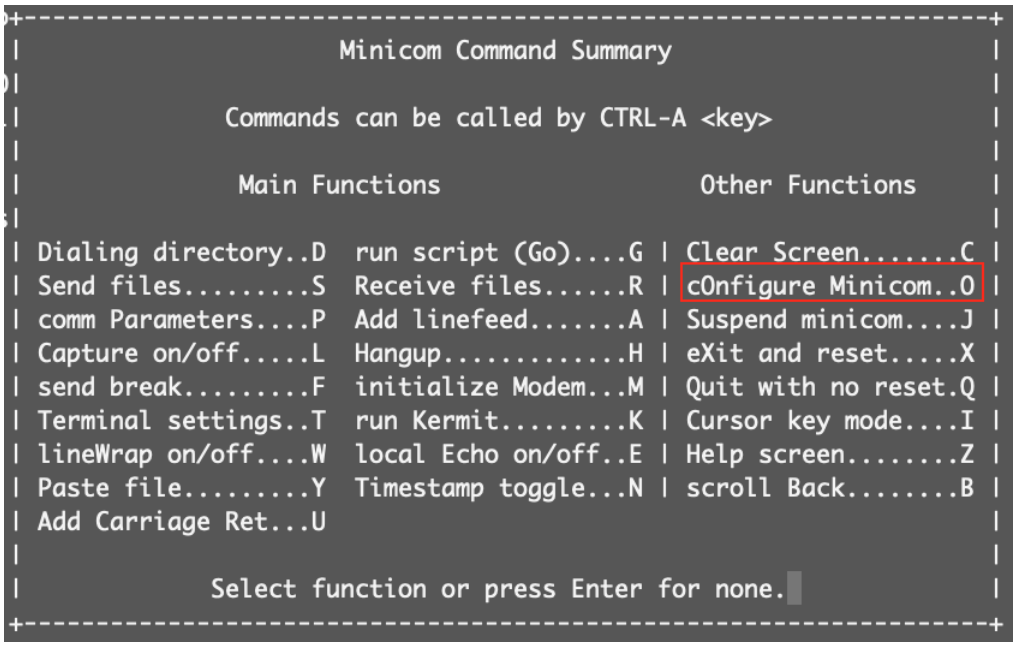

Figure 2.7: Hit CTRL-A, then Z to open the minicom help menu. Inside the help menu press O to open the minicom configuration menu.

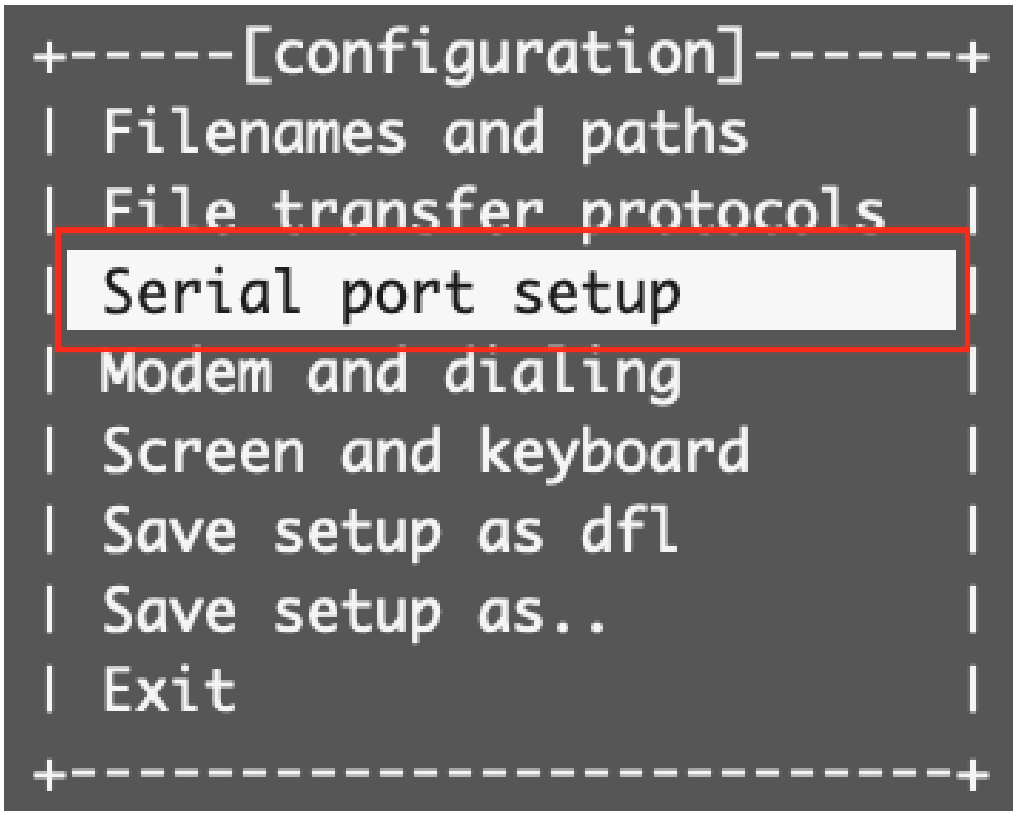

Figure 2.8: Select "Serial Port Setup" in the configuration menu.

In the configuration menu, open the serial port setup menu. Figure 2.9 shows the serial port setup menu. Inside the setup menu, hit F to toggle hardware flow control. Make sure flow-control is off then exit the menu and try uploading data again. Disabling flow-control may send all of the buffered data that was waiting to send while flow control was on. If so, reset the "Poor Processor" or "Many Poor Processor" FPGA implementation to restart the bootloader program.

Figure 2.9: Inside the minicom configuration menu, make sure hardware flow control is off. Hit F to toggle the setting.

To ensure hardware flow control is off by default, hit CTRL-A then O to open the minicom configuration menu again and select "Save setup as dfl", shown in Figure 2.10. Hit enter to save the setup at the default location and file name. The next time minicom is started, it will use the current setup (now without hardware flow control).

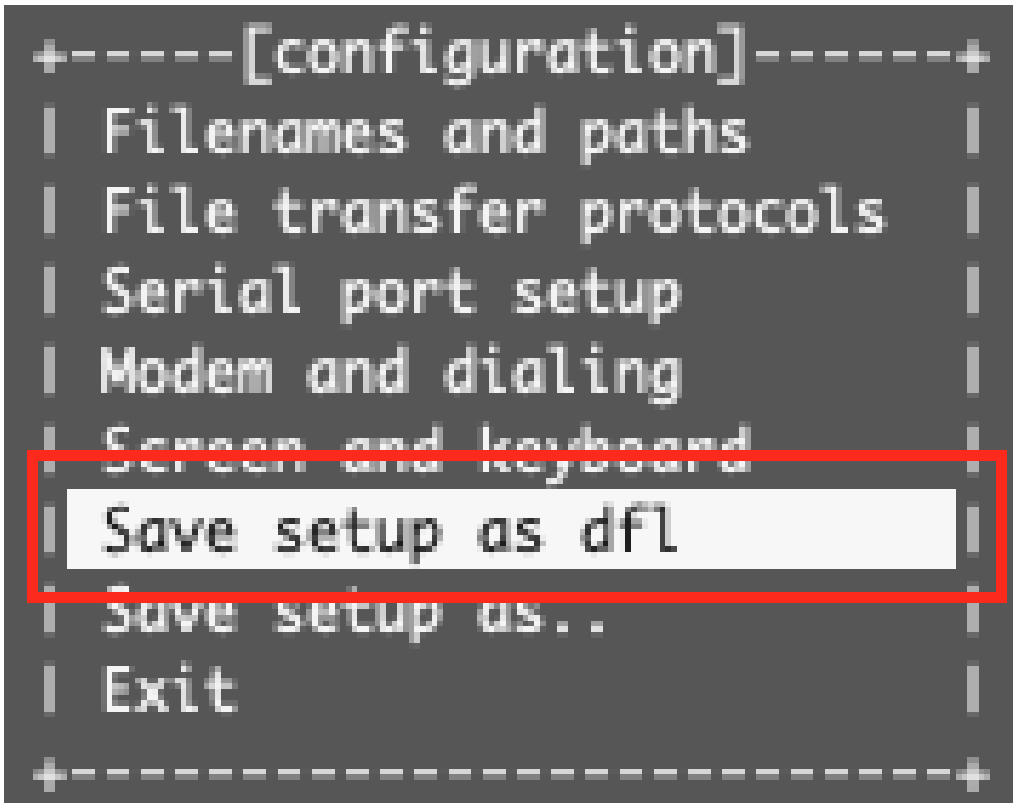

Figure 2.10: Inside the minicom configuration menu, make sure hardware flow control is off. Hit F to toggle the setting.

Note that the minicom menu will sometimes obscure text that is printed immediately after program loading. A workaround to ensure minicom properly displays all of the characters printed is to manually write the memory image to the UART by logging in as root and cat'ing the memory image to the UART device as shown in the following commands.

Listing 2.22: Write the memory image to the FPGA.

Chapter 3

Demonstration Systems

This chapter describes the architecture of the demonstration systems included with The Trireme Platform.

3.0.1 Poor Processor

The Poor Processor is a single-core processor with a NoC, distributed memory and a UART. The data memory interface of the seven stage RV32I core is connected directly to the NoC Node 0 without a cache hierarchy (it's cacheless). One port dual port BRAM is also connected to the NoC. The second BRAM port is connected directly to the core's instruction memory interface. All instructions must be on the Node 0 BRAM; otherwise the instruction memory interface will not be able to access them. The data memory interface can access any address in the system. Figure 3.1 shows a block diagram of the Poor Processor.

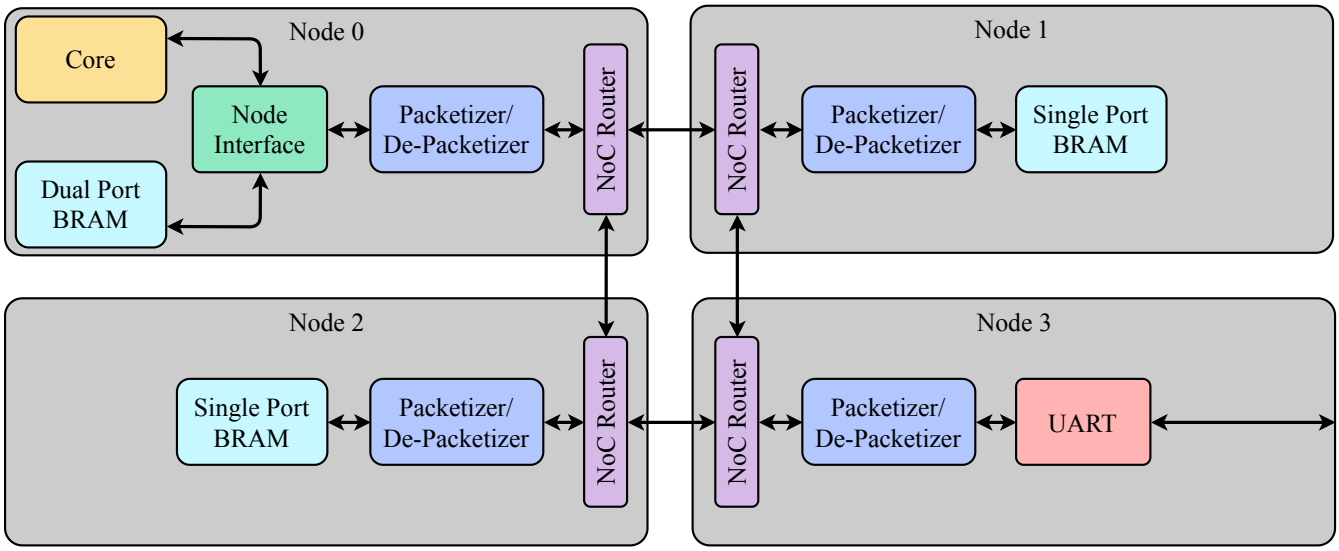

Figure 3.1: A block diagram of the Poor Processor architecture.

By default, each node in the Poor Processor has 4096 Bytes (12 address bits of memory). Address bits 13 and 14 are used to select the node at which a memory operation is performed. The first 768 bytes of memory (addresses 0x000-0x2FF) in Node 0 are reserved for the bootloader program. The bootloader program (bootloader.c) reads 3228 bytes of memory from the UART and writes them to addresses 0x0300-0x0FFF. The memory at Nodes 1 and 2 is uninitialized. An FPGA will probably initialize the remaining BRAM to zeros but if the system is reset without re-configuring the FPGA, data from the previous program execution will remain in the BRAM. Programs should not assume the contents of the memory are zero at the start of their execution. Figure 3.2. After the bootloader program has initialized instruction memory, it jumps to address 768 (0x300) to begin execution of the loaded program.

By default the UART Tx address is 0x3020 and the Rx address is 0x3010. Writing a byte to the Tx address will transmit it over the UART. There is no need to check the buffer status bits before transmission. Reading a half-word (2 bytes) from the Rx Address will read any data available in the Rx buffer (lower byte) and the status of the data

(upper byte). When the upper byte is 0x01, the data in the lower byte of the loaded data is valid. When the upper byte is 0x00, the data in the lower byte of the loaded data is invalid, i.e. the Rx buffer is empty.

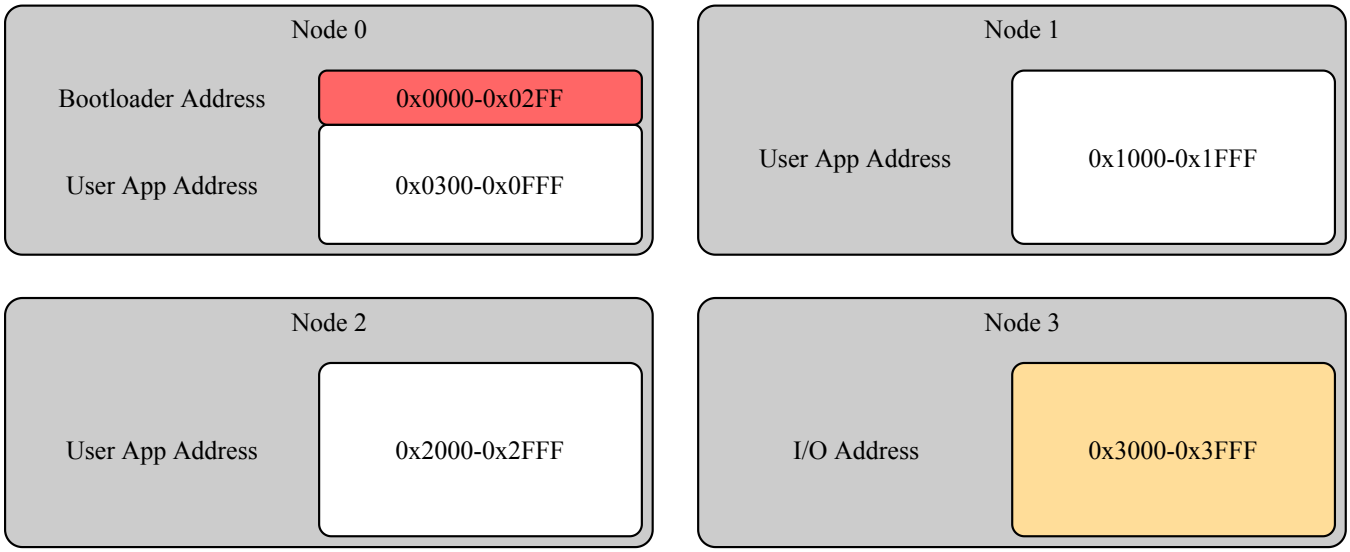

Figure 3.2: The address mapping for each node in the Poor Processor.

3.0.2 Many Poor Processor

The Many Poor Processor is similar to the Poor Processor except it has three poor_processor_nodes connected to the NoC instead of one poor_processor_node and two memory_nodes. A UART is included in node 3, just as in the Poor Processor. Figure 3.3 shows a block diagram of the architecture.

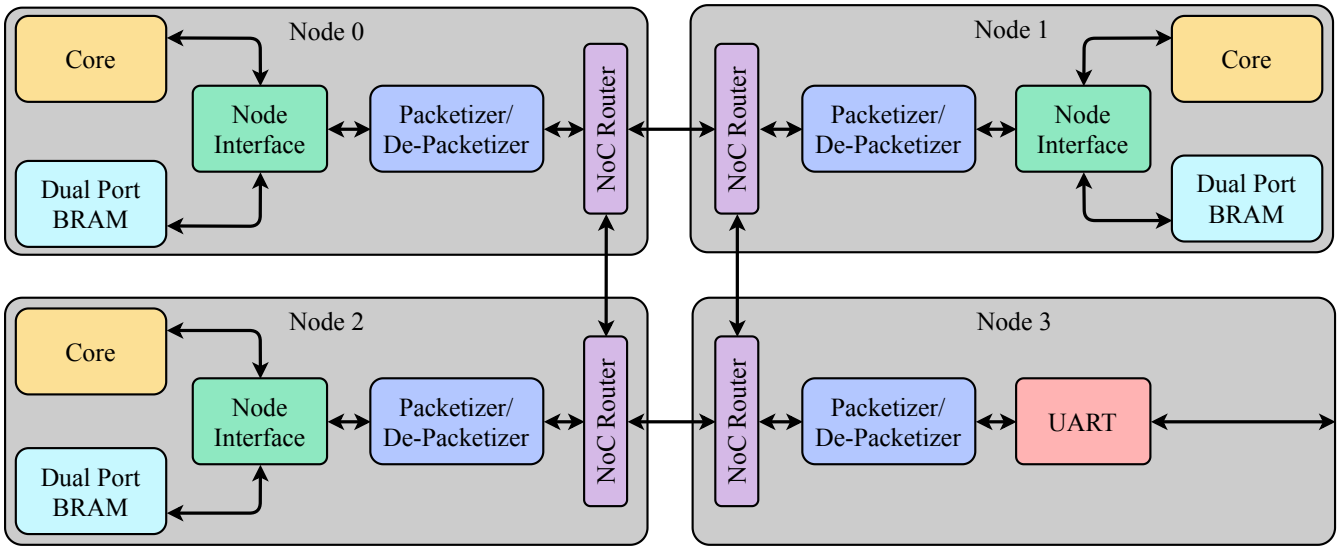

Figure 3.3: A block diagram of the Many Poor Processor architecture.

By default, the Many Poor Processor has 4096 Bytes of memory (12 address bits) at each poor_processor node. The UART node does not have any memory, only I/O mapped into the address space.preceding sentence is awkaward but not sure how to fix With 12 address bits worth of memory space at each node in the NoC, address bits 13 and 14 are used to select the node in the network the address will access. Figure 3.4 shows the address mapping to each node on the network. The first 768 bytes of memory (addresses 0x000-0x2FF) in each node are reserved for the bootloader program. The core 0 bootloader program (multi-core-bootloader.asm) reads 3328 bytes ($4096 - 768 = 3328$) of data from the UART for each poor_processor node in the system and writes the data to the user application memory (addresses 0x300-0xFFF at each node). When all of the data has been received from the UART and written to the

user application memory, core 0 unlocks the other cores by writing 0x00000000 to address 312 (0x138). Other cores read the lock value and loop until the lock is set to 0x00000000 by core 0. Once the lock value is set to 0x00000000, all of the cores jump to address 768 (0x300) to execute their local program. Each core's instruction memory interface can only access local addresses (addresses that are accessible without traversing the NoC), meaning core 0 will jump to the physical address 0x0300, core 1 will jump to the physical address 0x1300 and core 2 will jump to the physical address 0x2300. The UART addresses and behavior in the Many Poor Processor is identical to the Poor Processor.

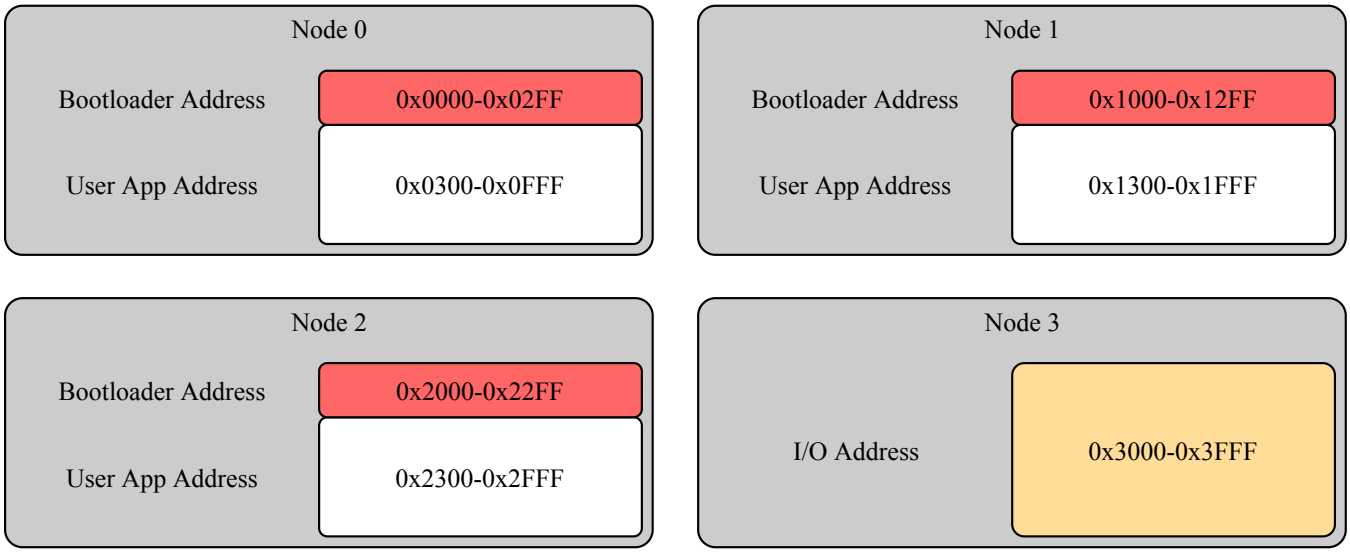

Figure 3.4: The address mapping for each node in the Many Poor Processor.

Chapter 4

Register Transfer Level Code

This chapter contains documentation for all of The Trireme Platform Verilog RTL code. Sections and subsections exist for each directory in The Trireme Platform rtl directory.

4.1 Common Modules

These modules are used in more than one of the core, memory, router, nodes, or top modules.

4.1.1 Pipeline Register

The pipeline_register module outputs a registered input. A stall input signal will hold the register output constant. A flush input signal will register and output a "flush" input instead. The flush signal takes priority if both stall and flush signals are high. The width of the registered signal is set with a parameter. Multiple signals can be concatenated together so only one pipeline_register module is needed per stage.

Table 4.1: Pipeline Register Parameters

Parameters	**Description**
PIPELINE_STAGE	Unused parameter. Meant to be a unique number to ID signals printed by scan logic.
PIPE_WIDTH	The width in bits of the input, flush and output signals.
SCAN_CYCLES_MIN	Unused parameter. Meant to bound when scan logic prints.
SCAN_CYCLES_MAX	Unused parameter. Meant to bound when scan logic prints.

Table 4.2: Pipeline Register Ports

Port	**Direction**	**Width**	**Description**
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	Synchronous, active high reset signal. Sets pipeline register to zeros.
stall	Input	1	Active high. Holds the output constant if flush is not asserted.
flush	Input	1	Active high. Registers and outputs the flush input data.
pipe_input	Input	PIPE_WIDTH	Data that is registered and output on the positive clock edge if stall and flush are low.
flush_input	Input	PIPE_WIDTH	Data that is registered and output on the positive clock edge if flush is high.

Table 4.2: Pipeline Register Ports

Port	Direction	Width	Description
pipe_output	Output	PIPE_WIDTH	The registered data.

4.1.2 Priority Encoder

The priority encoder module combinationally selects one request from a vector of one bit request signals (the decode port). The selected request's index is output on the encode port. The valid signal is raised whenever a request is made and the encode output is valid. The direction of the priority can be selected with the PRIORITY parameter. Valid priorities are: "LSB" - least significant bit is highest priority, and "MSB" - most significant bit is highest priority. The WIDTH parameter sets the number of request lines.

Table 4.3: Priority Encoder Parameters

Parameters	Description
WIDTH	The width in bits of the decode vector. Sets the number of requests in the priority encoder (one for each bit in the decode vector).
PRIORITY	The direction of the priority encoding. Valid options are "LSB" and "MSB" (with quotes).

Table 4.4: Priority Encoder Ports

Port	Direction	Width	Description
decode	Input	WIDTH	An active high vector of request signals.
encode	Output	log2(WIDTH)	The index of the selected decode request.
valid	Output	1	Active High. Indicates a request has been made and the encode signal is valid.

4.1.3 FIFO Queue

This module implements a First-In First-Out (FIFO) Queue. Power of two queue depths are supported. The input and output ports are synchronized to a single clock. The FIFO full signal can be raised early by setting the Q_IN_BUFFERS parameter greater than 0. The FIFO can operate in "show-ahead" mode by setting the peek input to 1. In "show-ahead" mode the next valid data is constantly output on the read_data port until the read enable signal indicates it has been read. An word can be read and written from the FIFO in the same cycle.

Table 4.5: FIFO Parameters

Parameters	Description
DATA_WIDTH	The bit width of the FIFO read and write ports.
Q_DEPTH_BITS	The number of address bits for the FIFO memory. The FIFO depth is $2^{Q\_DEPTH\_BITS}$
Q_IN_BUFFERS	The number of empty spots in the FIFO that trigger the full flag to be raised. The full signal is raised whenever the number of words in the FIFO is greater than or equal to $(2^{Q\_DEPTH\_BITS}) - (Q\_IN\_BUFFERS)$

Table 4.6: FIFO Ports

Port	Direction	Width	Description
clk	Input	1	The clock for all synchronous logic and the read and write ports.
reset	Input	1	Synchronous, active high reset signal. Empties the FIFO.
write_data	Input	DATA_WIDTH	Input for new data.
wrtEn	Input	1	Active high write enable. Add write_data to the queue if it is not full
rdEn	Input	1	Active high read enable. If the queue is not empty, remove a word from the queue and output it. Data will be valid the same cycle.
peek	Input	1	Active high signal. Read the next valid word from the queue in the same cycle. The word is not removed from the queue.
read_data	Output	DATA_WIDTH	The data output from the queue. Valid if valid signal is high.
valid	Output	1	Active high. Indicates that read_data is valid.
full	Output	1	Active high. Indicates there are more than $(2^{Q\_DEPTH\_BITS}) - (Q\_IN\_BUFFERS)$ words in the queue.
empty	Output	1	Active high. Indicates the queue is empty. Will go high the same cycle the last word is read from the queue.

4.1.4 Arbiter

TODO: write this section after the arbiter is merged into the master branch.

4.2 Core Designs

The different types of RISC-V cores included in The Trireme Platform are described here.

4.2.1 Base Core Modules

These modules are common to more than one type of core.

4.2.1.1 Fetch Issue

The fetch issue module requests instructions from the instruction memory based on the current Program Counter (PC) value. The PC is stored in a register inside the fetch issue module. The next_PC_select input from a control unit chooses the next PC value. The PC value is output on the issue_PC and i_mem_read_address ports. The issue_PC port sends the expected PC to the fetch receive module. The i_mem_read_address port connects to the address bus of the instruction memory. Separate issue_PC and i_mem_read_address ports are used because previous versions of the memory interface used word addresses instead of byte addresses, making the outputs different. The outputs were not merged for backwards compatibility.

Table 4.7: Fetch Issue Parameters

Parameters	Description
CORE	A unique number to identify the core this module is in.
RESET_PC	The PC value set when a reset is triggered.
ADDRESS_BITS	The number of bits in the address. Sets the width of the PC register.
SCAN_CYCLES_MIN	Unused parameter. Meant to bound when scan logic prints.

Table 4.7: Fetch Issue Parameters

Parameters	Description
SCAN_CYCLES_MAX	Unused parameter. Meant to bound when scan logic prints.

Table 4.8: Fetch Issue Ports

Port	Direction	Width	Description
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	Synchronous, active high reset signal. Sets the PC register to RESET_PC.
next_PC_select	Input	2	The control signal to select the next PC value.
target_PC	Input	ADDRESS_BITS	The target address of branches and jumps. Set as the next PC when next_PC_select is 2'b10.
issue_PC	Output	ADDRESS_BITS	The current PC register value. This port connects to the fetch receive module.
i_mem_read_address	Output	ADDRESS_BITS	The current PC register value. This port connects to the instruction memory.

4.2.1.2 Fetch Receive

The fetch receive module takes in the data returned by the instruction memory and outputs it to the decode stage. It appears that the flush feature (currently the only feature in the module) is made redundant by the flush of the pipeline stage but this has not been tested.

Future versions of this module will need to select the appropriate part of the i_mem_data to output as an instruction. Both the 64-bit or compressed ISAs will have instructions that are a different size than the DATA_WIDTH parameter that sets the i_mem_data width.

Table 4.9: Fetch Receive Parameters

Parameters	Description
DATA_WIDTH	The number of bits in the core data path. Except for highly customized and experimental designs, this will be equivalent to the RISC-V XLEN parameter. Expected values are 32 or 64.
SCAN_CYCLES_MIN	Unused parameter. Meant to bound when scan logic prints.
SCAN_CYCLES_MAX	Unused parameter. Meant to bound when scan logic prints.

Table 4.10: Fetch Receive Ports

Port	Direction	Width	Description
flush	Input	1	The flush signal for the fetch receive pipeline stage. This may be made redundant by the flush feature of the pipeline register module. When high, the instruction output is set as a NOP.
i_mem_data	Input	DATA_WIDTH	The data returned by the instruction memory.
instruction	Output	DATA_WIDTH	The instruction to be sent to the decode stage.

4.2.1.3 Decode Unit

The decode unit combinationally decodes an input instruction to output the appropriate XLEN extended immediate field and register file data for use in the execute stage. Instruction opcode, funct3, funct7 fields are also output for use in the control unit. Branch and jump targets are computed by adding the appropriate immediate field to the PC input. The register file is instantiated inside the decode unit. The register file write port is exposed as inputs for connection to the writeback stage.

Table 4.11: Decode Unit Parameters

Parameters	**Description**
CORE	A unique number to identify the core this module is in.
ADDRESS_BITS	The number of bits in the address. Sets the width of the PC, branch_target and JAL_target ports.
SCAN_CYCLES_MIN	Prevents scan logic from printing for cycles less than this threshold.
SCAN_CYCLES_MAX	Prevents scan logic from printing for cycles greater than this threshold.

Table 4.12: Decode Unit Ports

Port	**Direction**	**Width**	**Description**
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	Synchronous, active high reset signal. Sets x0 in the register file to zero and resets scan logic.
PC	Input	ADDRESS_BITS	The PC value for the input instruction.
instruction	Input	32	The input instruction to decode.
extend_sel	Input	2	Control bits to select between different immediate encodings for the extend_imm output.
write	Input	1	Active high register file write enable.
write_reg	Input	5	The register file address to write to.
write_data	Input	32	The data from the writeback stage to be written to the regiter file. The register file address is set by the write_reg input port.
rs1_data	Output	32	The register file data stored in the address selected by the instruction's rs1 bits.
rs2_data	Output	32	The register file data stored in the address selected by the instruction's rs2 bits.
rd	Output	5	The instruction input rd bits (instruction bits 7-11).
opcode	Output	7	The instruction input opcode bits (instruction bits 0-6).
funct7	Output	7	The instruction input funct7 bits (instruction bits 25-31).
funct3	Output	3	The instruction input funct3 bits (instruction bits 12-14).
extend_imm	Output	32	The XLEN extended immediate field of the instruction input.
branch_target	Output	ADDRESS_BITS	A branch target address. Generated by summing the decoded B-type immediate field with the PC input.
JAL_target	Output	ADDRESS_BITS	A Jump And Link target address. Generated by summing the J-type immediate field wit the PC input.

Table 4.13: Decode Unit Sub-Modules

Sub-Module	Description
regFile	The RISC-V register file with 32 XLEN-bit words.

4.2.1.4 Control Unit

The control unit combinationally generates control signals for the whole processor. This control unit is intended to act as a base module that can be wrapped by other combinational logic. Pipelined processors should wrap this module, the hazard detection module and a custom stall module into a core-specific control unit. Note that the control signal generation is not optimized for area. Instead, it has been designed to be easily wrapped and extended with additional logic. Many control signals are generated by comparing the opcode, funct3 and funct7 fields of the instruction when a single bit of the instruction could be checked instead. Such an implementation is larger in both area and lines of code but also makes clear which instructions should set a signal high or low.

Table 4.14: Control Unit Parameters

Parameters	Description
CORE	A unique number to identify the core this module is in.
ADDRESS_BITS	The number of bits in the address. Sets the width of the PC, branch_target and JAL_target ports.
NUM_BYTES	The number of bytes in the DATA_WIDTH parameter. DATA_WIDTH is set in the module that instantiates this control unit. This parameter is equivalent to DATA_WIDTH/8.
LOG2_NUM_BYTES	The log2 of NUM_BYTES. The value is computed based on the set NUM_BYTES value. This parameter does not need to be set by the instantiating module. It is always safe to use the default value.
SCAN_CYCLES_MIN	Prevents scan logic from printing for cycles less than this threshold.
SCAN_CYCLES_MAX	Prevents scan logic from printing for cycles greater than this threshold.

Table 4.15: Control Unit Ports

Port	Direction	Width	Description
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	Synchronous, active high reset signal. Sets x0 in the register file to zero and resets scan logic.
opcode_decode	Input	7	The decode stage's current instruction opcode field.
opcode_execute	Input	7	The execute stage's current instruction opcode field.
funct3	Input	3	The decode stage's current instruction funct3 field.
funct7	Input	7	The decode stage's current instruction funct7 field.
JALR_target_execute	Input	ADDRESS_BITS	The Jump And Link Register target address (where a JALR instruction would jump to) of the instruction in the execute stage.

Table 4.15: Control Unit Ports

Port	Direction	Width	Description
branch_target_execute	Input	ADDRESS_BITS	The branch target address (where a taken branch of a branch instruction would jump to) of the instruction in the execute stage.
JAL_target_decode	Input	ADDRESS_BITS	The Jump And Link target address (where a JAL instruction would jump to) of the instruction in the decode stage.
branch_execute	Input	1	A control signal to indicate the instruction in the execute stage is branching. On a "branch taken" the signal will be 1. Otherwise it is 0.
true_data_hazard	Input	1	A hazard signal to indicate there is a data hazard in the processor pipeline (if pipelining is implemented) that cannot be solved with data forwarding. When forwarding is implemented, true data hazards should only happen when an instruction needs a value loaded by a load instruction that has not reached the writeback stage yet.
d_mem_issue_hazard	Input	1	A hazard signal to indicate the data memory interface is not ready to receive an operation.
d_mem_recv_hazard	Input	1	A hazard signal to indicate the memory receive stage is expecting load data but the data memory interface has not returned it yet.
i_mem_hazard	Input	1	A hazard signal to indicate the instruction memory interface is either not ready for new data or has not returned the requested data.
JALR_branch_hazard	Input	1	A hazard signal to indicate a JALR or branch instruction has not computed the target PC in the execute stage yet.
JAL_branch_hazard	Input	1	A hazard signal to indicate a JAL instruction has not computed the target PC in the decode stage yet.
branch_op	Output	1	A control signal for the execute stage to indicate the current instruction is a branch instruction.
memRead	Output	1	A control signal for the data memory interface to indicate that a memory read (load) should be performed. Set to 1 for load instructions.
ALU_operation	Output	6	A control signal to select the ALU operation performed in the execute stage. Only 4 bits are needed for the base instruction set. Additional bits are included to support ISA extensions and custom instructions.
memWrite	Output	1	A control signal for the data memory interface to indicate a memory write (store) should be performed. Set to 1 for store instructions.

Table 4.15: Control Unit Ports

Port	Direction	Width	Description
log2_bytes	Output	LOG2_NUM_BYTES	A control signal for the data memory interface to indicate the number of bytes in a load or store operation. Set to 2 for 4-byte word operations. Set to 1 for 2-byte half-word operations. Set to 0 for byte operations.
unsigned_load	Output	1	A control signal used to sign extend loaded data values when high. Set to 1 for signed load operations.
next_PC_sel	Output	2	A control signal used to select the next PC value in the fetch issue stage. Selects between PC+4 (2'b00), PC (2'b01), and target PC (2'b10).
operand_A_sel	Output	2	A control signal used to select the value input into the operand A port of the ALU. A value of 2'b00 selects the data read from the register file. A 2'b01 value selects the PC of the instruction entering the execution unit. A 2'b10 value selects the execute instruction's PC+4. A 2'b11 value selects 0 (for LUIs).
operand_B_sel	Output	1	A control signal used to select between the rs2 data (0) and instruction immediate field (1).
extend_sel	Output	1	A control signal used to select between different instruction immediate field encodings in the decode unit. Setting 2'b00 (or 2'b11) selects the I-type immediate. Setting 2'b01 selects the S-type immediate. Setting 2'b10 selects the U-type immediate.
regWrite	Output	1	A control signal used to enable register file writes. Set to 1 for all non-store instructions.
target_PC	Output	ADDRESS_BITS	The target PC send to the fetch issue stage.
i_mem_read	Output	1	The instruction memory interface read enable signal. Currently it is always set to 1.
scan	Input	1	An active high signal to print debug information in simulation.

4.2.1.5 Hazard Detection Unit

The hazard detection unit takes in control signals and outputs hazard signals that can be decoded to set pipeline stall or flush signals. This module is not needed in the single cycle core. Only the five and seven stage pipeline cores use it.

Table 4.16: Hazard Detection Unit Parameters

Parameters	Description
CORE	A unique number to identify the core this module is in.
ADDRESS_BITS	The number of bits in the address. Sets the width of the PC register and address buses.

Table 4.16: Hazard Detection Unit Parameters

Parameters	**Description**
SCAN_CYCLES_MIN	Prevents scan logic from printing for cycles less than this threshold.
SCAN_CYCLES_MAX	Prevents scan logic from printing for cycles greater than this threshold.

Table 4.17: Hazard Detection Unit Ports

Port	**Direction**	**Width**	**Description**
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	Synchronous, active high reset signal. Resets scan logic.
fetch_valid	Input	1	The fetch/instruction memory interface data valid signal. A high value indicates the data returned by the instruction memory is valid.
fetch_ready	Input	1	The fetch/instruction memory interface ready signal. When this signal is high, the instruction memory interface can accept new requests.
issue_request	Input	1	The read signal of the fetch/instruction memory interface. When this signal is high, the core is requesting to read the next instruction. Currently, this signal is always high.
issue_PC	Input	ADDRESS_BITS	The address of the instruction requested by the core.
fetch_address_in	Input	ADDRESS_BITS	The address of the data returned by the instruction memory interface.
memory_valid	Input	1	The data memory interface's data valid signal. A high value indicates the data returned by the instruction memory is valid.
memory_ready	Input	1	The data memory interface ready signal. When this signal is high, the data memory interface can accept new requests.
load_address	Input	ADDRESS_BITS	The address sent to the data memory for a load operation.
memory_address_in	Input	ADDRESS_BITS	The address of the data returned by the data memory interface.
opcode_decode	Input	7	The opcode field of the instruction in the decode stage.
opcode_execute	Input	7	The opcode field of the instruction in the execute stage.
branch_execute	Input	1	The branch taken signal from the execute stage.
i_mem_hazard	Output	1	An active high signal to indicate an instruction memory hazard. Either the instruction memory was not ready or the requested read operation did not return valid data.
d_mem_hazard_issue	Output	1	An active high signal to indicate the data memory interface is not ready to receive new requests.

Table 4.17: Hazard Detection Unit Ports

Port	Direction	Width	Description
d_mem_hazard_receive	Output	1	An active high signal to indicate the data memory interface has not returned the requested read/load data.
JALR_branch_hazard	Output	1	An active high signal to indicate that a branch was taken or a JALR instruction was executed.
JAL_hazard	Output	1	An active high signal to indicate that a JAL instruction was executed.

4.2.1.6 Execute Unit

The execute unit wraps the RV32i ALU with logic to select ALU operands and generate JALR target addresses. The operand_A_sel signals controls a mux to select between rs1 data, 0 (LUI), PC (AUIPC) or PC+4 (JAL, JALR) as the first operand. A zero value is fed into the ALU for LUI instructions so an add add operation can be performed instead of including a dedicated "operand B pass-through" ALU operation. The operand_B_sel mux selects between the data read from the rs2 port of the register file and the instruction's extended immediate field. The extended immediate field is selected for store operations so the immediate field can be added to the rs1 data. The rs2 data is passed around the ALU to the memory issue stage to send store data to the memory hierarchy.

Table 4.18: Execute Unit Parameters

Parameters	Description
CORE	A unique number to identify the core this module is in.
DATA_WIDTH	The number of bits in the core data path. Except for highly customized and experimental designs, this will be equivalent to the RISC-V XLEN parameter. Expected values are 32 or 64.
ADDRESS_BITS	The number of bits in the address. Sets the width of the PC, branch_target and JAL_target ports.
SCAN_CYCLES_MIN	Prevents scan logic from printing for cycles less than this threshold.
SCAN_CYCLES_MAX	Prevents scan logic from printing for cycles greater than this threshold.

Table 4.19: Execute Unit Ports

Port	Direction	Width	Description
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	Synchronous, active high reset signal. Resets scan logic.
ALU_operation	Input	6	The ALU's mux control signal. Select the appropriate ALU operation to output. Only 5 bits are needed for the RV32i ISA. A 6th bit has been included to support additional instruction operations.
PC	Input	ADDRESS_BITS	The PC value of the instruction currently in the execute stage of the core.
operand_A_sel_execute	Input	2	A control signal to select the first operand fed into the ALU.

Table 4.19: Execute Unit Ports

Port	Direction	Width	Description
operand_B_sel_execute	Input	1	A control signal to select the second operand fed into the ALU. A '1' selects the instruction's extended immediate field. A '0' selects the instruction's rs2 data.
branch_op	Input	1	Active high control signal to indicate the current instruction is a branch instruction.
rs1_data	Input	DATA_WIDTH	Data read from the rs1 port of the register file in the decode stage.
rs2_data	Input	DATA_WIDTH	Data read from the rs2 port of the register file in the decode stage.
extend	Input	DATA_WIDTH	The current instruction's extended immediate field.
branch	Output	1	An active high control signal to indicate a branch taken result for the current instruction.
ALU_result	Output	DATA_WIDTH	The value computed by the ALU based on the selected operation and operands inputs.
JALR_target	Output	ADDRESS_BITS	The computed address target for JALR instructions.
scan	Input	1	An active high signal to print debug information in simulation.

Table 4.20: Execute Unit Sub-Modules

Sub-Module	Description
ALU	The RV32i Arithmetic Logic Unit.

4.2.1.7 ALU

The ALU module implements an RV32i Arithmetic Logic Unit. The appropriate operands are selected in the execute_unit module and fed into the ALU. The ALU_operation signal selects from a mux the appropriate computation result. The ALU is entirely combinational. The output is registered (potentially after more muxing between the execute and memory issue stages.

Table 4.21: ALU Parameters

Parameters	Description
DATA_WIDTH	The number of bits in the core data path. Except for highly customized and experimental designs, this will be equivalent to the RISC-V XLEN parameter. Expected values are 32 or 64.

Table 4.22: ALU Ports

Port	Direction	Width	Description
ALU_operation	Input	6	The ALU's mux control signal. Select the appropriate operation to output. Only 5 bits are needed for the RV32i ISA. A 6th bit has been included to support additional instruction operations.

Table 4.22: ALU Ports

Port	Direction	Width	Description
operand_A	Input	DATA_WIDTH	The first operand for ALU operations. The rs1 data is fed into this port (for instructions that use rs1 data).
operand_B	Input	DATA_WIDTH	The second operand for ALU operations. The rs2 data or XLEN extended immediate field is fed into this port.
ALU_result	Output	DATA_WIDTH	The value computed based on the selected operation and operands inputs.

4.2.1.8 Memory Issue Unit

The memory issue module takes in data and addresses output from the execution unit and issues read and write operations on the data memory interface. Byte enable signals are included to enable/disable writes to individual bytes in a word of memory. Internal logic shifts the data from execute unit to align byte and half-word stores with a potentially non-word aligned address. Load and store operations must be aligned to addresses that are multiples of the given data size. In other words, bytes can be written to any address, half words can be written to even (every other) address and words can be written to every fourth address. The memory receive module similarly handles shifting of read data. Debug logic is included in this module to print warnings when unsupported (unaligned) read or write operations are issued.

Table 4.23: Memory Issue Unit Parameters

Parameters	Description
CORE	A unique number to identify the core this module is in.
DATA_WIDTH	The number of bits in the core data path. Except for highly customized and experimental designs, this will be equivalent to the RISC-V XLEN parameter. Expected values are 32 or 64.
ADDRESS_BITS	The number of bits in the address. Sets the width of the interface's address ports.
NUM_BYTES	The number of bytes in a data word. This sets the number of byte enable signals.
LOG2_NUM_BYTES	The log base2 of the number of bytes in a data word. The value is set based on NUM_BYTES and should not be overridden by the instantiating module.
SCAN_CYCLES_MIN	Prevents scan logic from printing for cycles less than this threshold.
SCAN_CYCLES_MAX	Prevents scan logic from printing for cycles greater than this threshold.

Table 4.24: Memory Issue Unit Ports

Port	Direction	Width	Description
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	Synchronous, active high reset signal. Resets scan logic.
load	Input	1	The active high control signal to indicate a memory read should be performed.
store	Input	1	The active high control signal to indicate a memory write should be performed.
address	Input	ADDRESS_BITS	The address input from the execute stage to send out the data memory interface.

Table 4.24: Memory Issue Unit Ports

Port	Direction	Width	Description
store_data	Input	DATA_WIDTH	The data input from the execute stage to send out the memory interface on store operations.
log2_bytes	Input	LOG2_NUM_BYTES	The log2 of the number of bytes to load or store.
memory_read	Output	1	The read enable signal of the data memory interface.
memory_write	Output	1	The write enable signal of the data memory interface.
memory_byte_en	Output	NUM_BYTES	The per-byte write enable signals. Read operations ignore this signal.
memory_address	Output	ADDRESS_BITS	The data memory interface address bus.
memory_data	Output	DATA_WIDTH	The data memory interface write data bus.

4.2.1.9 Memory Receive Unit

The memory receive module takes in read data words from the data memory interface and outputs it to the writeback unit. Internal logic shifts the loaded data word to align byte and half-word loads with a potentially non-word aligned address. Load operations must be aligned to addresses that are multiples of the given data size. In other words, bytes can be read from any address, half words can be read from even (every other) address and words can be read from every fourth address.

Table 4.25: Memory Receive Unit Parameters

Parameters	Description
CORE	A unique number to identify the core this module is in.
DATA_WIDTH	The number of bits in the core data path. Except for highly customized and experimental designs, this will be equivalent to the RISC-V XLEN parameter. Expected values are 32 or 64.
ADDRESS_BITS	The number of bits in the address. Sets the width of the interface's address ports.
NUM_BYTES	The number of bytes in a data word. This sets the number of byte enable signals.
LOG2_NUM_BYTES	The log base2 of the number of bytes in a data word. The value is set based on NUM_BYTES and should not be overridden by the instantiating module.
SCAN_CYCLES_MIN	Prevents scan logic from printing for cycles less than this threshold.
SCAN_CYCLES_MAX	Prevents scan logic from printing for cycles greater than this threshold.

Table 4.26: Memory Receive Unit Ports

Port	Direction	Width	Description
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	Synchronous, active high reset signal. Resets scan logic.
log2_bytes	Input	LOG2_NUM_BYTES	The log2 of the number of bytes to load or store.
unsigned_load	Input	1	A control signal to indicate is the loaded data should be sign extended. When high, data is not sign extended.

Table 4.26: Memory Receive Unit Ports

Port	Direction	Width	Description
memory_data_in	Input	DATA_WIDTH	The data memory interface read data bus.
memory_address_in	Input	ADDRESS_BITS	The address associated with the memory_data_in value.
load_data	Output	DATA_WIDTH	The loaded and shifted data to be sent to the write-back unit.

4.2.1.10 Writeback Unit

The writeback unit selects the between the ALU result and data loaded from the data memory interface. The selected data is sent to the register file for writing. Register file write enable and write address signals are passed through the writeback module without modification.

Table 4.27: Writeback Unit Parameters

Parameters	Description
CORE	A unique number to identify the core this module is in.
DATA_WIDTH	The number of bits in the core data path. Except for highly customized and experimental designs, this will be equivalent to the RISC-V XLEN parameter. Expected values are 32 or 64.
SCAN_CYCLES_MIN	Prevents scan logic from printing for cycles less than this threshold.
SCAN_CYCLES_MAX	Prevents scan logic from printing for cycles greater than this threshold.

Table 4.28: Writeback Unit Ports

Port	Direction	Width	Description
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	Synchronous, active high reset signal. Resets scan logic.
ALU_operation	Input	6	The ALU's mux control signal. Select the appropriate ALU operation to output. Only 5 bits are needed for the RV32i ISA. A 6th bit has been included to support additional instruction operations.
opWrite	Input	1	The register file write enable input.
opSel	Input	1	A mux control signal to select ALU_result or memory_data as the register file write data.
opReg	Input	5	The register address to write to.
ALU_result	Input	DATA_WIDTH	The data output by the ALU.
memory_data	Input	DATA_WIDTH	The data returned by the data memory interface for load operations.
write	Output	1	The register file write enable output signal.
write_reg	Output	5	The register file write address output signal.
write_data	Output	DATA_WIDTH	The register file write data output signal.

4.2.2 Single Cycle Core

These are the modules specific to the single cycle in-order core.

4.2.2.1 Single Cycle Control Unit

The single cycle control unit combinatorially generates signals for the whole processor. It includes an instantiation of the base core control module and a hazard detection unit. Wrapping shared base madules with core specific logic maximizes code re-use and modularity in the Trireme Platform.

Table 4.29: Single Cycle Control Unit Parameters

Parameters	**Description**
CORE	A unique number to identify the core this module is in.
ADDRESS_BITS	The number of bits in the address. Sets the width of the PC, branch_target and JAL_target ports.
NUM_BYTES	The number of bytes in the DATA_WIDTH parameter. (DATA_WIDTH is set in the module that instantiates this control unit. This parameter is equivalent to DATA_WIDTH/8.
LOG2_NUM_BYTES	The log2 of NUM_BYTES. The value is computed based on the set NUM_BYTES value. This parameter does not need to be set by the instantiating module. It is always safe to use the default value.
SCAN_CYCLES_MIN	Prevents scan logic from printing for cycles less than this threshold.
SCAN_CYCLES_MAX	Prevents scan logic from printing for cycles greater than this threshold.

Table 4.30: Single Cycle Control Unit Ports

Port	**Direction**	**Width**	**Description**
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	Synchronous, active high reset signal. Sets x0 in the register file to zero and resets scan logic.
opcode_decode	Input	7	The decode stage's current instruction opcode field.
opcode_execute	Input	7	The execute stage's current instruction opcode field.
funct3	Input	3	The decode stage's current instruction funct3 field.
funct7	Input	7	The decode stage's current instruction funct7 field.
JALR_target_execute	Input	ADDRESS_BITS	The Jump And Link Register target address (where a JALR instruction would jump to) of the instruction in the execute stage.
branch_target_execute	Input	ADDRESS_BITS	The branch target address (where a taken branch of a branch instruction would jump to) of the instruction in the execute stage.
JAL_target_decode	Input	ADDRESS_BITS	The Jump And Link target address (where a JAL instruction would jump to) of the instruction in the decode stage.

Table 4.30: Single Cycle Control Unit Ports

Port	**Direction**	**Width**	**Description**
branch_execute	Input	1	A control signal to indicate the instruction in the execute stage is branching. On a "branch taken" the signal will be 1. Otherwise it is 0.
branch_op	Output	1	A control signal for the execute stage to indicate the current instruction is a branch instruction.
memRead	Output	1	A control signal for the data memory interface to indicate that a memory read (load) should be performed. Set to 1 for load instructions.
ALU_operation	Output	6	A control signal to select the ALU operation performed in the execute stage. Only 4 bits are needed for the base instruction set. Additional bits are included to support ISA extensions and custom instructions.
memWrite	Output	1	A control signal for the data memory interface to indicate a memory write (store) should be performed. Set to 1 for store instructions.
log2_bytes	Output	LOG2_NUM_BYTES	A control signal for the data memory interface to indicate the number of bytes in a load or store operation. Set to 2 for 4-byte word operations. Set to 1 for 2-byte half-word operations. Set to 0 for byte operations.
unsigned_load	Output	1	A control signal used to sign extend loaded data values when high. Set to 1 for signed load operations.
next_PC_sel	Output	2	A control signal used to select the next PC value in the fetch issue stage. Selects between PC+4 (2'b00), PC (2'b01), and target PC (2'b10).
operand_A_sel	Output	2	A control signal used to select the value input into the operand A port of the ALU. A value of 2'b00 selects the data read from the register file. A 2'b01 value selects the PC of the instruction entering the execution unit. A 2'b10 value selects the execute instruction's PC+4. A 2'b11 value selects 0 (for LUIs).
operand_B_sel	Output	1	A control signal used to select between the rs2 data (0) and instruction immediate field (1).
extend_sel	Output	1	A control signal used to select between different instruction immediate field encodings in the decode unit. Setting 2'b00 (or 2'b11) selects the I-type immediate. Setting 2'b01 selects the S-type immediate. Setting 2'b10 selects the U-type immediate.
regWrite	Output	1	A control signal used to enable register file writes. Set to 1 for non-store instructions when no data hazard is present.
target_PC	Output	ADDRESS_BITS	The target PC send to the fetch issue stage.

Table 4.30: Single Cycle Control Unit Ports

Port	Direction	Width	Description
i_mem_read	Output	1	The instruction memory interface read enable signal. Currently it is always set to 1.
fetch_valid	Input	1	The fetch/instruction memory interface data valid signal. A high value indicates the data returned by the instruction memory is valid.
fetch_ready	Input	1	The fetch/instruction memory interface ready signal. When this signal is high, the instruction memory interface can accept new requests.
issue_request	Input	1	The read signal of the fetch/instruction memory interface. When this signal is high, the core is requesting to read the next instruction. Currently, this signal is always high.
issue_PC	Input	ADDRESS_BITS	The address of the instruction requested by the core.
fetch_address_in	Input	ADDRESS_BITS	The address of the data returned by the instruction memory interface.
memory_valid	Input	1	The data memory interface's data valid signal. A high value indicates the data returned by the instruction memory is valid.
memory_ready	Input	1	The data memory interface ready signal. When this signal is high, the data memory interface can accept new requests.
load_address	Input	ADDRESS_BITS	The address sent to the data memory for a load operation.
memory_address_in	Input	ADDRESS_BITS	The address of the data returned by the data memory interface.
flush_fetch_receive	Output	1	An active high control signal to flush the fetch receive stage during instruction memory hazards.
scan	Input	1	An active high signal to print debug information in simulation.

Table 4.31: Single Cycle Control Unit Sub-Modules

Sub-Module	Description
hazard_detection_unit	The Hazard Detection Unit takes in control signals and outputs hazard signals that can be decoded to set pipeline stall or flush signals. It is not used in the single cycle core.
control_unit	The base control unit to generate control signals common to each in-order core.

4.2.2.2 Single Cycle Core

The single cycle core combines the base **fetch_issue**, **fetch_receive**, **decode**, **execute**, **memory_issue**, **memory_receive**, and **writeback** modules into simple in-order RV32I core. A dedicated control module is used to generate control signals for each submodule in the core.

Table 4.32: Single Cycle Core Parameters

Parameters	**Description**
CORE	A unique number to identify this core.
RESET_PC	The value the PC register in this core is set to when the reset signal is asserted.
DATA_WIDTH	The number of bits in the core data path. Except for highly customized and experimental designs, this will be equivalent to the RISC-V XLEN parameter. Expected values are 32 or 64.
ADDRESS_BITS	The number of bits in the address. Sets the width of the PC, branch_target and JAL_target ports.
NUM_BYTES	The number of bytes in the DATA_WIDTH parameter. (DATA_WIDTH is set in the module that instantiates this control unit. This parameter is equivalent to DATA_WIDTH/8.
SCAN_CYCLES_MIN	Prevents scan logic from printing for cycles less than this threshold.
SCAN_CYCLES_MAX	Prevents scan logic from printing for cycles greater than this threshold.

Table 4.33: Single Cycle Core Ports

Port	**Direction**	**Width**	**Description**
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	Synchronous, active high reset signal. Sets x0 in the register file to zero and resets scan logic.
start	Input	1	Unused input port. Formerly used to set PC to an arbitrary value. Kept to maintain backwards compatibility. Likely to be removed in future versions.
program_address	Input	ADDRESS_BITS	Unused input port. Formerly used to set PC to arbitrary value when start signal is asserted. Kept to maintain backwards compatibility. Likely to be removed in future versions.
fetch_valid	Input	1	The fetch/instruction memory interface data valid signal. A high value indicates the data returned by the instruction memory is valid.
fetch_ready	Input	1	The fetch/instruction memory interface ready signal. When this signal is high, the instruction memory interface can accept new requests.
fetch_in_data	Input	DATA_WIDTH	The data returned by the instruction memory.
fetch_address_in	Input	ADDRESS_BITS	The address of the data returned by the instruction memory interface.
memory_valid	Input	1	The data memory interface's data valid signal. A high value indicates the data returned by the instruction memory is valid.
memory_ready	Input	1	The data memory interface ready signal. When this signal is high, the data memory interface can accept new requests.
memory_data_in	Input	DATA_WIDTH	The data memory interface read data bus.

Table 4.33: Single Cycle Core Ports

Port	Direction	Width	Description
memory_address_in	Input	ADDRESS_BITS	The address associated with the memory_data_in value.
fetch_read	Output	1	The fetch interface read enable signal. Currently it is always set to 1.
fetch_read_address	Output	ADDRESS_BITS	The current PC register value.
memory_read	Output	1	The read enable signal of the data memory interface.
memory_write	Output	1	The write enable signal of the data memory interface.
memory_byte_en	Output	NUM_BYTES	The per-byte write enable signals. This signal is only used by write operations. All bytes are read for every read.
memory_address	Output	ADDRESS_BITS	The data memory interface address bus.
memory_data	Output	DATA_WIDTH	The data memory interface write data bus.
scan	Input	1	An active high signal to print debug information in simulation.

Table 4.34: Single Cycle Core Sub-Modules

Sub-Module	Description
fetch_issue	Holds the PC register and issues instruction read requests.
fetch_receive	Inserts NOP if received instruction is not valid.
decode_unit	Breaks instruction word into RS1, RS2 and immediate signals. Contains register file.
single_cycle_control_unit	The core-specific control unit to generate control signals throughout the core.
execution_unit	Selects the ALU input signals. Contains the ALU.
memory_issue	Initiates data memory loads and stores.
memory_receive	Selects byte, half-word or word from the data returned by the data memory interface.
writeback_unit	Selects between loaded data or ALU output for register file writing.

4.2.3 Five Stage Core

The base modules and the modules described here form a pipelined RV32I core with five pipeline stages. The Five Stage Core includes Fetch, Decode, Execution, Memory, and Writeback stages. The stage sub-modules are common to all of the cores. Each stage module is described in Section 4.2.1. Some base modules are wrapped with additional logic specific to this five stage core. The fetch and memory stages each take a single cycle, i.e the instruction and data memory reads must happen combinationally or the core will stall. If BRAM memories or a cache hierarchy are used, the core will stall every other cycle while the BRAM or cache is read. The seven stage core prevents stalls with an additional pipeline stage between issue and receive stages.

4.2.3.1 Five Stage Decode Unit

The decode unit of the Five Stage Core wraps the base **decode_unit**. The five stage decode module receives the instruction and the PC from the decode pipe. Additional inputs are provided by the **five_stage_control_unit** The module decodes the input instruction combinationally to output the appropriate XLEN immediate field and register file data for use in the execute stage. The instruction opcocde, funct3, and funct7 fields are also output to the control unit. The JAL target is calculated by adding the appropriate immediate field to the PC input.

Table 4.35: Five Stage Decode Unit Parameters

Parameters	**Description**
CORE	A unique number to identify the core this module is in.
DATA_WIDTH	The number of bits in the core data path. Except for highly customized and experimental designs, this will be equivalent to the RISC-V XLEN parameter. Expected values are 32 or 64.
ADDRESS_BITS	The number of bits in the address. Sets the width of the PC, branch_target and JAL_target ports.
SCAN_CYCLES_MIN	Prevents scan logic from printing for cycles less than this threshold.
SCAN_CYCLES_MAX	Prevents scan logic from printing for cycles greater than this threshold.

Table 4.36: Five Stage Decode Unit Ports

Port	**Direction**	**Width**	**Description**
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	Synchronous, active high reset signal. Sets x0 in the register file to zero and resets scan logic.
PC	Input	ADDRESS_BITS	The PC value for the input instruction.
instruction	Input	32	The input instruction to decode.
extend_sel	Input	2	Control bits to select between different immediate encodings for the extend_imm output.
write	Input	1	Active high register file write enable.
write_reg	Input	5	The register file address to write to.
write_data	Input	32	The data from the writeback stage to be written to the register file. The register file address is set by the write_reg input port.
rs1_data	Output	32	The register file data stored in the address selected by the instruction's rs1 bits.
rs2_data	Output	32	The register file data stored in the address selected by the instruction's rs2 bits.
rd	Output	5	The instruction input rd bits (instruction bits 7-11).
opcode	Output	7	The instruction input opcode bits (instruction bits 0-6).
funct7	Output	7	The instruction input funct7 bits (instruction bits 25-31).
funct3	Output	3	The instruction input funct3 bits (instruction bits 12-14).
extend_imm	Output	32	The XLEN extended immediate field of the instruction input.
branch_target	Output	ADDRESS_BITS	A branch target address. Generated by summing the decoded B-type immediate field with the PC input.

Table 4.36: Five Stage Decode Unit Ports

Port	Direction	Width	Description
JAL_target	Output	ADDRESS_BITS	A Jump And Link target address. Generated by summing the J-type immediate field wit the PC input.
rs1_data_bypass	Input	2	Control bits to select between the data stored in the rs1 register, the ALU_result_execute bypass signal, the ALU_result_memory bypass signal, or the ALU_result_writeback signal.
rs2_data_bypass	Input	2	Control bits to select between the data stored in the rs2 register, the ALU_result_execute bypass signal, the ALU_result_memory bypass signal, or the ALU_result_writeback signal.
ALU_result_execute	Input	DATA_WIDTH	The data received from the execution stage bypass signal.
ALU_result_memory	Input	DATA_WIDTH	The data received from the memory stage bypass signal.
ALU_result_writeback	Input	DATA_WIDTH	The data received from the writeback stage bypass signal.

Table 4.37: Five Stage Decode Unit Sub-Modules

Sub-Module	Description
decode_unit	The base decode unit that contains instruction parsing logic and the register file.

4.2.3.2 Five Stage Control Unit

The five stage control unit module combinationally generates signals for the whole processor. It connects the base control unit, hazard detection unit, five stage stall unit, and five stage bypass unit into a single control unit. Wrapping shared base madules with core specific logic maximizes code re-use and modularity in the Trireme Platform.

Table 4.38: Five Stage Control Unit Parameters

Parameters	Description
CORE	A unique number to identify the core this module is in.
ADDRESS_BITS	The number of bits in the address. Sets the width of the PC, branch_target and JAL_target ports.
NUM_BYTES	The number of bytes in the DATA_WIDTH parameter. (DATA_WIDTH is set in the module that instantiates this control unit. This parameter is equivalent to DATA_WIDTH/8.
LOG2_NUM_BYTES	The log2 of NUM_BYTES. The value is computed based on the set NUM_BYTES value. This parameter does not need to be set by the instantiating module. It is always safe to use the default value.
SCAN_CYCLES_MIN	Prevents scan logic from printing for cycles less than this threshold.
SCAN_CYCLES_MAX	Prevents scan logic from printing for cycles greater than this threshold.

Table 4.39: Five Stage Control Unit Ports

Port	Direction	Width	Description
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	Synchronous, active high reset signal. Sets x0 in the register file to zero and resets scan logic.
opcode_decode	Input	7	The decode stage's current instruction opcode field.
opcode_execute	Input	7	The execute stage's current instruction opcode field.
funct3	Input	3	The decode stage's current instruction funct3 field.
funct7	Input	7	The decode stage's current instruction funct7 field.
JALR_target_execute	Input	ADDRESS_BITS	The Jump And Link Register target address (where a JALR instruction would jump to) of the instruction in the execute stage.
branch_target_execute	Input	ADDRESS_BITS	The branch target address (where a taken branch of a branch instruction would jump to) of the instruction in the execute stage.
JAL_target_decode	Input	ADDRESS_BITS	The Jump And Link target address (where a JAL instruction would jump to) of the instruction in the decode stage.
branch_execute	Input	1	A control signal to indicate the instruction in the execute stage is branching. On a "branch taken" the signal will be 1. Otherwise it is 0.
branch_op	Output	1	A control signal for the execute stage to indicate the current instruction is a branch instruction.
memRead	Output	1	A control signal for the data memory interface to indicate that a memory read (load) should be performed. Set to 1 for load instructions.
ALU_operation	Output	6	A control signal to select the ALU operation performed in the execute stage. Only 4 bits are needed for the base instruction set. Additional bits are included to support ISA extensions and custom instructions.
memWrite	Output	1	A control signal for the data memory interface to indicate a memory write (store) should be performed. Set to 1 for store instructions.
log2_bytes	Output	LOG2_NUM_BYTES	A control signal for the data memory interface to indicate the number of bytes in a load or store operation. Set to 2 for 4-byte word operations. Set to 1 for 2-byte half-word operations. Set to 0 for byte operations.
unsigned_load	Output	1	A control signal used to sign extend loaded data values when high. Set to 1 for signed load operations.

Table 4.39: Five Stage Control Unit Ports

Port	Direction	Width	Description
next_PC_sel	Output	2	A control signal used to select the next PC value in the fetch issue stage. Selects between PC+4 (2'b00), PC (2'b01), and target PC (2'b10).
operand_A_sel	Output	2	A control signal used to select the value input into the operand A port of the ALU. A value of 2'b00 selects the data read from the register file. A 2'b01 value selects the PC of the instruction entering the execution unit. A 2'b10 value selects the execute instruction's PC+4. A 2'b11 value selects 0 (for LUIs).
operand_B_sel	Output	1	A control signal used to select between the rs2 data (0) and instruction immediate field (1).
extend_sel	Output	1	A control signal used to select between different instruction immediate field encodings in the decode unit. Setting 2'b00 (or 2'b11) selects the I-type immediate. Setting 2'b01 selects the S-type immediate. Setting 2'b10 selects the U-type immediate.
regWrite	Output	1	A control signal used to enable register file writes. Set to 1 for all non-store instructions.
target_PC	Output	ADDRESS_BITS	The target PC send to the fetch issue stage.
i_mem_read	Output	1	The instruction memory interface read enable signal. Currently it is always set to 1.
scan	Input	1	An active high signal to print debug information in simulation.
fetch_valid	Input	1	A signal that goes into the hazard detection unit and contributes to the assessment of whether there is an instruction memory hazard.
fetch_ready	Input	1	A signal that goes into the hazard detection unit and contributes to the assessment of whether there is an instruction memory hazard.
issue_PC	Input	ADDRESS_BITS	A signal from the fetch unit that goes into the hazard detection unit and contributes to the assessment of whether there is an instruction memory hazard.
fetch_address_in	Input	ADDRESS_BITS	This address contributes to the assessment of whether there is an instruction memory hazard.
memory_valid	Input	1	A signal that goes into the hazard detection unit and contributes to the assessment of whether there is a data memory hazard.
memory_ready	Input	1	A signal that goes into the hazard detection unit and contributes to the assessment of whether there is a data memory hazard.

Table 4.39: Five Stage Control Unit Ports

Port	Direction	Width	Description
load_memory	Input	1	A signal that goes into the hazard detection unit and contributes to the assessment of whether there is a data memory hazard.
store_memory	Input	1	A signal that is currently unused, but may be used to optimize the hazard detection logic for data memory hazards.
load_address	Input	ADDRESS_BITS	This address contributes to the assessment of whether there is a data memory hazard.
memory_address_in	Input	ADDRESS_BITS	This address contributes the assessment of whether there is a data memory hazard.
stall_decode	Output	1	A signal sent from the five stage stall unit to the decode pipe; when high, stalls the pipeline in the decode stage.
stall_execute	Output	1	A signal sent from the five stage stall unit to the execution stage; when high, stalls the pipeline in the execution stage.
stall_memory	Output	1	A signal sent from the five stage stall unit to the memory stage; when high, stalls the pipeline in the memory stage.
flush_decode	Output	1	A signal sent from the five stage stall unit to the decode stage; when high, flushes the decode stage pipe.
flush_execute	Output	1	A signal sent from the five stage stall unit to the execution unit; when high, flushes the execution stage pipe.
flush_writeback	Output	1	A signal sent from the five stage stall unit to the writeback unit; when high, flushes the writeback stage pipe.
rs1_data_bypass	Output	2	A signal sent to the decode unit to determine whether the rs1 data should be accessed from the regFile or a bypass from a later stage.
rs2_data_bypass	Output	2	A signal sent to the decode unit to determine whether the rs2 data should be accessed from the regFile or a bypass from a later stage.
rs1	Input	5	This port contributes to the determination of which stage, if any, has a hazard with the rs1 register.
rs2	Input	5	This port contributes to the determination of which stage, if any, has a hazard with the rs2 register.
rd_execute	Input	5	This port contributes to the determination of whether there is a hazard in the execution stage.
rd_memory	Input	5	This port contrinutes to the determination of whether there is a hazard in the memory stage.

Table 4.39: Five Stage Control Unit Ports

Port	Direction	Width	Description
rd_writeback	Input	5	This port contributes to the determination of whether there is a hazard in the writeback stage.
regWrite_execute	Input	1	A signal that contributes to the determination of whether there is a hazard in the execution stage.
regWrite_memory	Input	1	A signal that contributes to the determination of whether there is a hazard in the memory stage.
regWrite_writeback	Input	1	A signal that contributes to the determination of whether there is a hazard in the writeback stage.

Table 4.40: Five Stage Control Unit Sub-Modules

Sub-Module	Description
hazard_detection_unit	The Hazard Detection Unit takes in control signals and outputs hazard signals that can be decoded to set pipeline stalls or flush signals.
five_stage_stall_unit	The Five Stage Stall Unit takes in hazard signals and outputs stall or flush signals to the relevant stage.
five_stage_bypass_unit	The Five Stage Bypass Unit takes in hazard signals and outputs bypass signals.
control_unit	The base control unit to generate control signals common to each in-order core.

4.2.3.3 Five Stage Stall Unit

The five stage stall unit takes in signals from the hazard detection unit and the five stage control unit. It outputs stall signals to the decode, execute, memory pipes. Flush signals are sent to the decode, execute, and writeback pipes.

Table 4.41: Five Stage Stall Unit Parameters

Parameters	Description
CORE	A unique number to identify the core this module is in.
SCAN_CYCLES_MIN	Prevents scan logic from printing for cycles less than this threshold.
SCAN_CYCLES_MAX	Prevents scan logic from printing for cycles greater than this threshold.

Table 4.42: Five Stage Stall Unit Ports

Port	Direction	Width	Description
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	Synchronous, active high reset signal. Resets scan logic.
true_data_hazard	Input	1	A signal indicating that a true data hazard exists.
d_mem_hazard	Input	1	A signal indicating that a data memory hazard exists.
i_mem_hazard	Input	1	A signal indicating that an instruction memory hazard exists.
JALR_branch_hazard	Input	1	A signal indicating that a JALR or branch hazard exists.

Table 4.42: Five Stage Stall Unit Ports

Port	Direction	Width	Description
JAL_hazard	Input	1	A signal indicating that a JAL hazard exists.
stall_decode	Output	1	A signal that stalls the decode stage.
stall_execute	Output	1	A signal that stalls the execute stage.
stall_memory	Output	1	A signal that stalls the memory stage.
flush_decode	Output	1	A signal that enables a flush of the decode pipe.
flush_execute	Output	1	A signal that enables a flush of the execute pipe.
flush_writeback	Output	1	A signal that enables a flush of the writeback stage.

4.2.3.4 Five Stage Bypass Unit

The five stage bypass unit takes in hazard signals from the control unit, and outputs mux control signals to select a data source for the rs1_data and rs2_data output signals.

Table 4.43: Five Stage Bypass Unit Parameters

Parameters	Description
CORE	A unique number to identify the core this module is in.
SCAN_CYCLES_MIN	Prevents scan logic from printing for cycles less than this threshold.
SCAN_CYCLES_MAX	Prevents scan logic from printing for cycles greater than this threshold.

Table 4.44: Five Stage Bypass Unit Ports

Port	Direction	Width	Description
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	Synchronous, active high reset signal. Resets the scan logic.
true_data_hazard	Input	1	A signal indicating that a true data hazard exists.
rs1_hazard_execute	Input	1	A signal that indicates a hazard for the rs1 register in the execute stage.
rs1_hazard_memory	Input	1	A signal that indicates a hazard for the rs1 register in the memory stage.
rs1_hazard_writeback	Input	1	A signal that indicates a hazard for the rs1 register in the writeback stage.
rs2_hazard_execute	Input	1	A signal that indicates a hazard for the rs2 register in the execute stage.
rs2_hazard_memory	Input	1	A signal that indicates a hazard for the rs2 register in the memory stage.
rs2_hazard_writeback	Input	1	A signal that indicates a hazard for the rs2 register in the writeback stage.
rs1_data_bypass	Output	2	A signal sent to the decode unit to determine whether the rs1 data should be accessed from the regFile or a bypass from a later stage.

Table 4.44: Five Stage Bypass Unit Ports

Port	Direction	Width	Description
rs2_data_bypass	Output	2	A signal sent to the decode unit to determine whether the rs2 data should be accessed from the regFile or a bypass from a later stage.

4.2.4 Seven Stage Core

The base modules and the modules described here form a pipelined RV32I core with seven pipeline stages. The Seven Stage Core includes Fetch Issue, Fetch Receive, Decode, Execution, Memory Issue, Memory Receive, and Writeback stages. The stage sub-modules are common to all of the cores. Each stage module is described in Section 4.2.1. Some base modules are wrapped with additional logic specific to this seven stage core. The fetch and memory stages each take two cycles. In the first cycle, the core issues a fetch or data memory operation. In the second cycle, the core receives a fetch or data memory data if the issued operation was a read. Using two pipeline stages allows the core to issue memory operations each cycle, even to memories with a one cycle latency, such as BRAMs or L1 cache.

4.2.4.1 Seven Stage Decode Unit

The seven stage decode module instantiates the base **decode_unit** module, which, contains the register file. These modules are described in Section 4.2.1. The seven stage decode module receives two signals from the decode pipe that precedes it, the instruction and the PC. It also takes inputs from the control unit, and processor data-path (for pipeline forwarding). The module decodes the input instruction combinationally to output the appropriate XLEN immediate field and register file data for use in the execute stage. The branch and JAL targets are calculated by adding the appropriate immediate type to the PC input. The module outputs the branch/JAL targets and instruction opcode, funct3, and funct7 fields to the control unit.

Table 4.45: Seven Stage Decode Unit Parameters

Parameters	Description
CORE	A unique number to identify the core this module is in.
DATA_WIDTH	The number of bits in the core data path. Except for highly customized and experimental designs, this will be equivalent to the RISC-V XLEN parameter. Expected values are 32 or 64.
ADDRESS_BITS	The number of bits in the address. Sets the width of the PC, branch_target and JAL_target ports.
SCAN_CYCLES_MIN	Prevents scan logic from printing for cycles less than this threshold.
SCAN_CYCLES_MAX	Prevents scan logic from printing for cycles greater than this threshold.

Table 4.46: Seven Stage Decode Unit ports

Port	Direction	Width	Description
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	A synchronous, active high reset signal. sets x0 in the register file to zero and resets scan logic.
pc	Input	ADDRESS_BITS	the pc value for the input instruction.
instruction	Input	32	The input instruction to decode.
extend_sel	Input	2	Control bits to select between different immediate encodings for the extend_imm output.

Table 4.46: Seven Stage Decode Unit ports

Port	Direction	Width	Description
write	Input	1	Active high register file write enable.
write_reg	Input	5	The register file address to write to.
write_data	Input	32	The data from the writeback stage to be written to the register file. The register file address is set by the write_reg input port.
rs1_data	Output	32	The register file data stored in the address selected by the instruction's rs1 bits.
rs2_data	Output	32	The register file data stored in the address selected by the instruction's rs2 bits.
rd	Output	5	The instruction input rd bits (instruction bits 7-11).
opcode	Output	7	The instruction input opcode bits (instruction bits 0-6).
funct7	Output	7	The instruction input funct7 bits (instruction bits 25-31).
funct3	Output	3	The instruction input funct3 bits (instruction bits 12-14).
extend_imm	Output	32	The XLEN extended immediate field of the instruction input.
branch_target	Output	ADDRESS_BITS	A branch target address. Generated by summing the decoded B-type immediate field with the PC input.
JAL_target	Output	ADDRESS_BITS	A Jump And Link target address. Generated by summing the J-type immediate field wit the PC input.
rs1_data_bypass	Input	2	Control bits to select between the data stored in the rs1 register, the ALU_result_execute bypass signal, the ALU_result_memory bypass signal, or the ALU_result_writeback signal.
rs2_data_bypass	Input	2	Control bits to select between the data stored in the rs2 register, the ALU_result_execute bypass signal, the ALU_result_memory bypass signal, or the ALU_result_writeback signal.
ALU_result_execute	Input	DATA_WIDTH	The data received from the execution stage bypass signal.
ALU_result_memory	Input	DATA_WIDTH	The data received from the memory stage bypass signal.
ALU_result_writeback	Input	DATA_WIDTH	The data received from the writeback stage bypass signal.

Table 4.47: Seven Stage Decode Unit Sub-Modules

Sub-Module	Description
decode_unit	The base decode unit that contains instruction parsing logic and the register file.

4.2.4.2 Seven Stage Control Unit

The seven stage control unit module combinationally generates signals for the whole processor. It connects the base control unit, hazard detection unit, seven stage stall unit, and seven stage bypass unit into a single control unit. Wrapping shared base madules with core specific logic maximizes code re-use and modularity in the Trireme Platform.

Table 4.48: Seven Stage Control Unit Parameters

Parameters	**Description**
CORE	A unique number to identify the core this module is in.
ADDRESS_BITS	The number of bits in the address. Sets the width of the PC, branch_target and JAL_target ports.
NUM_BYTES	The number of bytes in the DATA_WIDTH parameter. (DATA_WIDTH is set in the module that instantiates this control unit. This parameter is equivalent to DATA_WIDTH.
LOG2_NUM_BYTES	The log2 of NUM_BYTES. The value is computed based on the set NUM_BYTES value. This parameter does not need to be set by the instantiating module. It is always safe to use the default value.
SCAN_CYCLES_MIN	Prevents scan logic from printing for cycles less than this threshold.
SCAN_CYCLES_MAX	Prevents scan logic from printing for cycles greater than this threshold.

Table 4.49: Seven Stage Control Unit Ports

Port	**Direction**	**Width**	**Description**
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	Synchronous, active high reset signal. Sets x0 in the register file to zero and resets scan logic.
opcode_decode	Input	7	The decode stage's current instruction opcode field.
opcode_execute	Input	7	The execute stage's current instruction opcode field.
opcode_memory_issue	Input	7	The memory issue stage's current instruction opcode field.
opcode_memory_receive	Input	7	The memory receive stage's current instruction opcode field.
funct3	Input	3	The decode stage's current instruction funct3 field.
funct7	Input	7	The decode stage's current instruction funct7 field.
JALR_target_execute	Input	ADDRESS_BITS	The Jump And Link Register target address (where a JALR instruction would jump to) of the instruction in the execute stage.
branch_target_execute	Input	ADDRESS_BITS	The branch target address (where a taken branch of a branch instruction would jump to) of the instruction in the execute stage.

Table 4.49: Seven Stage Control Unit Ports

Port	Direction	Width	Description
JAL_target_decode	Input	ADDRESS_BITS	The Jump And Link target address (where a JAL instruction would jump to) of the instruction in the decode stage.
branch_execute	Input	1	A control signal to indicate the instruction in the execute stage is branching. On a "branch taken" the signal will be 1. Otherwise it is 0.
branch_op	Output	1	A control signal for the execute stage to indicate the current instruction is a branch instruction.
memRead	Output	1	A control signal for the data memory interface to indicate that a memory read (load) should be performed. Set to 1 for load instructions.
ALU_operation	Output	6	A control signal to select the ALU operation performed in the execute stage. Only 4 bits are needed for the base instruction set. Additional bits are included to support ISA extensions and custom instructions.
memWrite	Output	1	A control signal for the data memory interface to indicate a memory write (store) should be performed. Set to 1 for store instructions.
log2_bytes	Output	LOG2_NUM_BYTES	A control signal for the data memory interface to indicate the number of bytes in a load or store operation. Set to 2 for 4-byte word operations. Set to 1 for 2-byte half-word operations. Set to 0 for byte operations.
unsigned_load	Output	1	A control signal used to sign extend loaded data values when high. Set to 1 for signed load operations.
next_PC_sel	Output	2	A control signal used to select the next PC value in the fetch issue stage. Selects between PC+4 (2'b00), PC (2'b01), and target PC (2'b10).
operand_A_sel	Output	2	A control signal used to select the value input into the operand A port of the ALU. A value of 2'b00 selects the data read from the register file. A 2'b01 value selects the PC of the instruction entering the execution unit. A 2'b10 value selects the execute instruction's PC+4. A 2'b11 value selects 0 (for LUIs).
operand_B_sel	Output	1	A control signal used to select between the rs2 data (0) and instruction immediate field (1).
extend_sel	Output	1	A control signal used to select between different instruction immediate field encodings in the decode unit. Setting 2'b00 (or 2'b11) selects the I-type immediate. Setting 2'b01 selects the S-type immediate. Setting 2'b10 selects the U-type immediate.

Table 4.49: Seven Stage Control Unit Ports

Port	Direction	Width	Description
regWrite	Output	1	A control signal used to enable register file writes. Set to 1 for all non-store instructions.
target_PC	Output	ADDRESS_BITS	The target PC send to the fetch issue stage.
i_mem_read	Output	1	The instruction memory interface read enable signal. Currently it is always set to 1.
scan	Input	1	An active high signal to print debug information in simulation.
fetch_valid	Input	1	A signal that goes into the hazard detection unit and contributes to the assessment of whether there is an instruction memory hazard.
fetch_ready	Input	1	A signal that goes into the hazard detection unit and contributes to the assessment of whether there is an instruction memory hazard.
issue_PC	Input	ADDRESS_BITS	A signal from the fetch unit that goes into the hazard detection unit and contributes to the assessment of whether there is an instruction memory hazard.
fetch_address_in	Input	ADDRESS_BITS	This address contributes to the assessment of whether there is an instruction memory hazard.
memory_valid	Input	1	A signal that goes into the hazard detection unit and contributes to the assessment of whether there is a data memory hazard.
memory_ready	Input	1	A signal that goes into the hazard detection unit and contributes to the assessment of whether there is a data memory hazard.
load_memory_receive	Input	1	A signal that goes into the hazard detection unit and contributes to the assessment of whether there is a data memory hazard.
store_memory_issue	Input	1	A signal that is currently unused, but may be used to optimize the hazard detection logic for data memory hazards.
load_address_receive	Input	ADDRESS_BITS	This address contributes to the assessment of whether there is a data memory hazard.
memory_address_in	Input	ADDRESS_BITS	This address contributes the assessment of whether there is a data memory hazard.
stall_fetch_receive	Output	1	A signal sent from the seven stage stall unit to the fetch receive pipe; when high, stalls the pipeline in the fetch receive stage.
stall_decode	Output	1	A signal sent from the seven stage stall unit to the decode pipe; when high, stalls the pipeline in the decode stage.

Table 4.49: Seven Stage Control Unit Ports

Port	**Direction**	**Width**	**Description**
stall_execute	Output	1	A signal sent from the seven stage stall unit to the execution stage; when high, stalls the pipeline in the execution stage.
stall_memory_issue	Output	1	A signal sent from the seven stage stall unit to the memory issue stage; when high, stalls the pipeline in the memory issue stage.
stall_memory_receive	Output	1	A signal sent from the seven stage stall unit to the memory receive stage; when high, stalls the pipeline in the memory receive stage.
flush_fetch_receive	Output	1	A signal sent from the seven stage stall unit to the fetch receive stage; when high, flushes the fetch receive stage pipe.
flush_decode	Output	1	A signal sent from the seven stage stall unit to the decode stage; when high, flushes the decode stage pipe.
flush_execute	Output	1	A signal sent from the seven stage stall unit to the execution unit; when high, flushes the execution stage pipe.
flush_writeback	Output	1	A signal sent from the seven stage stall unit to the writeback unit; when high, flushes the writeback stage pipe.
rs1_data_bypass	Output	2	A signal sent to the decode unit to determine whether the rs1 data should be accessed from the regFile or a bypass from a later stage.
rs2_data_bypass	Output	2	A signal sent to the decode unit to determine whether the rs2 data should be accessed from the regFile or a bypass from a later stage.
rs1	Input	5	This port contributes to the determination of which stage, if any, has a hazard with the rs1 register.
rs2	Input	5	This port contributes to the determination of which stage, if any, has a hazard with the rs2 register.
rd_execute	Input	5	This port contributes to the determination of whether there is a hazard in the execution stage.
rd_memory_issue	Input	5	This port contributes to the determination of whether there is a hazard in the memory issue stage.
rd_memory_receive	Input	5	This port contributes to the determination of whether there is a hazard in the memory receive stage.
rd_writeback	Input	5	This port contributes to the determination of whether there is a hazard in the writeback stage.

Table 4.49: Seven Stage Control Unit Ports

Port	Direction	Width	Description
regWrite_execute	Input	1	A signal that contributes to the determination of whether there is a hazard in the execution stage.
regWrite_memory_issue	Input	1	A signal that contributes to the determination of whether there is a hazard in the memory issue stage.
regWrite_memory_receive	Input	1	A signal that contributes to the determination of whether there is a hazard in the memory receive stage.
regWrite_writeback	Input	1	A signal that contributes to the determination of whether there is a hazard in the writeback stage.
issue_request	Output	1	A signal that goes to both instruction memory and the fetch receive module requesting an instruction. Always set to high.

Table 4.50: Seven Stage Control Unit Sub-Modules

Sub-Module	Description
hazard_detection_unit	The Hazard Detection Unit takes in control signals and outputs hazard signals that can be decoded to set pipeline stalls or flush signals.
seven_stage_stall_unit	The Seven Stage Stall Unit takes in hazard signals and outputs stall or flush signals to the relevant stage.
seven_stage_bypass_unit	The Seven Stage Bypass Unit takes in hazard signals and outputs bypass signals.
control_unit	The base control unit to generate control signals common to each in-order core.

4.2.4.3 Seven Stage Stall Unit

The seven stage stall unit takes in signals from the hazard detection unit and the seven stage control unit. It outputs stall signals to the fetch receive, decode, execute, memory issue, and memory receive pipes. Flush signals are sent to the fetch receive, decode, execute, and writeback pipes.

Table 4.51: Seven Stage Stall Unit Parameters

Parameters	Description
CORE	A unique number to identify the core this module is in.
SCAN_CYCLES_MIN	Prevents scan logic from printing for cycles less than this threshold.
SCAN_CYCLES_MAX	Prevents scan logic from printing for cycles greater than this threshold.

Table 4.52: Seven Stage Stall Unit Ports

Port	Direction	Width	Description
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	Synchronous, active high reset signal. Resets scan logic.
true_data_hazard	Input	1	A signal indicating that a true data hazard exists.
d_mem_hazard	Input	1	A signal indicating that a data memory hazard exists.
i_mem_hazard	Input	1	A signal indicating that an instruction memory hazard exists.
JALR_branch_hazard	Input	1	A signal indicating that a JALR or branch hazard exists.
JAL_hazard	Input	1	A signal indicating that a JAL hazard exists.
clog	Input	1	A signal indicating that the decode pipe is clogged, and a stall is necessary.
stall_fetch_receive	Output	1	A signal that stalls the fetch receive stage.
stall_decode	Output	1	A signal that stalls the decode stage.
stall_execute	Output	1	A signal that stalls the execute stage.
stall_memory_issue	Output	1	A signal that stalls the memory issue stage.
stall_memory_receive	Output	1	A signal that stalls the memory receive stage.
flush_fetch_receive	Output	1	A signal that enables a flush of the fetch receive pipe.
flush_decode	Output	1	A signal that enables a flush of the decode pipe.
flush_execute	Output	1	A signal that enables a flush of the execute pipe.
flush_writeback	Output	1	A signal that enables a flush of the writeback stage.

4.2.4.4 Seven Stage Bypass Unit

The seven stage bypass unit takes in hazard signals from the control unit, and sends out a signal that selects which source should be used to provide the data for the rs1_data and rs2_data signals from the decode unit to the execute pipe.

Table 4.53: Seven Stage Bypass Unit Parameters

Parameters	Description
CORE	A unique number to identify the core this module is in.
SCAN_CYCLES_MIN	Prevents scan logic from printing for cycles less than this threshold.
SCAN_CYCLES_MAX	Prevents scan logic from printing for cycles greater than this threshold.

Table 4.54: Seven Stage Bypass Unit Ports

Port	Direction	Width	Description
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	Synchronous, active high reset signal. Resets the scan logic.
true_data_hazard	Input	1	A signal indicating that a true data hazard exists.

Table 4.54: Seven Stage Bypass Unit Ports

Port	Direction	Width	Description
rs1_hazard_execute	Input	1	A signal that indicates a hazard for the rs1 register in the execute stage.
rs1_hazard_memory_issue	Input	1	A signal that indicates a hazard for the rs1 register in the memory issue stage.
rs1_hazard_memory_receive	Input	1	A signal that indicates a hazard for the rs1 register in the memory receive stage.
rs1_hazard_writeback	Input	1	A signal that indicates a hazard for the rs1 register in the writeback stage.
rs2_hazard_execute	Input	1	A signal that indicates a hazard for the rs2 register in the execute stage.
rs2_hazard_memory_issue	Input	1	A signal that indicates a hazard for the rs2 register in the memory issue stage.
rs2_hazard_memory_receive	Input	1	A signal that indicates a hazard for the rs2 register in the memory receive stage.
rs2_hazard_writeback	Input	1	A signal that indicates a hazard for the rs2 register in the writeback stage.
rs1_data_bypass	Output	3	A signal sent to the decode unit to determine whether the rs1 data should be accessed from the regFile or a bypass from a later stage.
rs2_data_bypass	Output	3	A signal sent to the decode unit to determine whether the rs2 data should be accessed from the regFile or a bypass from a later stage.

4.3 Memory Designs

The different memory hierarchies are included in The Trireme Platform.

4.3.1 Base Cache Subsystem Modules

The memory hierarchy of the Trireme Platform is modular in design. This allows for a substantial amount of module reuse. This subsection includes these common modules.

4.3.1.1 Cache Memory

This module creates cache memory, using two **dual_ port_ RAM** modules per cache way, one for data storage and one for metadata storage. It also instantiates three **one_hot_decoder** modules per cache way and one replacement controller module per cache way.

Table 4.55: Cache Memory Unit Parameters

Parameters	Description
STATUS_BITS	The number of bits needed to express the status of the cache line at a cache port.
COHERENCE_BITS	The number of bits needed to express the status of the cache line's coherence state.
OFFSET_BITS	A figure that determines, along with ADDRESS_BITS and INDEX_BITS, the number of tag bits for the cache memory. It also helps determine the number of WORDS_PER_LINE in the cache.

Table 4.55: Cache Memory Unit Parameters

Parameters	**Description**
DATA_WIDTH	The number of bits in the core data path. Except for highly customized and experimental designs, this will be equivalent to the RISC-V XLEN parameter. Expected values are 32 or 64.
NUMBER_OF_WAYS	The number of ways in the cache.
REPLACEMENT_MODE	A one-bit parameter to select the replacement mode; 0 selects random replacement and 1 selects LRU replacement.
ADDRESS_BITS	The number of bits in the address.
INDEX_BITS	The number of bits in the memory address index.
READ_DURING_WRITE	A parameter permits cross-port read during write behavior. It is overridden when the cache memory module is used as a directory cache.
TAG_BITS	The numbrer of bits used to tag cache entries.
WORDS_PER_LINE	The number of words in each line of the cache.
BLOCK_WIDTH	The bit width of each block of the cache.
SBITS	The number of bits in each cache entry's metadata.
MBITS	The number of bits in each cache entry's metadata and tag.
WAY_BITS	The number of bits needed to index the ways in the cache.
COH_BITS	The number of bits needed to express the coherence status of a cache entry.

Table 4.56: Cache Memory Unit Ports

Port	**Direction**	**Width**	**Description**
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	Synchronous, active high reset signal.
read0	Input	1	A signal enabling a read from port0.
write0	Input	1	A signal enabling a write to port0.
invalidate0	Input	1	A signal that invalidates the cache entry at port0.
index0	Input	INDEX_BITS	The index of the cache entry at port0.
tag0	Input	TAG_BITS	The tag of the cache entry at port0.
meta_data0	Input	SBITS	The metadata of the cache entry at port0.
data_in0	Input	BLOCK_WIDTH	The data to be written in the cache entry at port0.
way_select0	Input	WAY_BITS	The way selected for the cache entry at port0.
data_out0	Output	BLOCK_WIDTH	The data to be read from the cache entry at port0.
tag_out0	Output	TAG_BITS	The tag of the cache entry at port0.
matched_way0	Output	WAY_BITS	A signal to indicate that the selected way is the correct one.
coh_bits0	Output	COH_BITS	???
status_bits0	Output	STATUS_BITS	The status bits for the cache line at port0.
hit0	Output	1	A signal indicating a hit on the cache line at port0.

Table 4.56: Cache Memory Unit Ports

Port	Direction	Width	Description
read1	Input	1	A signal enabling a read from port1.
write1	Input	1	A signal enabling a write to port1.
invalidate1	Input	1	A signal that invalidates the cache entry at port1.
index1	Input	INDEX_BITS	The index of the cache entry at port1.
tag1	Input	TAG_BITS	The tag of the cache entry at port1.
meta_data1	Input	SBITS	The metadata of the cache entry at port1.
data_in1	Input	BLOCK_WIDTH	The data to be written in the cache entry at port1.
way_select1	Input	WAY_BITS	The way selected for the cache entry at port1.
data_out1	Output	BLOCK_WIDTH	The data to be read from the cache entry at port1.
tag_out1	Output	TAG_BITS	The tag of the cache entry at port1.
matched_way1	Output	WAY_BITS	A signal to indicate that the selected way is the correct one.
coh_bits1	Output	COH_BITS	???
status_bits1	Output	STATUS_BITS	The status bits for the cache line at port1.
hit1	Output	1	A signal indicating a hit on the cache line at port1.

Table 4.57: Cache Memory Unit Sub-Modules

Sub-Module	Description
dual_port_ram	The cache memory instantiates two dual_port_ram modules for each cache way, one for data and one for metadata.
one_hot_decoder	The cache memory instantiates three one_hot_decoder modules per cache way. These track tag matches and their validity, as well as which way should be replaced.
replacement_controller	The cache memory instantiates one replacement_controlloer module per cache way. This governs which cache line is chosen for replacement.

4.3.1.2 One Hot Decoder Unit

The **one_hot_ decoder** module instatntiations report whether the cache line(s) requested at port0 and/or port1 hit or missed, as well as the selected way from the replacement controller unit.

Table 4.58: One Hot Decoder Unit Parameters

Parameters	Description
WIDTH	The width of the tag_match0, tag_match1, and replace_way_encoded input values.

Table 4.59: One Hot Decoder Unit Ports

Port	Direction	Width	Description
encoded	Input	WIDTH	The encoded value to be decoded.

Table 4.59: One Hot Decoder Unit Ports

Port	Direction	Width	Description
decoded	Output	log2(WIDTH)	The decoded value resulting from decoding.
valid	Output	1	A signal to indicate that the output decoded value is valid.

4.3.1.3 Replacement Controller

The **replacement_controller** implements a method for selecting an empty cache way to fill, or, finding no empty ways, implements a least recently used replacement scheme. It instantiates the **empty_way_select** and **LRU** units.

Table 4.60: Replacement Controller Parameters

Parameters	Description
NUMBER_OF_WAYS	The number of ways in the cache.
INDEX_BITS	The number of bits in the memory address index.

Table 4.61: Replacement Controller Ports

Port	Direction	Width	Description
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	Synchronous, active high reset signal.
ways_in_use	Input	NUMBER_OF_WAYS	Indicates which cache ways in use.
current_index	Input	INDEX_BITS	The current memory index.
replacement_policy_select	Input	1	Chooses whether to use the LRU (0) or random (1) replacement policy.
current_access	Input	log2(NUMBER_OF_WAYS)	The index to the internal array containing the write order.
access_valid	Input	1	A signal to indicate whether an element in the write order array is valid.
scan	Input	1	CURRENTLY CALLED REPORT IN THIS MODULE - NEEDS TO BE UPDATED!
selected_way	Output	NUMBER_OF_WAYS	The way to be filled after the replacement controller logic runs.

Table 4.62: Replacement Controller Sub-Modules

Sub-Module	Description
LRU	Implements the Least Recently Used replacement policy.
Empty Way Select	Checks whether there is an empty way and, if so, selects it.

4.3.1.4 Empty Way Select Module

The **empty_way_ select** module keeps track of which cache ways are in use and whether any way is empty.

Table 4.63: Empty Way Select Unit Parameters

Parameters	Description
NUMBER_OF_WAYS	The number of ways in the cache.

Table 4.64: Empty Way Select Unit Ports

Port	Direction	Width	Description
ways_in_use	Input	NUMBER_OF_WAYS	Indicates which cache ways are in use.
next_empty_way	Output	NUMBER_OF_WAYS	Indicates which cache way is the next one to use.
valid	Output	1	A signal that indicates whether the next_empty_way should be used.

4.3.1.5 LRU

The**LRU** module institutes a Least Recently Used replacement scheme for cache line replacement. It instantiates the **dual_port_RAM** unit.

Table 4.65: LRU Parameters

Parameters	Description
WIDTH	Used with the log2 function to determine the LRU memory depth.
INDEX_BITS	The number of bits in the memory address index.

Table 4.66: LRU Ports

Port	Direction	Width	Description
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	Synchronous, active high reset signal.
current_index	Input	INDEX_BITS	The current memory index.
access	Input	log2(WIDTH)	The index to the internal array containing the write order. CHECK THIS WITH SAHAN
access_valid	Input	1	A signal to indicate whether an element in the write order array is valid. CHECK WITH SAHAN
lru	Output	WIDTH	An array whose contents indicate which element is the least recently used cache line. CHECK WITH SAHAN

Table 4.67: LRU Sub-Modules

Sub-Module	Description
dual_port_ram	The Dual Port RAM provides storage space for the LRU. CHECK WITH SAHAN

4.3.1.6 Dual Port RAM

The **dual_port_ RAM** module includes pass-through logic. It instantiates the **simple_dual_port_RAM**. In addition, it adds a parameter to choose between new and old data when one port reads the address written by the other.

Table 4.68: Dual Port RAM Parameters

Parameters	Description
DATA_WIDTH	The number of bits in the core data path. Except for highly customized and experimental designs, this will be equivalent to the RISC-V XLEN parameter. Expected values are 32 or 64.
ADDRESS_WIDTH	The number of bits in the address. NAME NEEDS TO BE CHANGED TO ADDRESS_BITS to match cores etc
INDEX_BITS	The number of bits in the memory address index.
RW	Determines whether new or old data is read when one port reads the address written by the other port.

Table 4.69: Dual Port RAM Ports

Port	Direction	Width	Description
clock	Input	1	The clock signal for synchronous logic.
we0	Input	1	A signal that indicates a request to write to port0 when high. It contributes to the determination of whether port0 may write (since port1 has priority).
we1	Input	1	A signal that enables a write to port1 when high.
data_in0	Input	DATA_WIDTH	The data to be written to port0 when writing to port0 is enabled.
data_in1	Input	DATA_WIDTH	The data to be written to port1 when writing to port1 is enabled.
address0	Input	INDEX_BITS	The memory index of the address to be read from, or written to, port0.
address1	Input	INDEX_BITS	The memory index of the address to be read from, or written to, port1.
data_out0	Output	DATA_WIDTH	The data to be read from port0.
data_out1	Output	DATA_WIDTH	The data to be read from port1.

Table 4.70: Dual Port RAM Sub-Modules

Sub-Module	Description
simple_dual_port_ram	The Simple Dual Port RAM module creates the RAM accessed in the Dual Port RAM.

4.3.1.7 Simple Dual Port RAM

The **simple_dual_port_RAM** is a dual port RAM without pass-through logic for cross-port read and write. When both ports seek to write simultaneously to the same address, port1 gets priority. Note: the **simple_dual_port_RAM** does not have a reset signal. Resetting the RAM is controlled by the cache controller module.

Table 4.71: Simple Dual Port RAM Parameters

Parameters	Description
DATA_WIDTH	The number of bits in the core data path. Except for highly customized and experimental designs, this will be equivalent to the RISC-V XLEN parameter. Expected values are 32 or 64.
ADDRESS_WIDTH	The number of bits in the address. NAME NEEDS TO BE CHANGED TO ADDRESS_BITS to match cores etc
INDEX_BITS	The number of bits in the memory address index.

Table 4.72: Simple Dual Port RAM Ports

Port	Direction	Width	Description
clock	Input	1	The clock signal for synchronous logic.
we0	Input	1	A signal that indicates a request to write to port0 when high. It contributes to the determination of whether port0 may write (since port1 has priority).
we1	Input	1	A signal that enables a write to port1 when high.
data_in0	Input	DATA_WIDTH	The data to be written to port0 when writing to port0 is enabled.
data_in1	Input	DATA_WIDTH	The data to be written to port1 when writing to port1 is enabled.
address0	Input	INDEX_BITS	The memory index of the address to be read from, or written to, port0.
address1	Input	INDEX_BITS	The memory index of the address to be read from, or written to, port1.
data_out0	Output	DATA_WIDTH	The data to be read from port0.
data_out1	Output	DATA_WIDTH	The data to be read from port1.

4.3.1.8 Coherence Controller

The **coherence_controller** serves as the bus master for any bus connecting two cache layers. It instantiates the **rr_arbiter**, the **one_hot_encoder**, and the **priority_encoder**. Note that the description of the **priority_encoder** can be found in Section 4.1.
The **coherence_controller** will be used with most cache instantiations. The exception is a design that has only an L1 cache connected directly to the NOC. Connecting the last level cache (whether L1 or a shared cache) to the NOC will use the **directory_controller** module, and thus, that module is described with the rest of the NOC-related memory modules.

Table 4.73: Coherence Controller Unit Parameters

Parameters	Description
MSG_BITS	The number of bits in a bus message.
NUM_CACHES	The number of caches connected by the bus governed by the controller.

Table 4.74: Coherence Controller Ports

Port	Direction	Width	Description
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	Synchronous, active high reset signal.
cache2mem_msg	Input	NUM_CACHES*MSG_BITS	The message to be transmitted from the cache to the memory on the bus.
mem2controller_msg	Input	MSG_BITS	The message to be transmitted from the memory to the controller.
bus_msg	Input	MSG_BITS	The message on the bus.
bus_control	Output	BUS_SIG_WIDTH	The control signal for the bus.
bus_en	Output	1	A signal to enable the bus.
req_ready	Output	1	A signal indicating a request is ready on the bus.
curr_master	Output	BUS_PORTS	A one-hot signal to indicate which port has control of the bus.

Table 4.75: Coherence Controller Sub-Modules

Sub-Module	Description
rr_arbiter	The round robin arbiter determines which request will be served next.
one_hot_encoder	The one hot encoder encodes the request to be served next.
priority_encoder	The priority encoder module combinationally selects one request from the vector of one bit request signals.

4.3.1.9 Round Robin Arbiter

The **rr_arbiter** unit takes in requests and determines which request to serve next. It instantiates the **priority_encoder** twice, once for masked requests and once for unmasked requests. Recall that the **priority_encoder** description can be found in Section 4.1.

Table 4.76: Round Robin Arbiter Module Parameters

Parameters	Description
WIDTH	The number of bits in the request vector.

Table 4.77: Round Robin Arbiter Module Ports

Port	Direction	Width	Description
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	Synchronous, active high reset signal.
requests	Input	WIDTH	A vector of requests for the arbited resource.
grant	Output	log2(WIDTH)	The identifier of the request that will be granted access to the resource.
valid	Output	1	A signal to indicate that the grant value is valid.

Table 4.78: Round Robin Arbiter Module Sub-Modules

Sub-Module	Description
priority_encoder	The priority encoder module combinationally selects one request from the vector of one bit request signals.

4.3.1.10 One Hot Encoder Unit

The **one_hot_encoder** unit creates a bit vector that encodes which requester should have access to the requested resource next.

Table 4.79: One Hot Encoder Module Parameters

Parameters	Description
WIDTH	The number of bits in the output vector.

Table 4.80: Round Robin Arbiter Module Ports

Port	Direction	Width	Description
in	Input	log2(WIDTH)	The signal to be represented in the encoded vector.
valid_input	Input	1	A signal to indicate whether the in signal is valid.
out	Output	WIDTH	The encoded output vector.

4.3.1.11 Mux Bus Unit

The **mux_bus** unit provides all buses for the cache structure.

Table 4.81: Mux Bus Unit Parameters

Parameters	Description
WIDTH	The width of the bus.
NUM_PORTS	The number of ports connected to the bus.

Table 4.82: Mux Bus Unit Ports

Port	Direction	Width	Description
data_in	Input	WIDTH*NUM_PORTS	The data entering the bus.
enable_port	Input	log2(NUM_PORTS)	A signal to index into the internal inputs vector.
valid_enable	Input	1	A signal indicating the vaildity of the enabled port.
data_out	Output	WIDTH	The data output from the bus.

4.3.1.12 Message Handler

The **message_handler** module receives messages from the NoC and buffers them in separate FIFOs. If the request buffer is full, a signal is sent back to the requester. Responses are given priority, because they cannot be rejected. It instantiates two **fifo** buffers, one each for requests and responses. The FIFO module is described in 4.1.

Table 4.83: Message Handler Module Parameters

Parameters	**Description**
CACHE_OFFSET_BITS	The number of bits in the cache offset.
DATA_WIDTH	The number of bits in the core data path. Except for highly customized and experimental designs, this will be equivalent to the RISC-V XLEN parameter. Expected values are 32 or 64.
ADDRESS_BITS	The number of bits in the address.
MSG_BITS	The number of bits in a message on the bus.
REQ_BUF_DEPTH_BITS	The number of bits needed to represent the depth of the request buffer's depth.
RESP_BUF_DEPTH_BITS	The number of bits needed to represent the depth of the response buffer's depth.
PARENT	Indicates whether the parent module is a cache or a driectory. Used to filter messages received and put them into the request and response FIFOs.
ID_BITS	The number of bits in the source and destination IDs.
DEFAULT_DEST	Makes the destination with ID = 0 the default destination.
CACHE_WORDS	The number of words in a cache line.
CACHE_WIDTH	The number of bits in a cache line.
BUF_WIDTH	The width, in bits, of the buffer.

Table 4.84: Message Handler Module Ports

Port	**Direction**	**Width**	**Description**
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	Synchronous, active high reset signal.
noc_msg_in	Input	MSG_BITS	The message from the NoC.
noc_address_in	Input	ADDRESS_BITS	The address from the NoC.
noc_data_in	Input	CACHE_WIDTH	The data from the NoC.
noc_src_id	Input	ID_BITS	The ID of the NoC source.
packetizer_busy	Input	1	A signal to inidcate whether the packetizer is busy.
noc_msg_out	Output	MSG_BITS	The message out to the NoC.
noc_address_out	Output	ADDRESS_BITS	The address out to the NoC.
noc_data_out	Output	CACHE_WIDTH	The data out to the NoC.
noc_dest_id	Output	ID_BITS	The ID of the NoC destination.
ctrl_msg_in	Input	MSG_BITS	The message in from the parent module's controller unit.
ctrl_address_in	Input	ADDRESS_BITS	The address in from the parent module's controller unit.
ctrl_data_in	Input	CACHE_WIDTH	The data in from the parent module's controller unit.
ctrl_dest_in	Input	ID_BITS	The ID of the destination from the parent module's controller unit.
intf_busy	Output	1	A signal to indicate that the parent module's controller unit is busy.

Table 4.84: Message Handler Module Ports

Port	Direction	Width	Description
reqbuf_read	Input	1	A signal indicating that the item at the head of the request buffer should be read.
respbuf_read	Input	1	A signal indicating that the item at the head of the response buffer should be read.
reqbuf_empty	Output	1	A signal indicating that the request buffer is empty.
reqbuf_full	Output	1	A signal indicating that the request buffer is full.
reqbuf_valid	Output	1	A signal indicating that the entry in the request buffer is valid.
respbuf_empty	Output	1	A signal indicating that the response buffer is empty.
respbuf_full	Output	1	A signal indicating that the response buffer is full.
respbuf_valid	Output	1	A signal indicating that the entry in the response buffer is valid.
reqbuf_data	Output	BUF_WIDTH	The data at the head of the request buffer.
respbuf_data	Output	BUF_WIDTH	The data at the head of the response buffer.

Table 4.85: Message Handler Module Sub-Modules

Sub-Module	Description
fifo	Creates a first-in-first-out queue to serve as a buffer.

4.3.2 L1 Cache

The **L1_cache** provides data and instruction memory on a per core basis for each core. These caches are connected to higher level caches and/or the main memory by buses. If the system involves a NoC, the L1 cache may be connected to that NoC either through the provided interface through additional layers of caches.

4.3.2.1 L1 Bus Interface Module

The **L1_bus_interface** module connects the L1 cache to the bus interface. This allows it to communicate with other caches, the main memory, and/or a NoC, if it is being used.

Table 4.86: L1 Bus Interface Module Parameters

Parameters	Description
CACHE_OFFSET_BITS	The maximum number of bits in the offset from the cache side.
BUS_OFFSET_BITS	The width of the bus in bits.
DATA_WIDTH	The number of bits in the core data path. Except for highly customized and experimental designs, this will be equivalent to the RISC-V XLEN parameter. Expected values are 32 or 64.
ADDRESS_WIDTH	The number of bits in the address.
MSG_BITS	The number of bits in a message on the bus.
MAX_OFFSET_BITS	The maximum number of offset bits.

Table 4.87: L1 Bus Interface Module Ports

Port	Direction	Width	Description
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	Synchronous, active high reset signal.
cache_offset	Input	log2(CACHE_OFFSET_BITS)	The value representing the cache offset.
cache_msg_in	Input	MSG_BITS	The message in from the cache.
cache_address_in	Input	ADDRESS_BITS	The address in from the cache.
cache_data_in	Input	CACHE_WIDTH	The data in from the cache.
cache_msg_out	Output	MSG_BITS	The message out to the cache.
cache_address_out	Output	ADDRESS_WIDTH	The address out to the cache.
cache_data_out	Output	CACHE_WIDTH	The data out to the cache.
snoop_msg_in	Input	MSG_BITS	The message in from the snooper.
snoop_address_in	Input	ADDRESS_BITS	The address in from the snooper.
snoop_data_in	Input	CACHE_WIDTH	The data in from the snooper.
snoop_msg_out	Output	MSG_BITS	The message out to the snooper.
snoop_address_out	Output	ADDRESS_WIDTH	The address out to the snooper.
snoop_data_out	Output	CACHE_WIDTH	The data out to the snooper.
bus_msg_in	Input	MSG_BITS	The message in from the bus.
bus_address_in	Input	ADDRESS_BITS	The address in from the bus.
bus_data_in	Input	CACHE_WIDTH	The data in from the bus.
bus_msg_out	Output	MSG_BITS	The message out to the bus.
bus_address_out	Output	ADDRESS_WIDTH	The address out to the bus.
bus_data_out	Output	CACHE_WIDTH	The data out to the bus.
active_offset	Output	log2(MAX_OFFSET_BITS)	The value of the active offset.
req_ready	Input	1	A signal to indicate that there is a request ready on the bus.
bus_master	Input	1	A signal to indicate which cache is serving as bus master.

4.3.2.2 L1 Caching Logic Unit

The **L1_caching_logic** unit provides the cache logic for the data and instruction memory for each core. It also provides interfaces with the core the relevant L1 cache serves and either a bus or the NoC. It instantiates the **cache_controller** unit and the **cache_memory**.

Table 4.88: Cache Memory Unit Parameters

Parameters	Description
STATUS_BITS	The number of bits needed to express the status of the cache line at a cache port.

Table 4.88: Cache Memory Unit Parameters

Parameters	Description
COHERENCE_BITS	The number of bits needed to express the status of the cache line's coherence state.
CACHE_OFFSET_BITS	A figure that determines, along with ADDRESS_BITS and INDEX_BITS, the number of tag bits for the cache memory. It also helps determine the number of WORDS_PER_LINE in the cache.
DATA_WIDTH	The number of bits in the core data path. Except for highly customized and experimental designs, this will be equivalent to the RISC-V XLEN parameter. Expected values are 32 or 64.
NUMBER_OF_WAYS	The number of ways in the cache.
ADDRESS_BITS	The number of bits in the address.
INDEX_BITS	The number of bits in the memory address index.
MSG_BITS	The number of bits in a message.
REPLACEMENT_MODE	A one-bit parameter to select the replacement mode; 0 selects random replacement and 1 selects LRU replacement.
COHERENCE_PROTOCOL	The protocol followed by the coherence controller. Uses the MESI protocol.
CORE	A unique number to identify the core this module is in.
CACHE_NO	A unique number to identify the current cache.
CACHE_WORDS	The number of words in a cache line.
CACHE_WIDTH	The number of bits in a cache line.
TAG_BITS	The numbrer of bits used to tag cache entries.
WAY_BITS	The number of bits needed to index the ways in the cache.
SBITS	The number of bits in each cache entry's metadata.

Table 4.89: L1 Caching Logic Module Ports

Port	Direction	Width	Description
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	Synchronous, active high reset signal.
read	Input	1	A signal to enable a read from the cache.
write	Input	1	A signal to enable a write to the cache.
invalidate	Input	1	A signal that invalidates the current cache line.
flush	Input	1	A signal to enable a flush of the cache.
w_byte_en	Input	DATA_WIDTH/8	Allows byte-addressable writes to the cache.
address	Input	ADDRESS_BITS	The address to read from or write to.
data_in	Input	DATA_WIDTH	Data to be written to the cache.
report	Input	1	NEEDS TO BE REPLACED WITH SCAN
data_out	Output	DATA_WIDTH	Data read from the cache.

Table 4.89: L1 Caching Logic Module Ports

Port	Direction	Width	Description
out_address	Output	ADDRESS_BITS	The address sent out to the core.
ready	Output	1	A signal to indicate the output is ready.
valid	Output	1	A signal to indicate the output is vaild.
port1_read	Input	1	A signal to the coherence controller indicating a read from port1.
port1_write	Input	1	A signal to the coherence controller indicating a write to port1.
port1_invalidate	Input	1	A signal that invalidates the cache entry at port1.
port1_index	Input	INDEX_BITS	The index of the cache entry at port1.
port1_tag	Input	TAG_BITS	The tag of the entry at port1.
port1_metadata	Input	SBITS	The metadata of the entry at port1.
port1_data_in	Input	CACHE_WIDTH	The data of the entry at port1.
port1_way_select	Input	WAY_BITS	The way selected for the cache entry at port1.
port1_data_out	Output	CACHE_WIDTH	The data sent out of port1.
port1_matched_way	Output	WAY_BITS	A signal to indicate that the selected way is the correct one.
port1_coh_bits	Output	COHERENCE_BITS	???
port1_status_bits	Output	STATUS_BITS	The status bits for the cache entry at port1.
port1_hit	Output	1	A signal indicating a hit on the cache entry at port1.
mem2cache_msg	Input	MSG_BITS	The message from the memory to the cache.
mem2cache_data	Input	CACHE_WIDTH	The data sent from the memory to the cache.
mem2cache_address	Input	ADDRESS_BITS	The address sent from the memory to the cache.
cache2mem_msg	Output	MSG_BITS	The message from the cache to the memory.
cache2mem_data	Output	CACHE_WIDTH	The data sent from the cache to the memory.
cache2mem_address	Output	ADDRESS_BITS	The address sent from the cache to the memory.
i_reset	Output	1	A reset signal for the message on the bus. ???

Table 4.90: L1 Caching Logic Unit Sub-Modules

Sub-Module	Description
cache_controller	Provides bus master serves to the L1 cache.
cache_memory	Creates the cache memory.

4.3.2.3 L1 Cache Bus Wrapper Module

The **L1cache_bus_wrapper** module wraps the **cache_bus_interface** and **L1_caching_logic** modules into one aggregate module.

Table 4.91: L1 Cache Bus Wrapper Module Parameters

Parameters	**Description**
STATUS_BITS	The number of bits needed to express the status of the cache line at a cache port.
COHERENCE_BITS	The number of bits needed to express the status of the cache line's coherence state.
CACHE_OFFSET_BITS	A figure that determines, along with ADDRESS_BITS and INDEX_BITS, the number of tag bits for the cache memory. It also helps determine the number of WORDS_PER_LINE in the cache.
DATA_WIDTH	The number of bits in the core data path. Except for highly customized and experimental designs, this will be equivalent to the RISC-V XLEN parameter. Expected values are 32 or 64.
NUMBER_OF_WAYS	The number of ways in the cache.
ADDRESS_BITS	The number of bits in the address.
INDEX_BITS	The number of bits in the memory address index.
MSG_BITS	The number of bits in a message.
BUS_OFFSET_BITS	The number of offset btis for the bus.
MAX_OFFSET_BITS	The maximum number of total offset bits.
REPLACEMENT_MODE	A one-bit parameter to select the replacement mode; 0 selects random replacement and 1 selects LRU replacement.
COHERENCE_PROTOCOL	The protocol followed by the coherence controller. Uses the MESI protocol.
CORE	A unique number to identify the core this module is in.
CACHE_NO	A unique number to identify the current cache.
CACHE_WORDS	The number of words in a cache line.
BUS_WORDS	The number of words on the bus.
CACHE_WIDTH	The number of bits in a cache line.
BUS_WIDTH	The number of bits on the bus.
TAG_BITS	The numbrer of bits used to tag cache entries.
WAY_BITS	The number of bits needed to index the ways in the cache.
SBITS	The number of bits in each cache entry's metadata.

Table 4.92: L1 Cache Bus Wrapper Module Ports

Port	**Direction**	**Width**	**Description**
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	Synchronous, active high reset signal.
read	Input	1	A signal to enable a read from the cache.
write	Input	1	A signal to enable a write to the cache.
invalidate	Input	1	A signal that invalidates the current cache line.
flush	Input	1	A signal to enable a flush of the cache.

Table 4.92: L1 Cache Bus Wrapper Module Ports

Port	Direction	Width	Description
w_byte_en	Input	DATA_WIDTH/8	Allows byte-addressable writes to the cache.
address	Input	ADDRESS_BITS	The address to read from or write to.
data_in	Input	DATA_WIDTH	Data to be written to the cache.
report	Input	1	NEEDS TO BE REPLACED WITH SCAN
data_out	Output	DATA_WIDTH	Data read from the cache.
out_address	Output	ADDRESS_BITS	The address sent out to the core.
ready	Output	1	A signal to indicate the output is ready.
valid	Output	1	A signal to indicate the output is vaild.
bus_msg_in	Input	MSG_BITS	The message entering the bus.
bus_address_in	Input	ADDRESS_BITS	The address entering the bus.
bus_data_in	Input	DATA_WIDTH	The data entering the bus.
bus_master	Input	1	A signal that indicates which module controls the bus.
req_ready	Input	1	A signal to indicate that a request is ready on the bus.
curr_offset	Input	log2(MAX_OFFSET_BITS)	The current offset on the bus.
bus_msg_out	Output	MSG_BITS	The message exiting the bus.
bus_address_out	Output	ADDRESS_BITS	The address exiting the bus.
bus_data_out	Output	BUS_WIDTH	The data exiting the bus.
active_offset	Output	log2(MAX_OFFSET_BITS)	The active offset bits on the bus.

Table 4.93: L1 Cache Bus Wrapper Sub-Modules

Sub-Module	Description
L1_caching_logic	Provides the cache logic for the data and instruction memory for each core.
cache_bus_interface	Provides the basic bus interface connecting the cache to a shared bus.

4.3.2.4 Cache Bus Interface Module

The **cache_bus_interface** module provides the basic bus interface connecting the cache to a shared bus. It has two interfaces: one with the cache and one with the shared bus. It instantiates the **snooper** unit and the **bus_interface** module.

Table 4.94: Cache Bus Interface Module Parameters

Parameters	Description
STATUS_BITS	The number of bits needed to express the status of the cache line at a cache port.
COHERENCE_BITS	The number of bits needed to express the status of the cache line's coherence state.

Table 4.94: Cache Bus Interface Module Parameters

Parameters	**Description**
CACHE_OFFSET_BITS	A figure that determines, along with ADDRESS_BITS and INDEX_BITS, the number of tag bits for the cache memory. It also helps determine the number of WORDS_PER_LINE in the cache.
DATA_WIDTH	The number of bits in the core data path. Except for highly customized and experimental designs, this will be equivalent to the RISC-V XLEN parameter. Expected values are 32 or 64.
NUMBER_OF_WAYS	The number of ways in the cache.
ADDRESS_BITS	The number of bits in the address.
INDEX_BITS	The number of bits in the memory address index.
MSG_BITS	The number of bits in a message.
BUS_OFFSET_BITS	The number of offset btis for the bus.
MAX_OFFSET_BITS	The maximum number of total offset bits.
REPLACEMENT_MODE	A one-bit parameter to select the replacement mode; 0 selects random replacement and 1 selects LRU replacement.
COHERENCE_PROTOCOL	The protocol followed by the coherence controller. Uses the MESI protocol.
CORE	A unique number to identify the core this module is in.
CACHE_NO	A unique number to identify the current cache.
CACHE_WORDS	The number of words in a cache line.
BUS_WORDS	The number of words on the bus.
CACHE_WIDTH	The number of bits in a cache line.
BUS_WIDTH	The number of bits on the bus.
TAG_BITS	The numbrer of bits used to tag cache entries.
WAY_BITS	The number of bits needed to index the ways in the cache.
SBITS	The number of bits in each cache entry's metadata.

Table 4.95: Cache Bus Interface Module Ports

Port	**Direction**	**Width**	**Description**
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	Synchronous, active high reset signal.
bus_msg_in	Input	MSG_BITS	The message entering the bus.
bus_address_in	Input	ADDRESS_BITS	The address entering the bus.
bus_data_in	Input	CACHE_WIDTH	The data entering the bus.
bus_master	Input	1	A signal that indicates which module controls the bus.
req_ready	Input	1	A signal to indicate that a request is ready on the bus.
curr_offset	Input	log2(MAX_OFFSET_BITS)	The current offset on the bus.

Table 4.95: Cache Bus Interface Module Ports

Port	Direction	Width	Description
bus_msg_out	Output	MSG_BITS	The message exiting the bus.
bus_address_out	Output	ADDRESS_BITS	The address exiting the bus.
bus_data_out	Output	BUS_WIDTH	The data exiting the bus.
active_offset	Output	log2(MAX_OFFSET_BITS)	The active offset bits on the bus.
cache_msg_in	Input	MSG_BITS	The message in from the cache.
cache_address_in	Input	ADDRESS_BITS	The address in from the cache.
cache_data_in	Input	CACHE_WIDTH	The data in from the cache.
i_reset	Input	1	A reset signal for the message on the bus. ???
cache_msg_out	Output	MSG_BITS	The message out to the cache.
cache_address_out	Output	ADDRESS_BITS	The address out to the cache.
cache_data_out	Output	CACHE_WIDTH	The data out to the cache.
port1_read_data	Input	CACHE_WIDTH	Data read from the cache memory.
port1_matched_way	Input	WAY_BITS	A signal to indicate that the selected way is the correct one.
port1_coh_bits	Input	COHERENCE_BITS	???
port1_status_bits	Input	STATUS_BITS	The status bits for the cache entry.
port1_hit	Input	1	A signal indicating a hit on the cache entry.
port1_read	Output	1	A sginal indicating a read through port1.
port1_write	Output	1	A signal indicating a write through port1.
port1_invalidate	Output	1	A signal that invalidates the cache entry at port1.
port1_index	Output	INDEX_BITS	The index of the cache entry at port1.
port1_tag	Output	TAG_BITS	The tag of the cache entry at port1.
port1_metadata	Output	SBITS	The metadata of the cache entry at port1.
port1_write_data	Output	CACHE_WIDTH	The data to be written from the cache entry at port1.
port1_way_select	Output	WAY_BITS	The cache way selected for the cache entry at port1.

Table 4.96: Cache Bus Interface Module Sub-Modules

Sub-Module	Description
snooper	The snooper monitors which cacke blocks are in use and where.
L1_bus_interface	The L1_bus_interface module connects the L1 cache to the bus interface.

4.3.2.5 Cache Controller Module

The **cache_controller** module provides bus master services to the L1 caches. It provides interfaces with the **cache_memory** and **bus_interface** modules. It also takes input from the **snooper** unit. It also provides the ability to flush the caches to which it connects, as well as logic to arbitrate which request has priority and which will be stalled.

Table 4.97: Cache Controller Module Parameters

Parameters	Description
STATUS_BITS	The number of bits needed to express the status of the cache line at a cache port.
COHERENCE_BITS	The number of bits needed to express the status of the cache line's coherence state.
OFFSET_BITS	A figure that determines, along with ADDRESS_BITS and INDEX_BITS, the number of tag bits for the cache memory. It also helps determine the number of WORDS_PER_LINE in the cache.
DATA_WIDTH	The number of bits in the core data path. Except for highly customized and experimental designs, this will be equivalent to the RISC-V XLEN parameter. Expected values are 32 or 64.
NUMBER_OF_WAYS	The number of ways in the cache.
ADDRESS_BITS	The number of bits in the address.
INDEX_BITS	The number of bits in the memory address index.
MSG_BITS	The number of bits in a message.
CORE	A unique number to identify the core this module is in.
CACHE_NO	A unique number to identify the current cache.

Table 4.98: Cache Controller Module Ports

Port	Direction	Width	Description
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	Synchronous, active high reset signal.
read	Input	1	A signal to enable a read from the cache.
write	Input	1	A signal to enable a write to the cache.
invalidate	Input	1	A signal that invalidates the current cache line.
flush	Input	1	A signal to enable a flush of the cache.
w_byte_en	Input	DATA_WIDTH/8	Allows byte-addressable writes to the cache.
address	Input	ADDRESS_BITS	The address to read from or write to.
data_in	Input	DATA_WIDTH	Data to be written to the cache.
report	Input	1	NEEDS TO BE REPLACED WITH SCAN
data_out	Output	DATA_WIDTH	Data read from the cache.
out_address	Output	ADDRESS_BITS	The address sent out to the core.
ready	Output	1	A signal to indicate the output is ready.
valid	Output	1	A signal to indicate the output is vaild.
read0	Input	1	A signal enabling a read from port0.

Table 4.98: Cache Controller Module Ports

Port	Direction	Width	Description
write0	Input	1	A signal enabling a write to port0.
invalidate0	Input	1	A signal that invalidates the cache entry at port0.
index0	Input	INDEX_BITS	The index of the cache entry at port0.
tag0	Input	TAG_BITS	The tag of the cache entry at port0.
meta_data0	Input	SBITS	The metadata of the cache entry at port0.
data_in0	Input	BLOCK_WIDTH	The data to be written in the cache entry at port0.
tag_in0	Input	TAG_BITS	The tag of the cache line coming in to port0.
way_select0	Output	WAY_BITS	The way selected for the cache entry at port0.
data_out0	Output	BLOCK_WIDTH	The data to be read from the cache entry at port0.
matched_way0	Input	WAY_BITS	A signal to indicate that the selected way is the correct one.
coh_bits0	Input	COH_BITS	???
status_bits0	Input	STATUS_BITS	The status bits for the cache line at port0.
hit0	Input	1	A signal indicating a hit on the cache line at port0.
read1	Output	1	A signal enabling a read from port1.
write1	Output	1	A signal enabling a write to port1.
invalidate1	Output	1	A signal that invalidates the cache entry at port1.
index1	Output	INDEX_BITS	The index of the cache entry at port1.
tag1	Output	TAG_BITS	The tag of the cache entry at port1.
meta_data1	Output	SBITS	The metadata of the cache entry at port1.
way_select1	Input	WAY_BITS	The way selected for the cache entry at port1.
data_out1	Output	BLOCK_WIDTH	The data to be read from the cache entry at port1.
i_reset	Output	1	A reset signal for the message on the bus.
mem2cache_msg	Input	MSG_BITS	The message from the memory to the cache.
mem2cache_data	Input	CACHE_WIDTH	The data sent from the memory to the cache.
mem2cache_address	Input	ADDRESS_BITS	The address sent from the memory to the cache.
cache2mem_msg	Output	MSG_BITS	The message from the cache to the memory.
cache2mem_data	Output	CACHE_WIDTH	The data sent from the cache to the memory.
cache2mem_address	Output	ADDRESS_BITS	The address sent from the cache to the memory.
snoop_address	Input	ADDRESS_BITS	The address from the snooper.
snoop_read	Input	1	A signal indicating that the snooper is reading data.
snoop_modify	Input	1	A signal indicating that the snooper is modifying data.

4.3.2.6 Cache NoC Interface Module

The **cache_noc_interface** module connects the caches to a NoC. It provides two interfaces: one to the cache (which interfaces with the **cache_controller** and services memory requests, as well as reading from/writing to the cache

memory for coherence operations) and one to the NoC.

Table 4.99: Cache NoC Interface Module Parameters

Parameters	**Description**
STATUS_BITS	The number of bits needed to express the status of the cache line at a cache port.
COHERENCE_BITS	The number of bits needed to express the status of the cache line's coherence state.
CACHE_OFFSET_BITS	A figure that determines, along with ADDRESS_BITS and INDEX_BITS, the number of tag bits for the cache memory. It also helps determine the number of WORDS_PER_LINE in the cache.
DATA_WIDTH	The number of bits in the core data path. Except for highly customized and experimental designs, this will be equivalent to the RISC-V XLEN parameter. Expected values are 32 or 64.
NUMBER_OF_WAYS	The number of ways in the cache.
ADDRESS_BITS	The number of bits in the address.
INDEX_BITS	The number of bits in the memory address index.
MSG_BITS	The number of bits in a message.
MAX_OFFSET_BITS	The maximum number of total offset bits.
REQ_BUF_DEPTH_BITS	The depth of the request buffer in bits. Equal to log2(request buffer depth).
RESP_BUF_DEPTH_BITS	The depth of the response buffer in bits. Equal to log2(response buffer depth).
CORE	A unique number to identify the core this module is in.
CACHE_NO	A unique number to identify the current cache.
CONTROLLER_TYPE	Specifies whether the controller type is blocking or non-blocking. The former will hold the signals to the interface asserted until the interface responds. The latter will deassert the signals after one cycle unless the interface is busy, in which case the interface will not register the message from the controller.
ID_BITS	The number of bits in the ID of sources.
DEFAULT_DEST	Assigns 0 as the default destination.
CACHE_WORDS	The number of words in a cache line.
CACHE_WIDTH	The number of bits in a cache line.
TAG_BITS	The numbrer of bits used to tag cache entries.
WAY_BITS	The number of bits needed to index the ways in the cache.
SBITS	The number of bits in each cache entry's metadata.
BUF_WIDTH	The number of bits in the buffer width.

Table 4.100: Cache NoC Interface Module Ports

Port	**Direction**	**Width**	**Description**
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	Synchronous, active high reset signal.
noc_msg_in	Input	MSG_BITS	The message entering the NoC.

Table 4.100: Cache NoC Interface Module Ports

Port	Direction	Width	Description
noc_address_in	Input	ADDRESS_BITS	The address entering the NoC.
noc_data_in	Input	CACHE_WIDTH	The data entering the NoC.
noc_src_id	Input	ID_BITS	The ID of the NoC source.
packetizer_busy	Input	1	A signal that indicates when the packetizer is busy.
noc_msg_out	Output	MSG_BITS	The message exiting the NoC.
noc_address_out	Output	ADDRESS_BITS	The address exiting the NoC.
noc_data_out	Output	BUS_WIDTH	The data exiting the NoC.
noc_dest_id	Output	ID_BITS	The ID of the NoC destination.
cache_msg_in	Input	MSG_BITS	The message in from the cache.
cache_address_in	Input	ADDRESS_BITS	The address in from the cache.
cache_data_in	Input	CACHE_WIDTH	The data in from the cache.
i_reset	Input	1	A reset signal for the message on the bus. ???
cache_msg_out	Output	MSG_BITS	The message out to the cache.
cache_address_out	Output	ADDRESS_BITS	The address out to the cache.
cache_data_out	Output	CACHE_WIDTH	The data out to the cache.
busy	Output	1	A signal to indicate that the NoC is busy to the cache controller.
port1_read_data	Input	CACHE_WIDTH	Data read from the cache memory.
port1_matched_way	Input	WAY_BITS	A signal to indicate that the selected way is the correct one.
port1_coh_bits	Input	COHERENCE_BITS	???
port1_status_bits	Input	STATUS_BITS	The status bits for the cache entry.
port1_hit	Input	1	A signal indicating a hit on the cache entry.
port1_read	Output	1	A sginal indicating a read through port1.
port1_write	Output	1	A signal indicating a write through port1.
port1_invalidate	Output	1	A signal that invalidates the cache entry at port1.
port1_index	Output	INDEX_BITS	The index of the cache entry at port1.
port1_tag	Output	TAG_BITS	The tag of the cache entry at port1.
port1_metadata	Output	SBITS	The metadata of the cache entry at port1.
port1_write_data	Output	CACHE_WIDTH	The data to be written from the cache entry at port1.
port1_way_select	Output	WAY_BITS	The cache way selected for the cache entry at port1.

Table 4.101: Cache NoC Interface Sub-Modules

Sub-Module	Description
message_handler	The message handler receives messages from the NoC and buffers them. It controls the outgoing messages to the NoC. It multiplexes between messages from the controller and any Nack messages from the handler itself.

4.3.2.7 Snooper Module

The **snooper** module monitors which cache blocks are in use and where. It interfaces with the **cache_memory** module, the **L1_bus_interface** module, and the shared bus.

Table 4.102: Snooper Module Parameters

Parameters	Description
CACHE_OFFSET_BITS	The maximum number of offset bits from the cache side.
BUS_OFFSET_BITS	The number of bits determining the width of the bus.
DATA_WIDTH	The number of bits in the core data path. Except for highly customized and experimental designs, this will be equivalent to the RISC-V XLEN parameter. Expected values are 32 or 64.
ADDRESS_WIDTH	The number of bits in the address.
MSG_BITS	The number of bits in a message on the bus.
INDEX_BITS	The maximum number of index bits for the cache.
COHERENCE_BITS	The number of bits needed to express the status of the cache line's coherence state.
STATUS_BITS	The number of bits needed to express the status of the cache line.
NUMBER_OF_WAYS	The number of ways in the cache.
MAX_OFFSET_BITS	The size in bits of the largest offset.

Table 4.103: Snooper Module Ports

Port	Direction	Width	Description
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	Synchronous, active high reset signal.
data_in	Input	CACHE_WIDTH	The data entering the bus from the cache memory.
matched_way	Input	WAY_BITS	The selected cache way.
coh_bits	Input	COHERENCE_BITS	???
status_bits	Input	STATUS_BITS	The status bits for the cache line on the bus.
hit	Input	1	A signal indicating a hit on a requested cache line.
read	Output	1	A signal enabling the data on the bus to be read.
write	Output	1	A signal enabling data to be written to the bus.
invalidate	Output	1	A signal that invalidates the cache line on the bus.
index	Output	INDEX_BITS	The index of the requested cache line.

Table 4.103: Snooper Module Ports

Port	Direction	Width	Description
tag	Output	TAG_BITS	The tag of the requested cache line.
meta_data	Output	SBITS	The metadata associated with the requested cache line.
data_out	Output	CACHE_WIDTH	The data to be written to the cache memory from the bus.
way_select	Output	WAY_BITS	The way selected for the requested cache line.
intf_msg	Input	MSG_BITS	The message on the L1 bus interface.
intf_address	Input	ADDRESS_BITS	The address of the cache line on the L1 bus interface.
int_data	Input	CACHE_WIDTH	The data stored in the cache line on the L1 bus interface.
snoop_msg	Output	MSG_BITS	The message from the L1 bus interface to either the cache memory or the shared bus.
snoop_address	Output	ADDRESS_WIDTH	The address of the cache line transferred from the L1 bus interface to the cache memory.
snoop_data	Output	CACHE_WIDTH	The data stored in the cache line transferred from the L1 bus interface to the cache memory.
bus_msg	Input	MSG_BITS	The message on the shared bus.
bus_address	Input	ADDRESS_WIDTH	The address transmitted on the shared bus.
req_ready	Input	1	A signal to indicate that there is a request ready on the shared bus.
bus_master	Input	1	A signal to indicate which coherence controller is master of the bus.
curr_offset	Input	log2(MAX_OFFSET_BITS)	The current offset.

4.3.3 Lx Cache

The Lx cache modules provide support and connections for lower level caches. They connect to the L1 cache (or the cache immediately above them in the hierarchy for caches with several layers) and either the main memory, the cache below them in the hierarchy, or the NoC, as appropriate. Their logic echoes that found in the analogous modules of the L1 cache.

4.3.3.1 Lx Cache Controller Module

The **Lxcache_controller** module provides cache control for an Lx cache (x = 2, 3, ...). It handles one request at a time and communicates with two interfaces. One the processor side, the interface is usually with a bus, although it could be connected to a directory controller if that directory is associated with a secondary cache instead of a main memory. On the memory side, the interface is with a bus or a NoC, but the controller should be agnostic to the connected interface.

At present, the controller is conservative, handling requests by blocking until it receives a response to a request from the interface with which it is currently communicating. The only exception to this behavior comes when the **noc_interface** issues a REQ_FLUSH or FwdGetS request while the controller is waiting for a response from the NoC interface.

Table 4.104: Lx Cache Controller Module Parameters

Parameters	Description
STATUS_BITS	The number of bits needed to express the status of the cache line at a cache port. Here includes the valid, dirty, and include bits. The last will be 0 if the controller is not tracking inclusion.
INCLUSION	A signal to indicate whether the cache tracks inclusion. When high, it tracks whether a cache line is cached in higher levels. When low, it is agnostic to whether a cache line is cached in upper levels.
COHERENCE_BITS	The number of bits needed to express the status of the cache line's coherence state.
OFFSET_BITS	A figure that determines, along with ADDRESS_BITS and INDEX_BITS, the number of tag bits for the cache memory. It also helps determine the number of CACHE_WORDS in a cache line.
DATA_WIDTH	The number of bits in the core data path. Except for highly customized and experimental designs, this will be equivalent to the RISC-V XLEN parameter. Expected values are 32 or 64.
NUMBER_OF_WAYS	The number of ways in the cache.
ADDRESS_BITS	The number of bits in the address.
INDEX_BITS	The number of bits in the memory address index.
MSG_BITS	The number of bits in a message.
LAST_LEVEL	Indicates whether this cache is the last level of the coherence domain. If high, silent evictions and upgrades from S to M are allowed. If low, the cache informs the directory or the shared lower cache below writing to the shared lines (as well as evictions if MEM_SIDE = "DIR").
MEM_SIDE	Indicates the type of coherence mechanism on the memory side. DIR indicates directory-based coherence and SNOOP indicates snooping on a shared bus. At present, it is set to DIR. It should not be changed unless you understand the memory subsystem latencies clearly.
REISSUE_COUNT	The maximum number of times a blocked request can be reissued. ???
CACHE_WORDS	The number of words in one cache line.
CACHE_WIDTH	The width of a cache line.
MBITS	The sum of the coherence bits and the status bits.
TAG_BITS	The number of bits in the tag for each cache line.
WAY_BITS	The number of bits needed to represent the number of ways in the cache.

Table 4.105: Lx Cache Controller Module Ports

Port	Direction	Width	Description
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	Synchronous, active high reset signal.
address	Input	ADDRESS_BITS	The address to read from or write to.
data_in	Input	DATA_WIDTH	Data to be written to the cache.
msg_in	Input	MSG_BITS	The message entering the controller.

Table 4.105: Lx Cache Controller Module Ports

Port	Direction	Width	Description
pending_requests	Input	1	An unused signal added to preserve the interface.
data_out	Output	DATA_WIDTH	Data read from the cache.
out_address	Output	ADDRESS_BITS	The address sent out to the core.
msg_out	Output	MSG_BITS	The message leaving the contoller.
read0	Output	1	A signal enabling a read from port0.
write0	Output	1	A signal enabling a write to port0.
invalidate0	Output	1	A signal that invalidates the cache entry at port0.
index0	Output	INDEX_BITS	The index of the cache entry at port0.
tag0	Output	TAG_BITS	The tag of the cache entry at port0.
meta_data0	Output	SBITS	The metadata of the cache entry at port0.
data_in0	Input	CACHE_WIDTH	The data to be written in the cache entry at port0.
tag_in0	Input	TAG_BITS	The tag of the cache line coming in to port0.
way_select0	Output	WAY_BITS	The way selected for the cache entry at port0.
data0	Output	CACHE_WIDTH	The data to be read from the cache entry at port0.
matched_way0	Input	WAY_BITS	A signal to indicate that the selected way is the correct one.
coh_bits0	Input	COH_BITS	???
status_bits0	Input	STATUS_BITS	The status bits for the cache line at port0.
hit0	Input	1	A signal indicating a hit on the cache line at port0.
i_reset	Output	1	A reset signal for the message on the bus.
mem2cache_msg	Input	MSG_BITS	The message from the memory to the cache.
mem2cache_data	Input	CACHE_WIDTH	The data sent from the memory to the cache.
mem2cache_address	Input	ADDRESS_BITS	The address sent from the memory to the cache.
mem_intf_busy	Input	1	A signal to indicate that the memory interface is busy.
mem_intf_address	Input	ADDRESS_BITS	The address in the memory interface.
mem_intf_address_valid	Input	1	A signal to indicate whether the address in the memory interface is valid.
cache2mem_msg	Output	MSG_BITS	The message from the cache to the memory.
cache2mem_data	Output	CACHE_WIDTH	The data sent from the cache to the memory.
cache2mem_address	Output	ADDRESS_BITS	The address sent from the cache to the memory.
scan	Input	1	A signal used for testing.

4.3.3.2 Lx Cache Wrapper Module

The **Lxcache_wrapper** module wraps the **Lxcache_controller**, the **lx_bus_interface** and a **cache_memory** modules into one aggregate module. The **cache_memory** is described elsewhere.

Table 4.106: Lx Cache Wrapper Module Parameters

Parameters	Description
STATUS_BITS	The number of bits needed to express the status of the cache line at a cache port.
INCLUSION	Determines whether the cache tracks inclusion or not. When high, tracks whether a cache line is cached in upper levels; when low, agnostic to whether a line is cached in upper levels.
COHERENCE_BITS	The number of bits needed to express the status of the cache line's coherence state.
CACHE_OFFSET_BITS	A figure that determines, along with ADDRESS_BITS and INDEX_BITS, the number of tag bits for the cache memory. It also helps determine the number of CACHE_WORDS in the cache.
DATA_WIDTH	The number of bits in the core data path. Except for highly customized and experimental designs, this will be equivalent to the RISC-V XLEN parameter. Expected values are 32 or 64.
NUMBER_OF_WAYS	The number of ways in the cache.
ADDRESS_BITS	The number of bits in the address.
INDEX_BITS	The number of bits in the memory address index.
MSG_BITS	The number of bits in a message.
BUS_OFFSET_BITS	The number of offset btis for the bus.
MAX_OFFSET_BITS	The maximum number of total offset bits.
REPLACEMENT_MODE	A one-bit parameter to select the replacement mode; 0 selects random replacement and 1 selects LRU replacement.
LAST_LEVEL	Indicates whether this cache is the last level of the coherence domain. If so, silent evictions and upgrades from S to M are allowed. If not, the cache should inform the directory or the shared cache below writing to shared lines (and evictions if MEM_SIDE = "DIR").
MEM_SIDE	Inidcates the type of coherence mechanism on the memory side. At present, set to "DIR", which should not be changed unless you understand the memory subsystem latencies clearly. The other possible setting is "SNOOP".
CACHE_WORDS	The number of words in a cache line.
BUS_WORDS	The number of words on the bus.
CACHE_WIDTH	The number of bits in a cache line.
BUS_WIDTH	The number of bits on the bus.
TAG_BITS	The numbrer of bits used to tag cache entries.
WAY_BITS	The number of bits needed to index the ways in the cache.
SBITS	The number of bits in each cache entry's metadata.

Table 4.107: Lx Cache Wrapper Module Ports

Port	Direction	Width	Description
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	Synchronous, active high reset signal.

Table 4.107: Lx Cache Wrapper Module Ports

Port	Direction	Width	Description
bus_msg_in	Input	MSG_BITS	The message entering the bus.
bus_address_in	Input	ADDRESS_BITS	The address entering the bus.
bus_data_in	Input	BUS_WIDTH	The data entering the bus.
bus_master	Input	1	A signal that indicates which module controls the bus.
req_ready	Input	1	A signal to indicate that a request is ready on the bus.
req_offset	Input	log2(MAX_OFFSET_BITS)	The offset for the request on the bus.
bus_msg_out	Output	MSG_BITS	The message exiting the bus.
bus_address_out	Output	ADDRESS_BITS	The address exiting the bus.
bus_data_out	Output	BUS_WIDTH	The data exiting the bus.
active_offset	Output	log2(MAX_OFFSET_BITS)	The active offset bits on the bus.
mem2cache_msg	Input	MSG_BITS	The message from the memory to the cache.
mem2cache_address	Input	ADDRESS_BITS	The address sent from the memory to the cache.
mem2cache_data	Input	CACHE_WIDTH	The data sent from the memory to the cache.
mem_intf_busy	Input	1	A signal to indicate that the memory interface is busy.
mem_intf_address	Input	ADDRESS_BITS	The address in the memory interface.
mem_intf_address_valid	Input	1	A signal to indicate whether the address in the memory interface is valid.
cache2mem_msg	Output	MSG_BITS	The message from the cache to the memory.
cache2mem_address	Output	ADDRESS_BITS	The address sent from the cache to the memory.
cache2mem_data	Output	CACHE_WIDTH	The data sent from the cache to the memory.
port1_read	Input	1	A signal to enable a read from the cache.
port1_write	Input	1	A signal to enable a write to the cache.
port1_invalidate	Input	1	A signal that invalidates the current cache line.
port1_index	Input	INDEX_BITS	The index of the cache entry.
port1_tag	Input	TAG_BITS	The tag of the cache entry.
port1_metadata	Input	SBITS	The metadata of the cache entry.
port1_write_data	Input	CACHE_WIDTH	The data to be written to the cache.
port1_way_select	Input	WAY_SELECT	The way selected for the cache entry.

Table 4.107: Lx Cache Wrapper Module Ports

Port	Direction	Width	Description
port1_read_data	Output	CACHE_WIDTH	The data to be read from the cache entry.
port1_matched_way	Output	WAY_BITS	A signal to inidcate that the selected way is the correct one.
port1_coh_bits	Output	COHERENCE_BITS	???
port1_status_bits	Output	STATUS_BITS	The status bits for the cache entry.
port1_hit	Output	1	A signal indicating a hit on the cache line.
scan	Input	1	A signal used for testing.

Table 4.108: Lx Cache Wrapper Sub-Modules

Sub-Module	Description
Lxcache_controller	Provides cache control for the Lx cache (x = 2, 3, ...).
lx_bus_interface	Provides the basic bus interface connecting the cache to a shared bus.
cache_memory	Provides the cache memory.

4.3.3.3 Lx Bus Interface Module

The **lx_bus_interface** module connects the Lx cache to the bus interface. This allows it to communicate with other caches, the main memory, and/or a NoC, if it is being used.

Table 4.109: Lx Bus Interface Module Parameters

Parameters	Description
CACHE_OFFSET_BITS	The maximum number of bits in the offset from the cache side.
BUS_OFFSET_BITS	The width of the bus in bits.
DATA_WIDTH	The number of bits in the core data path. Except for highly customized and experimental designs, this will be equivalent to the RISC-V XLEN parameter. Expected values are 32 or 64.
ADDRESS_WIDTH	The number of bits in the address.
MSG_BITS	The number of bits in a message on the bus.
MAX_OFFSET_BITS	The maximum number of offset bits.

Table 4.110: Lx Bus Interface Module Ports

Port	Direction	Width	Description
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	Synchronous, active high reset signal.
cache_msg_in	Input	MSG_BITS	The message in from the cache.
cache_address_in	Input	ADDRESS_BITS	The address in from the cache.

Table 4.110: Lx Bus Interface Module Ports

Port	Direction	Width	Description
cache_data_in	Input	CACHE_WIDTH	The data in from the cache.
cache_msg_out	Output	MSG_BITS	The message out to the cache.
cache_address_out	Output	ADDRESS_WIDTH	The address out to the cache.
cache_data_out	Output	CACHE_WIDTH	The data out to the cache.
pending_requests	Output	1	A signal that indicates whether there are requests pending.
bus_msg_in	Input	MSG_BITS	The message in from the bus.
bus_address_in	Input	ADDRESS_BITS	The address in from the bus.
bus_data_in	Input	CACHE_WIDTH	The data in from the bus.
bus_msg_out	Output	MSG_BITS	The message out to the bus.
bus_address_out	Output	ADDRESS_WIDTH	The address out to the bus.
bus_data_out	Output	CACHE_WIDTH	The data out to the bus.
active_offset	Output	log2(MAX_OFFSET_BITS)	The value of the active offset.
req_ready	Input	1	A signal to indicate that there is a request ready on the bus.
req_offset	Input	log2(MAX_OFFSET_BITS)	Helps determine how many words may be in requests.

4.3.3.4 Lx NoC Interface Module

The **lx_noc_interface** module connects the caches to a NoC. It provides two interfaces: one to the cache (which interfaces with the **cache_controller** and services memory requests, as well as reading from/writing to the cache memory for coherence operations) and one to the NoC.

Table 4.111: Lx NoC Interface Module Parameters

Parameters	Description
STATUS_BITS	The number of bits needed to express the status of the cache line at a cache port.
COHERENCE_BITS	The number of bits needed to express the status of the cache line's coherence state.
CACHE_OFFSET_BITS	A figure that determines, along with ADDRESS_BITS and INDEX_BITS, the number of tag bits for the cache memory. It also helps determine the number of WORDS_PER_LINE in the cache.
DATA_WIDTH	The number of bits in the core data path. Except for highly customized and experimental designs, this will be equivalent to the RISC-V XLEN parameter. Expected values are 32 or 64.
NUMBER_OF_WAYS	The number of ways in the cache.
ADDRESS_BITS	The number of bits in the address.
INDEX_BITS	The number of bits in the memory address index.
MSG_BITS	The number of bits in a message.
REQ_BUF_DEPTH_BITS	The number of bits needed to represent the depth of the request buffer. Equal to log2(request buffer depth).

Table 4.111: Lx NoC Interface Module Parameters

Parameters	Description
RESP_BUF_DEPTH_BITS	The number of bits needed to represent the depth of the response buffer. Equal to log2(response buffer depth).
CORE	A unique number to identify the core this module is in.
CACHE_NO	A unique number to identify the current cache.
CONTROLLER_TYPE	Specifies whether the controller type is blocking or non-blocking. The former will hold the signals to the interface asserted until the interface responds. The latter will deassert the signals after one cycle unless the interface is busy, in which case the interface will not register the message from the controller.
ID_BITS	The number of bits in the ID of sources.
DEFAULT_DEST	Assigns 0 as the default destination.
CACHE_WORDS	The number of words in a cache line.
CACHE_WIDTH	The number of bits in a cache line.
TAG_BITS	The numbrer of bits used to tag cache entries.
WAY_BITS	The number of bits needed to index the ways in the cache.
SBITS	The number of bits in each cache entry's metadata.
BUF_WIDTH	The number of bits in the buffer width.

Table 4.112: Lx NoC Interface Module Ports

Port	Direction	Width	Description
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	Synchronous, active high reset signal.
noc_msg_in	Input	MSG_BITS	The message entering the NoC.
noc_address_in	Input	ADDRESS_BITS	The address entering the NoC.
noc_data_in	Input	CACHE_WIDTH	The data entering the NoC.
noc_src_id	Input	ID_BITS	The ID of the NoC source.
packetizer_busy	Input	1	A signal that indicates when the packetizer is busy.
noc_msg_out	Output	MSG_BITS	The message exiting the NoC.
noc_address_out	Output	ADDRESS_BITS	The address exiting the NoC.
noc_data_out	Output	BUS_WIDTH	The data exiting the NoC.
noc_dest_id	Output	ID_BITS	The ID of the NoC destination.
interface_busy	Output	1	A signal to indicate whether the interface is busy.
cache_msg_in	Input	MSG_BITS	The message in from the cache.
cache_address_in	Input	ADDRESS_BITS	The address in from the cache.
cache_data_in	Input	CACHE_WIDTH	The data in from the cache.
cache_msg_out	Output	MSG_BITS	The message out to the cache.
cache_address_out	Output	ADDRESS_BITS	The address out to the cache.

Table 4.112: Lx NoC Interface Module Ports

Port	Direction	Width	Description
cache_data_out	Output	CACHE_WIDTH	The data out to the cache.
busy	Output	1	A signal to indicate that the NoC is busy to the cache controller.
current_address	Output	ADDRESS_BITS	The current address in the cache controller.
current_address_valid	Output	1	A signal to indicate that the current address is valid.
port1_read_data	Input	CACHE_WIDTH	Data read from the cache memory.
port1_matched_way	Input	WAY_BITS	A signal to indicate that the selected way is the correct one.
port1_coh_bits	Input	COHERENCE_BITS	???
port1_status_bits	Input	STATUS_BITS	The status bits for the cache entry.
port1_hit	Input	1	A signal indicating a hit on the cache entry.
port1_read	Output	1	A sginal indicating a read through port1.
port1_write	Output	1	A signal indicating a write through port1.
port1_invalidate	Output	1	A signal that invalidates the cache entry at port1.
port1_index	Output	INDEX_BITS	The index of the cache entry at port1.
port1_tag	Output	TAG_BITS	The tag of the cache entry at port1.
port1_metadata	Output	SBITS	The metadata of the cache entry at port1.
port1_write_data	Output	CACHE_WIDTH	The data to be written from the cache entry at port1.
port1_way_select	Output	WAY_BITS	The cache way selected for the cache entry at port1.

Table 4.113: Lx NoC Interface Sub-Modules

Sub-Module	Description
message_handler	The message handler receives messages from the NoC and buffers them. It controls the outgoing messages to the NoC. It multiplexes between messages from the controller and any Nack messages from the handler itself.

4.3.4 Directory

The Directory includes only the **directory_controller**, which connects NoC to either a shared cache or the main memory while maintaining memory coherence.

4.3.4.1 Directory Controller

The **directory_controller** stores the coherence state for all the cached memory blocks in the directory cache and manages cache coherence. Its memory interface can be connected to either the main memory or a shared cache. Its responds to requests from the caches after checking the coherence state of a memory block (from the directory cache) and getting the memory block through the memory interface. It instantiates the **memory_handler**, a **cache_memory** to serve as the directory cache, a **priority_encoder**, an **arbiter**, and two **one_hot_decoder** modules. These sub-modules are described elsewhere.

Table 4.114: Directory Controller Module Parameters

Parameters	Description
OFFSET_BITS	The number of offset bits for sharer caches.
DATA_WIDTH	The number of bits in the core data path. Except for highly customized and experimental designs, this will be equivalent to the RISC-V XLEN parameter. Expected values are 32 or 64.
ADDRESS_BITS	The number of bits in the address.
MSG_BITS	The number of bits in a message.
REQ_BUF_DEPTH_BITS	The number of bits needed to represent the depth of the request buffer.
RESP_BUF_DEPTH_BITS	The number of bits needed to represent the depth of the response buffer.
NUM_SHARER_BITS	The number of bits needed to represent the number of pointers to sharer caches.
SHARER_ID_BITS	The number of bits needed to identify the sharer caches.
DIR_INDEX_BITS	The number of bits to index into the directory cache.
DIR_WAYS	The number of ways in the directory cache.
DEFAULT_DEST	Sets a default destination to avoid unkown states.
ACTIVE_REQS	The number of active requests handled by the directory before it will reject the next request. It determines how many sets of registers are available to store the status of active requests.
CACHE_WORDS	The number of words in one cache line.
CACHE_WIDTH	The width of a cache line.
TAG_BITS	The number of bits in the tag for each cache line.
BUF_WIDTH	The number of bits in the buffer width.
WAY_BITS	The number of bits needed to represent the number of ways in the cache.
NUM_SHARERS	The number of sharer caches.
DIR_WIDTH	The number of bits needed to hold the SHARER_ID_BITS for all of the sharer caches.
DIR_DEPTH	The depth of the directory cache.

Table 4.115: Directory Controller Module Ports

Port	Direction	Width	Description
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	Synchronous, active high reset signal.
noc_msg_in	Input	MSG_BITS	The message entering the NoC.
noc_address_in	Input	ADDRESS_BITS	The address entering the NoC.
noc_data_in	Input	CACHE_WIDTH	The data entering the NoC.
noc_src_id	Input	ID_BITS	The ID of the NoC source.
packetizer_busy	Input	1	A signal that indicates when the packetizer is busy.
noc_msg_out	Output	MSG_BITS	The message exiting the NoC.

Table 4.115: Directory Controller Module Ports

Port	Direction	Width	Description
noc_address_out	Output	ADDRESS_BITS	The address exiting the NoC.
noc_data_out	Output	BUS_WIDTH	The data exiting the NoC.
noc_dest_id	Output	ID_BITS	The ID of the NoC destination.
mem_msg_in	Input	MSG_BITS	The message in from the memory.
mem_address_in	Input	ADDRESS_BITS	The address in from the memory.
mem_data_in	Input	CACHE_WIDTH	The data in from the memory.
mem_msg_out	Output	MSG_BITS	The message out to the memory.
mem_address_out	Output	ADDRESS_BITS	The address out to the memory.
mem_data_out	Output	CACHE_WIDTH	The data out to the memory.
report	Input	1	needs to be replaced with scan

Table 4.116: Directory Controller Unit Sub-Modules

Sub-Module	Description
message_handler	Receives messages from the NoC and buffers them. It also controls outgoing messages to the NoC and multiplexes between messages from the controller and any Nack messages from the handler itself.
cache_memory	Provides the directory cache memory. Its reads and writes are controlled by the directory controller. Its cache line holds a valid bit, a tag, and a sharer list. It provides limited pointer representation for sharers.
priority_encoder	Finds the next empty status register.
rr_arbiter	Selects the next status register ready to be serviced.
one_hot_decoder	One of the two one_hot_decoder modules finds the active request matching the current address and the other finds the index of the pointer that matches the current source.

4.3.5 Hierarchies

The hierarchies wrap the cache modules involved. The **two_level_cache_hierarchy** creates the L1 caches by instantiating the requisite number of L1 caches through the **L1cache_bus_wrapper** module. It also instantiates the needed buses with four **mux_bus** modules. In addition, a **coherence_controller** module and an **Lxcache_wrapper** unit are likewise instantiated. The **two_level_hierarchy_noc_wrapper** includes the NoC in its wrapper. It instantiates the **two_level_cache_hierarchy** module and an **lx_noc_interface** unit. The sub-modules are described elsewhere.

4.3.5.1 Two Level Cache Hierarchy

The **two_level_cache_hierarchy** organizes the highest levels of the cache hierarchy. It creates L1 cache and the buses needed to connect it to the L2 cache (and from there, lower level caches or a NoC). It manages coherence for the connected caches as well.

Table 4.117: Two Level Cache Hierarchy Module Parameters

Parameters	Description
STATUS_BITS_L1	The number of bits required to indicate the state of an L1 cache.
OFFSET_BITS_L1	The number of offset bits in each of the L1 caches. The value of this parameter is sliced and assigned per cache.
NUMBER_OF_WAYS_L1	The number of bits required to represent the number of ways in each L1 cache. It is sliced and assigned per cache.
INDEX_BITS_L1	The number of bits required to index into each L1 cache. Its value is sliced and assigned per cache.
REPLACEMENT_MODE_L1	Selects the replacement mode for all of the L1 caches.
STATUS_BITS_L2	The number of bits required to indicate the state of the L2 cache.
OFFSET_BITS_L2	The number of bits needed to represent the offset of the L2 cache.
NUMBER_OF_WAYS_L2	The number of ways in the L2 cache.
INDEX_BITS_L2	The number of bits required to index into the L2 cache.
REPLACEMENT_MODE_L2	Selects the replacement mode for the L2 cache.
L2_INCLUSION	Indicates whether the L2 cache is inclusive or exclusive.
COHERENCE_BITS	The number of bits required to indicate coherence state.
DATA_WIDTH	The number of bits in the core data path. Except for highly customized and experimental designs, this will be equivalent to the RISC-V XLEN parameter. Expected values are 32 or 64.
ADDRESS_BITS	The number of bits required to represent a memory address.
MSG_BITS	The number of bits in a bus message.
NUM_L1_CACHES	The number of L1 caches.
BUS_OFFSET_BITS	The number of offset bits on the bus.
MAX_OFFSET_BITS	The maximum number of offset bits.
L2_WORDS	The number of words in an L2 cache line. ???
L2_WIDTH	The number of bits of data in an L2 cache entry.
L2_TAG_BITS	The number of tag bits in an L2 cache entry.
L2_WAY_BITS	The number of bits required to represent which way is selected in the L2 cache.
L2_MBITS	The number of bits needed to represent the metadata of an L2 cache entry.

Table 4.118: Two Level Cache Hierarchy Module Ports

Port	Direction	Width	Description
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	Synchronous, active high reset signal.
read	Input	NUM_L1_CACHES	A signal to enable reading from a particular L1 cache.

Table 4.118: Two Level Cache Hierarchy Module Ports

Port	Direction	Width	Description
write	Input	NUM_L1_CACHES	A signal to enable writing to a particular L1 cache.
invalidate	Input	NUM_L1_CACHES	A signal to invalidate a particular L1 cache.
flush	Input	NUM_L1_CACHES	A signal to flush a particular L1 cache.
w_byte_en	Input	NUM_L1_CACHES * DATA_WIDTH/8	A signal to enable byte addressing in a particular L1 cache.
address	Input	NUM_L1_CACHES * ADDRESS_BITS	The address coming into a particular L1 cache.
data_in	Input	NUM_L1_CACHES * DATA_WIDTH	The data coming in to a particular L1 cache.
out_address	Output	NUM_L1_CACHES * ADDRESS_BITS	The address coming out of a particular L1 cache.
data_out	Output	NUM_L1_CACHES * DATA_WIDTH	The data coming out of a particular L1 cache.
valid	Output	NUM_L1_CACHES	A signal to indicate whether a particular L1 cache is valid.
ready	Output	NUM_L1_CACHES	A signal to indicate whether a particular L1 cache is ready.
mem2cachehier_msg	Input	MSG_BITS	The message from the memory to the cache hierarchy.
mem2cachehier_address	Input	ADDRESS_BITS	The address sent from the memory to the cache hierarchy.
mem2cachehier_data	Input	L2_WIDTH	The data sent from the memory to the cache hierarchy.
mem_intf_busy	Input	1	A signal to indicate whether the memory interface is busy.
mem_intf_address	Input	ADDRESS_BITS	The address in the memory interface.
mem_intf_address_valid	Input	1	A signal to indicate whether the address in the memory interface is valid.
cachehier2mem_msg	Output	MSG_BITS	The message from the cache hierarchy to the memory.
cachehier2mem_address	Output	ADDRESS_BITS	The address sent from the cache hierarchy to the memory.
cachehier2mem_data	Output	L2_WIDTH	The data sent from the cache hierarchy to the memory.

Table 4.118: Two Level Cache Hierarchy Module Ports

Port	Direction	Width	Description
port1_read	Input	1	A signal to enable a read from port1 of the cache.
port1_write	Input	1	A signal to enable a write to port1 of the cache.
port1_invalidate	Input	1	A signal to invalidate the cache entry at port1 of the cache.
port1_index	Input	INDEX_BITS_L2	The index of the cache entry at port1.
port1_tag	Input	L2_TAG_BITS	The tag of the cache entry at port1.
port1_metadata	Input	L2_MBITS	The metadata of the cache entry at port1.
port1_write_data	Input	L2_WIDTH	The data to be written to the cache entry at port1.
port1_way_select	Input	L2_WAY_BITS	The way selected for the cache entry at port1.
port1_read_data	Output	L2_WIDTH	The data to be read from the cache entry at port1.
port1_matched_way	Output	L2_WAY_BITS	A confirmation that the way selected for the cache entry at port1 is correct.
port1_coh_bits	Output	COHERENCE_BITS	The coherence state of the cache entry at port1.
port1_status_bits	Output	STATUS_BITS_L2	The state of the cache entry at port1.
port1_hit	Output	1	A signal to indicate a hit on the cache entry at port1.
scan	Input	1	A signal for testing the code.

Table 4.119: Two Level Cache Hierarchy Sub-Modules

Sub-Module	Description
L1cache_bus_wrapper	Creates the L1 cache.
mux_bus	Provides the needed buses to service cache communications.
coherence_controller	Manages traffic on the shared buses.
Lxcache_wrapper	Creates the L2 cache.

4.3.5.2 Two Level Hierarchy NoC Wrapper

The **two_level_hierarchy_noc_wrapper** organizes the connections between the last level cache (or the main memory) and the NoC.

Table 4.120: Two Level Hierarchy NoC Wrapper Module Parameters

Parameters	**Description**
STATUS_BITS_L1	The number of bits required to indicate the state of an L1 cache.
OFFSET_BITS_L1	The number of offset bits in each of the L1 caches. The value of this parameter is sliced and assigned per cache.
NUMBER_OF_WAYS_L1	The number of bits required to represent the number of ways in each L1 cache. It is sliced and assigned per cache.
INDEX_BITS_L1	The number of bits required to index into each L1 cache. Its value is sliced and assigned per cache.
REPLACEMENT_MODE_L1	Selects the replacement mode for all of the L1 caches.
STATUS_BITS_L2	The number of bits required to indicate the state of the L2 cache.
OFFSET_BITS_L2	The number of bits needed to represent the offset of the L2 cache.
NUMBER_OF_WAYS_L2	The number of ways in the L2 cache.
INDEX_BITS_L2	The number of bits required to index into the L2 cache.
REPLACEMENT_MODE_L2	Selects the replacement mode for the L2 cache.
L2_INCLUSION	Indicates whether the L2 cache is inclusive or exclusive.
COHERENCE_BITS	The number of bits required to indicate coherence state.
DATA_WIDTH	The number of bits in the core data path. Except for highly customized and experimental designs, this will be equivalent to the RISC-V XLEN parameter. Expected values are 32 or 64.
ADDRESS_BITS	The number of bits required to represent a memory address.
MSG_BITS	The number of bits in a bus message.
NUM_L1_CACHES	The number of L1 caches.
BUS_OFFSET_BITS	The number of offset bits on the bus.
MAX_OFFSET_BITS	The maximum number of offset bits.
REQ_BUF_DEPTH_BITS	The number of bits needed to represent the depth of the request buffer. Equal to log2(request buffer depth).
RESP_BUF_DEPTH_BITS	The number of bits needed to represent the depth of the response buffer. Equal to log2(response buffer depth).
CORE	A unique number to identify the core this module is in.
CACHE_NO	A unique number to identify the current cache.
CONTROLLER_TYPE	Specifies whether the controller type is blocking or non-blocking. The former will hold the signals to the interface asserted until the interface responds. The latter will deassert the signals after one cycle unless the interface is busy, in which case the interface will not register the message from the controller.
ID_BITS	The number of bits in the ID of sources.
DEFAULT_DEST	Assigns 0 as the default destination.
L2_WORDS	The number of words in an L2 cache line. ???
L2_WIDTH	The number of bits of data in an L2 cache entry.

Table 4.121: Two Level Hierarchy NoC Wrapper Module Ports

Port	Direction	Width	Description
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	Synchronous, active high reset signal.
read	Input	NUM_L1_CACHES	A signal to enable reading from a particular L1 cache.
write	Input	NUM_L1_CACHES	A signal to enable writing to a particular L1 cache.
invalidate	Input	NUM_L1_CACHES	A signal to invalidate a particular L1 cache.
flush	Input	NUM_L1_CACHES	A signal to flush a particular L1 cache.
w_byte_en	Input	NUM_L1_CACHES * DATA_WIDTH/8	A signal to enable byte addressing in a particular L1 cache.
address	Input	NUM_L1_CACHES * ADDRESS_BITS	The address coming into a particular L1 cache.
data_in	Input	NUM_L1_CACHES * DATA_WIDTH	The data coming in to a particular L1 cache.
out_address	Output	NUM_L1_CACHES * ADDRESS_BITS	The address coming out of a particular L1 cache.
data_out	Output	NUM_L1_CACHES * DATA_WIDTH	The data coming out of a particular L1 cache.
valid	Output	NUM_L1_CACHES	A signal to indicate whether a particular L1 cache is valid.
ready	Output	NUM_L1_CACHES	A signal to indicate whether a particular L1 cache is ready.
noc_msg_in	Input	MSG_BITS	The message entering the NoC.
noc_address_in	Input	ADDRESS_BITS	The address entering the NoC.
noc_data_in	Input	L2_WIDTH	The data entering the NoC.
noc_src_id	Input	ID_BITS	The ID of the NoC source.
packetizer_busy	Input	1	A signal that indicates when the packetizer is busy.
noc_msg_out	Output	MSG_BITS	The message exiting the NoC.
noc_address_out	Output	ADDRESS_BITS	The address exiting the NoC.
noc_data_out	Output	DATA_OUT	The data exiting the NoC.
noc_dest_id	Output	ID_BITS	The ID of the NoC destination.
interface_busy	Output	1	A signal to indicate whether the interface is busy.
scan	Input	1	A signal for testing the code.

Table 4.122: Two Level Hierarchy NoC Wrapper Sub-Modules

Sub-Module	Description
two_level_cache_hierarchy	Creates the L1 and L2 caches, as well as the necessary buses and controllers.
lx_nox_interface	Creates the NoC interface between the last level cache (or main memory) and the NoC.

4.3.6 Main Memory Modules

The main memory of the Trireme processor is parameterized, so it is thoroughly customizable. The main memory interface provides connections between cache, memory, and/or network.

4.3.6.1 Main Memory Module

The **main_memory** module includes a **simple_dual_port_ram** to serve as ___. At present, the main memory has one port. However, the number of ports is parameterized, and thus, it may be increased by the user by changing the parameter's value. Doing so should not require any modifications to the current code.

Table 4.123: Main Memory Module Parameters

Parameters	Description
DATA_WIDTH	The number of bits in the core data path. Except for highly customized and experimental designs, this will be equivalent to the RISC-V XLEN parameter. Expected values are 32 or 64.
ADDRESS_BITS	The number of bits required to represent a memory address.
MSG_BITS	The number of bits in a message to or from the memory.
INDEX_BITS	The number of bits required to index into the memory.
NUM_PORTS	The number of ports into/out of the memory.
PROGRAM	The program to be implemented by the processor. I think?

Table 4.124: Main Memory Module Ports

Port	Direction	Width	Description
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	Synchronous, active high reset signal.
msg_in	Input	NUM_PORTS * MSG_BITS	The message sent into the memory.
address	Input	NUM_PORTS * ADDRESS_WIDTH	The address to be accessed in the memory.
data_in	Input	NUM_PORTS * DATA_WIDTH	The data to be written in the memory.
msg_out	Output	NUM_PORTS * MSG_BITS	The message sent from the memory.
address_out	Output	NUM_PORTS * ADDRESS_BITS	The address that has been accessed in the memory.
data_out	Output	NUM_PORTS * DATA_WIDTH	The data to be read from the memory.

Table 4.125: Main Memory Module Sub-Modules

Sub-Module	Description
simple_dual_port_ram	Provides the BRAM for the memory.
arbiter	Provides an arbiter to govern which port has priority when NUM_PORTS is greater than 1.

4.3.6.2 Main Memory Interface Unit

The **main_memory_interface** connects the main memory to the caches and/or the NoC.

Table 4.126: Main Memory Interface Unit Parameters

Parameters	Description
OFFSET_BITS	A parameter that determines the number of words that can traverse the bus.
DATA_WIDTH	The number of bits in the core data path. Except for highly customized and experimental designs, this will be equivalent to the RISC-V XLEN parameter. Expected values are 32 or 64.
ADDRESS_BITS	The number of bits required to represent a memory address.
MSG_BITS	The number of bits in a message to or from the memory.

Table 4.127: Main Memory Interface Unit Ports

Port	Direction	Width	Description
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	Synchronous, active high reset signal.
cache2interface_msg	Input	MSG_BITS	The message entering the interface from the cache.
cache2interface_address	Input	ADDRESS_WIDTH	The address entering the interface from the cache.
cache2interface_data	Input	BUS_WIDTH	The data entering the interface from the cache.
interface2cache_msg	Output	MSG_BITS	The message exiting the interface for the cache.
interface2cache_address	Output	ADDRESS_WIDTH	The address exiting the interface for the cache.
interface2cache_data	Output	BUS_WIDTH	The data exiting the interface for the cache.
network2interface_msg	Input	MSG_BITS	The message from the network to the interface.
network2interface_address	Input	ADDRESS_WIDTH	The address sent from the network to the interface.
network2interface_data	Input	DATA_WIDTH	The data sent from the network to the interface.
interface2network_msg	Output	MSG_BITS	The message from the interface to the network.
interface2network_address	Output	ADDRESS_WIDTH	The address sent from the interface to the network.
interface2network_data	Output	DATA_WIDTH	The data sent from the interface to the network.

Table 4.127: Main Memory Interface Unit Ports

Port	Direction	Width	Description
mem2interface_msg	Input	MSG_BITS	The message from the memory to the interface.
mem2interface_address	Input	ADDRESS_WIDTH	The address entering the interface from the memory.
mem2interface_data	Input	DATA_WIDTH	The data entering the interface from the memory.
interface2mem_msg	Output	MSG_BITS	The message from the interface to the memory.
interface2mem_address	Output	ADDRESS_WIDTH	The address sent from the interface to the memory.
interface2mem_data	Output	DATA_WIDTH	The data sent from the interface to the memory.

4.4 Router Designs

The Trireme Platform includes a configurable router. Several routers can be connected together to form a Network-on-Chip (NoC). Currently only 2D-mesh topologies are supported. Other topologies can be added by replacing the routing logic within the routers.

Many signals in the router design are 2D-arrays of wires or registers with one packed and one unpacked dimension. These 2D-arrays are used to create pipelines for each router input port, allowing packets to be processed in parallel. Many of these signals need to be forwarded between stages, or between the control unit and stages. Typically only the selected value from the parallel pipeline dimension is input or output between modules, preventing the need to flatten and un-flatten the 2D-array in a generate loop.

The naming convention for such signals is to append two underscores and the indexing signal's name to the end of the 2D-array signal name to create the input/output vector signal name. A minimal example based on the **vc_allocation_stage** follows.

Listing 4.1: Naming convention for vector of signals selected from a 2D-array.

4.4.1 Base Router Modules

These modules are common to more than one type of router. Some routers may re-implement versions of these modules to support different features, such as different topologies or Virtual Channel (VC) allocation schemes.

4.4.1.1 Buffer Port

This module instantiates a FIFO buffer for each VC in a router port. Input signals select which buffer should be written to or read from. Finite State Machines are used to control the read and write signals of the different VC buffers.

Table 4.128: Buffer Port Parameters

Parameters	Description
DATA_WIDTH	The number of bits in a single word to be buffered. This parameter should be set to the width of a flit.
BUF_BITS	The log2 of the number of buffers/virtual channels used in this router port. One buffer is included in this module for each VC.
Q_DEPTH_BITS	The log2 of the number of depth of the buffer memory.

Table 4.128: Buffer Port Parameters

Parameters	Description
Q_IN_BUFFERS	The number of number of empty spaces in the buffer that triggers the full signals to go high. Setting this to 0 means the full signal will not go high untill the buffer is actually full. Setting this to a value ¿0, such as 3, means the full flag will go high when there are three empty spots in the buffer. This parameter is set to give the sender enough time to react to a full buffer.

Table 4.129: Buffer Port Ports

Port	Direction	Width	Description
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	Synchronous, active high reset signal. Clears the buffer memory.
wrtEn	Input	1	The buffer write enable signal. Writes to a full buffer are ignored. If buffer space is still available but the full signal is high becuse Q_IN_BUFFERS is ¿0, data can still be written.
write_data	Input	DATA_WIDTH	The data to write into a buffer.
in_vc	Input	BUF_BITS	The VC buffer to write the current write_data to.
rdEn	Input	1	The buffer read enable signal. When asserted, if the buffer is not empty, valid data will be output in the same cycle.
peek	Input	1	A signal to combinationally read data from the buffer without removing it from the queue. Holding this signal high will operate the fifo in "show-ahead-mode", where buffered data is output and held valid as soon as it is available.
read_vc	Input	BUF_BITS	The VC buffer to read from.
read_data	Output	DATA_WIDTH	The data read from the selected VC buffer.
out_vc	Output	BUF_BITS	The VC buffer that valid read_data was read from.
valid	Output	1	An active high signal to indicate read_data is valid.
empty	Output	2^{BUF_BITS}	An active high signal to indicate that no data is stored in the buffer. There is one empty signal for each VC buffer.
full	Output	2^{BUF_BITS}	An active high signal to indicate that the buffer is full. There is one full signal for each VC buffer.

Table 4.130: Buffer Port Sub-Modules

Sub-Module	Description
fifo	The First-In First-Out queue used to implement the buffer for each VC.

4.4.1.2 Input Buffer Stage

This module wraps all of a router's **buffer_port** modules into a single module. Router ports are broken up into "core ports" and "switch" ports. The core ports connect a core or processing element to the network. The switch ports connect routers together. Signals to read and write to each **buffer_port** are concatenated into flattened vectors of signals.

Table 4.131: Input Buffer Stage Parameters

Parameters	**Description**
FLOW_BITS	The number of bits used to identifiy a NoC flow. Usually this value is the source and destination NoC address concatenated together. The value is typically equvilant to $2 * log2(NUM\_SWITCHES)$, where NUM_SWITCHES is the number of routers in a NoC.
FLIT_WIDTH	The width of a single flit.
CORE_IN_PORTS	The number of ports to connect to a local core or processing element. Values other than 1 have not been tested.
SWITCH_IN_PORTS	The number of ports to connect to other routers/switches.
IN_PORTS	The total number of ports. This value is equal to CORE_IN_PORTS plus SWITCH_IN_PORTS. The default value of this parameter should always be used.
VC_BITS	The log2 value of the number of Virtual Channels (VC) in this router port.
VC_DEPTH_BITS	The log2 of buffer memory depth for each VC.

Table 4.132: Input Buffer Stage Ports

Port	**Direction**	**Width**	**Description**
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	Synchronous, active high reset signal. Clears the buffer memory.
c_flit_valid_in	Input	CORE_IN_PORTS	An active high signal to indicate the current core flit(s) are valid.
c_flit_in	Input	$FLIT\_WIDTH * CORE\_IN\_PORTS$	A flattened vector of input flits from the one or more core ports.
s_flit_valid_in	Input	SWITCH_IN_PORTS	An active high signal to indicate the current switch flits are valid.
s_flit_in	Input	$FLIT\_WIDTH * SWITCH\_IN\_PORTS$	A flattened vector of input flits from the one or more switch ports.
read_en_in	Input	IN_PORTS	A read enable signal for each port.
read_vc_in	Input	$VC\_BITS * IN\_PORTS$	A flattened vector of VC numbers to select wich VC buffer each port will read from.
read_flit_out	Output	$FLIT\_WIDTH * IN\_PORTS$	A flattened vector of flits output by each buffer port.
read_vc_out	Output	$VC\_BITS * IN\_PORTS$	A flattened vector of VC numbers to ID wich VC is being read for each buffer port.
read_valid_out	Output	IN_PORTS	A valid signal for the read data of each buffer port.
c_empty_out	Output	$2^{VC\_BITS} * CORE\_IN\_PORTS$	A vector of empty signals for each VC in each core buffer port.

Table 4.132: Input Buffer Stage Ports

Port	Direction	Width	Description
c_full_out	Output	$2^{VC\_BITS} * CORE\_IN\_PORTS$	A vector of full signals for each VC in each core buffer port.
s_empty_out	Output	$2^{VC\_BITS} * SWITCH\_IN\_PORTS$	A vector of empty signals for each VC in each switch buffer port.
s_full_out	Output	$2^{VC\_BITS} * SWITCH\_IN\_PORTS$	A vector of full signals for each VC in each switch buffer port.
scan	Input	1	An active high signal to print debug information in simulation. Currently unused in this module.

Table 4.133: Input Buffer Stage Sub-Modules

Sub-Module	Description
buffer_port	A group of FIFO queues for each VC in a router port.

4.4.1.3 Route Stage

This module takes in a flit for each router input port and assigns each of them a destination/output port. Routes are computed for each flit in parallel. Currently X-then-Y Dimension Order Routing and programmable table-based routing are supported. This module assumes routers are organized in a 2D-mesh topology.

Table 4.134: Route Stage Parameters

Parameters	Description
ROUTER_ID	A unique number to identify the router this module is in.
ID_BITS	The number of bits needed to uniquely identify a router.
RT_ALG	A string to indicate which routing algorithm to use. Currltly, valid values are "DOR_XY" for X-then-Y Dimension Order Routing or "TABLE" for programmable lookup table-based routing.
FLOW_BITS	The number of bits used to identifiy a NoC flow. Usually this value is the source and destination NoC address concatenated together. The value is typically equvilant to $2 * log2(NUM\_SWITCHES)$, where NUM_SWITCHES is the number of routers in a NoC.
TYPE_BITS	The number of bits used to represent the flit type (head, body, tail, or all). This value must be set to 2.
FLIT_WIDTH	The width of a single flit.
IN_PORTS	The total number of ports. This value is equal to CORE_IN_PORTS plus SWITCH_IN_PORTS.
OUT_PORTS	The total number of output ports. This value must be equal to IN_PORTS.
VC_BITS	The log2 value of the number of Virtual Channels (VC) in each router port.
ROW	The number of rows in a 2D mesh topology that this router is a part of.
COLUMN	The number of columns in a 2D mesh topology that this router is a part of.

Table 4.134: Route Stage Parameters

Parameters	Description
SCAN_CYCLES_MIN	Prevents scan logic from printing for cycles less than this threshold.
SCAN_CYCLES_MAX	Prevents scan logic from printing for cycles greater than this threshold.

Table 4.135: Route Stage Ports

Port	Direction	Width	Description
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	Synchronous, active high reset signal. Does not reset routing table contents.
on	Input	1	Enable signal for routing table logic. Usually held at a constant 1.
rt_table_program	Input	1	Write enable signal for routing table. Writes are ignored if on signal is low.
rt_flow_ID	Input	FLOW_BITS	Write address for routing table programming port.
rt_entry	Input	$log2(OUT\_PORTS)$	Write data for routing table programming port. Data value is the port number a flit should be output on for its given flow ID (the table address). Output port 0 is the core port 0. All core ports are numbered in incrimental order before switch ports. In a 2D-mesh switch ports are numberd W-S-E-N (counter clock-wise) with increasing port numbers.
flit_in	Input	$FLIT\_WIDTH * IN\_PORTS$	A flattend vector of flits (one for each router port) input to the route stage for route computation.
flit_vc_in	Input	$VC\_BITS * IN\_PORTS$	A flattend vector of VC numbers associated with each input flit.
flit_valid_in	Input	IN_PORTS	A valid signal for each input flit.
flit_id_out	Output	$FLOW\_BITS * IN\_PORTS$	A flattened vector of flow IDs for each flit.
flit_type_out	Output	$TYPE\_BITS * IN\_PORTS$	A flattened vector of flit types.
port_info_out	Output	$log2(OUT\_PORTS) * OUT\_PORTS$	A flattened vector of ports that each flit should be sent out.
scan	Input	1	An active high signal to print debug information in simulation.

4.4.1.4 Virtual Channel Allocation Stage

The Virtual Channel (VC) Allocation Stage module takes in flits from each router input port pipeline and outputs an assigned VC and a valid signal. One VC per output port can be allocated each cycle. Virtual Channels are allocated when a head flit enters the VC allocation stage. Deallocation occurs when a tail flit enters the switch traversal stage.

Dedicated **vc_fifo** modules store un-allocated virtual channels. When an allocation is requested, a virtual channel is read from the **vc_fifo** and stored in table of allocated VCs. When the VC is deallocated, it is removed from the table and written back into the **vc_fifo**.

Table 4.136: VC Allocation Stage Parameters

Parameters	**Description**
ROUTER_ID	A unique number to identify the router this module is in.
IN_PORTS	The total number of ports. This value is equal to CORE_IN_PORTS plus SWITCH_IN_PORTS.
OUT_PORTS	The total number of output ports. This value must be equal to IN_PORTS.
OUT_PORT_BITS	The log2 of OUT_PORTS, rounded up to the next integer.
ID_BITS	The number of bits needed to uniquely identify a router.
FLOW_BITS	The number of bits used to identifiy a NoC flow. Usually this value is the source and destination NoC address concatenated together. The value is typically equvilant to $2 * log2(NUM\_SWITCHES)$, where NUM_SWITCHES is the number of routers in a NoC.
VC_BITS	The log2 value of the number of Virtual Channels (VC) in each router port.
SCAN_CYCLES_MIN	Prevents scan logic from printing for cycles less than this threshold.
SCAN_CYCLES_MAX	Prevents scan logic from printing for cycles greater than this threshold.

Table 4.137: VC Allocation Stage Ports

Port	**Direction**	**Width**	**Description**
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	Synchronous, active high reset signal.
rt_table_program	Input	1	Write enable signal for routing table in the **route_stage** module. This signal is OR'd with the reset signal to act as an additional active high reset signal in this module.
allocate	Input	IN_PORTS	A signal from the router control unit to request a VC allocation for the incoming flits (the flit in the VC allocation stage of the router pipeline). One signal is included for each input port. May be high at same time as deallocate.
deallocate	Input	IN_PORTS	A signal from the router control unit to request that the VC of the flit in the switch traversal stage (**sw_traversal_stage**) stage is deallocated. May be high at same time as allocate.

Table 4.137: VC Allocation Stage Ports

Port	Direction	Width	Description
drained_flit_vc_st	Input	IN_PORTS	A signal from the router control unit to indicate a head flit is in the VC or Switch allocation stage. Because a VC is not de-allocated until the packet's tail filt reaches the Switch traversal stage, it is possible that another head flit piggybacked on the previously allocated VC. In this case, the VC cannot be de-allocated becasuse another packet is already in the pipeline using it.
flit_valid_rc	Input	IN_PORTS	Unused signal. May be deleted in the future.
port_info_rc	Input	$OUT\_PORT\_BITS * IN\_PORTS$	The computed output port of each flit entering the VC allocation stage.
flit_id_rc	Input	$FLOW\_BITS * IN\_PORTS$	The flow ID of each flit entering the VC allocation stage.
flit_vc_rc	Input	$VC\_BITS * IN\_PORTS$	The VC that each flit entering the VC allocation stage entered the router on.
flit_id_va_s_flit_vc_rc	Input	$FLOW\_BITS * IN\_PORTS$	The flow ID of the stalled flit on the current VC (the VC of the incommig flit).
flit_vc_va_s_flit_vc_rc	Input	$VC\_BITS * IN\_PORTS$	The VC of the stalled flit on the current VC (the VC of the incommig flit).
flit_valid_va_s_flit_vc_rc	Input	IN_PORTS	Valid signal of the stalled flit on the current VC (the VC of the incommig flit). When high, a stalled flit on the given port and incoming VC is waiting for an outgoing VC allocation.
port_info_va_s_flit_vc_rc	Input	$OUT\_PORT\_BITS * IN\_PORTS$	The output port of the stalled flit on the current VC (the VC of the incommig flit).
flit_vc_va	Input	$VC\_BITS * IN\_PORTS$	Unused. May be deleted in the future.
flit_vc_st	Input	$VC\_BITS * IN\_PORTS$	The VC that each flit in the switch traversal stage entered the router on.
port_info_st	Input	$OUT\_PORT\_BITS * IN\_PORTS$	The computed output port for each flit in the switch traversal stage.

Table 4.137: VC Allocation Stage Ports

Port	Direction	Width	Description
out_vc_st	Input	$VC\_BITS * IN\_PORTS$	The VC allocated to each flit that is in the switch traversal stage.
full_in	Input	$2^{VC\_BITS} * IN\_PORTS$	The buffer full signal from the router connected to each of this router's output ports. VC's cannot be allocated to full buffers, otherwise, a dependency is created that can deadlock the network.
out_vc_va	Output	$VC\_BITS * IN\_PORTS$	The allocated VC for each flit in the VC allocation stage.
out_vc_status_va	Output	IN_PORTS	A valid signal for each flit in the VC allocation stage. This signal will be high when a VC is allocated to a flit exiting the VC allocation stage.
scan	Input	1	An active high signal to print debug information in simulation.

Table 4.138: VC Allocation Stage Sub-Modules

Sub-Module	Description
vc_fifo	A custom FIFO to store the queue of available VCs. The only difference between this module and the **fifo** module is the reset behavior.

4.4.1.5 Virtual Channel FIFO

A modified version of the **fifo** module. The reset behavior of the **vc_fifo** module has been altered to fill the fifo will all available virtual channels, instead of setting the fifo to an empty state.

Table 4.139: VC FIFO Parameters

Parameters	Description
NODE_ID	A unique number to identify the node/router this module is in.
DATA_WIDTH	The bit width of the FIFO read and write ports.
Q_DEPTH_BITS	The number of address bits for the FIFO memory. The FIFO depth is $2^{Q\_DEPTH\_BITS}$

Table 4.140: VC FIFO Ports

Port	Direction	Width	Description
clk	Input	1	The clock for all synchronous logic and the read and write ports.
reset	Input	1	Synchronous, active high reset signal. Empties the FIFO.
write_data	Input	DATA_WIDTH	Input for new data.

Table 4.140: VC FIFO Ports

Port	Direction	Width	Description
wrtEn	Input	1	Active high write enable. Add write_data to the queue if it is not full
rdEn	Input	1	Active high read enable. If the queue is not empty, remove a word from the queue and output it. Data will be valid the same cycle.
peek	Input	1	Active high signal. Read the next valid word from the queue in the same cycle. The word is not removed from the queue.
read_data	Output	DATA_WIDTH	The data output from the queue. Valid if valid signal is high.
valid	Output	1	Active high. Indicates that read_data is valid.
full	Output	1	Active high. Indicates there are more than $(2^{Q\_DEPTH\_BITS}) - (Q\_IN\_BUFFERS)$ words in the queue.
empty	Output	1	Active high. Indicates the queue is empty. Will go high the same cycle the last word is read from the queue.

4.4.1.6 Switch Request Builder

The switch request builder is a combinational module that sits in the same pipeline stage as the **vc_allocation_stage** module. The **switch_request_builder** module takes in the computed output port of the flit currently in the VC allocation stage in addition to the computed output port for any flits in the same VC that are stalled in the switch allocation stage or switch traversal stage. Stalled flits in the switch traversal stage, then switch allocation stage take priority when building switch allocation requests.

Table 4.141: Switch Request Builder Parameters

Parameters	Description
IN_PORTS	The total number of ports. This value is equal to CORE_IN_PORTS plus SWITCH_IN_PORTS.
OUT_PORT_BITS	The log2 of IN_PORTS (or OUT_PORTS), rounded up to the next integer.

Table 4.142: Switch Request Builder Ports

Port	Direction	Width	Description
flit_valid_va	Input	IN_PORTS	An active high signal to indicate if each flit is valid.
port_info_va	Input	$OUT\_PORT\_BITS * IN\_PORTS$	The computed output port for each flit in the VC allocation stage.
flit_valid_sa_s	Input	IN_PORTS	The valid signal for the stalled flit in the switch allocation stage.
port_info_sa_s	Input	$OUT\_PORT\_BITS * IN\_PORTS$	The computed output port for each stalled flit in the switch allocation stage.
flit_valid_st_s	Input	IN_PORTS	The valid signal for the stalled flit in the switch traversal stage.
port_info_st_s	Input	$OUT\_PORT\_BITS * IN\_PORTS$	The computed output port for each stalled flit in the switch traversal stage.

Table 4.142: Switch Request Builder Ports

Port	Direction	Width	Description
vc_stall__flit_vc_va	Input	IN_PORTS	A control signal to indicate that, for a given flit's incomming virtual channel, the flit has not been allocated an outgoing virtual cannel by the VC allocation stage.
sw_stall__flit_vc_va	Input	IN_PORTS	A control signal to indicate that, for a given flit's incomming virtual channel, the flit has not been allocated an time slot on the router crossbar by the switch allocation stage.
requests	Input	IN_PORTS	An active high signal to indicate a valid flit is requesting a time slot to travers the router crossbar.
req_ports	Input	$OUT\_PORT\_BITS * IN\_PORTS$	The computed output port of each flit entering the switch allocation stage. The comuted output port is the port that the flit is requesting a time slot on. This signal is equivilant to port_info.
scan	Input	1	An active high signal to print debug information in simulation.

4.4.1.7 Switch Allocation Stage

The switch allocation stage instantiates an **arbiter** to select which flits may traverse the router crossbar and exit the router next. The original VC field of each flit (the one allocated by the last router the flit was at) is replaced in this stage with the VC allocated in the VC allocation stage. Note that the original flit continues down the pipeline for control purposes while the updated flit continues down the pipeline as data.

Table 4.143: Switch Allocation Stage Parameters

Parameters	Description
ROUTER_ID	A unique number to identify the router this module is in.
FLIT_WIDTH	The number of bits in each flit.
IN_PORTS	The total number of ports. This value is equal to CORE_IN_PORTS plus SWITCH_IN_PORTS.
OUT_PORT_BITS	The log2 of IN_PORTS (or OUT_PORTS), rounded up to the next integer.
HEAD_BITS	The number of non-data bits in a flit.
VC_BITS	The log2 value of the number of Virtual Channels (VC) in each router port.
SCAN_CYCLES_MIN	Prevents scan logic from printing for cycles less than this threshold.
SCAN_CYCLES_MAX	Prevents scan logic from printing for cycles greater than this threshold.

Table 4.144: Switch Allocation Stage Ports

Port	Direction	Width	Description
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	Synchronous, active high reset signal.
flit_sa_in	Input	$FLIT\_WIDTH * IN\_PORTS$	The input flits from each router input port pipeline.
out_vc_sa	Output	$VC\_BITS * IN\_PORTS$	The allocated VC for each incomming flit.
req_ports	Input	$OUT\_PORT\_BITS * IN\_PORTS$	The computed output port of each flit entering the switch allocation stage. The comuted output port is the port that the flit is requesting a time slot on. This signal is equivilant to port_info.
requests	Input	IN_PORTS	An active high signal to indicate a valid flit is requesting a time slot to travers the router crossbar.
grants	Output	IN_PORTS	An active high signal to indicate a valid flit has received a time slot to travers the router crossbar.
new_flit_sa	Output	$FLIT\_WIDTH * IN\_PORTS$	The updated flits that will be sent over the router crossbar and out a router output port. The original VC field in these flits is replaced by the VC that the VC allocation stage allocated.
scan	Input	1	An active high signal to print debug information in simulation.

Table 4.145: Switch Allocation Stage Sub-Modules

Sub-Module	Description
arbiter	A module to select which flit may travers the router crossbar next.

4.4.1.8 Switch Traversal Stage

The switch traversal stage wraps the **crossbar** module to create an module interface that follows the same naming conventions as the rest of the router modules.

Table 4.146: Switch Traversal Stage Parameters

Parameters	Description
FLIT_WIDTH	The number of bits in each flit.
IN_PORTS	The total number of ports. This value is equal to CORE_IN_PORTS plus SWITCH_IN_PORTS.
OUT_PORTS	The total number of output ports. This value must be equal to IN_PORTS.
OUT_PORT_BITS	The log2 of IN_PORTS (or OUT_PORTS), rounded up to the next integer.

Table 4.147: Switch Traversal Stage Ports

Port	Direction	Width	Description
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	Synchronous, active high reset signal.
flit_in	Input	$FLIT\_WIDTH * IN\_PORTS$	The input flits from each router input port pipeline.
req_ports	Input	$OUT\_PORT\_BITS * IN\_PORTS$	The computed output port of each flit entering the switch traversal stage. This signal is equivilant to port_info value.
grants	Input	IN_PORTS	An active high signal to indicate a valid flit has received a time slot to travers the router crossbar.
flit_out	Output	$FLIT\_WIDTH * OUT\_PORTS$	The flits output by the **crossbar** inside the switch traversal stage.
scan	Input	1	An active high signal to print debug information in simulation. Unused in this module.

Table 4.148: Switch Traversal Stage Sub-Modules

Sub-Module	Description
crossbar	A crossbar module to connect every input port to every output port.

4.4.1.9 Crossbar

This module connects each one of its input ports to each of its output ports. Input ports select a desired output port and an external controller arbitrates access by generating a "grants" signal. The number of input ports can be different from the number of output ports. However some router modules assume the number of input ports matches the number of output ports.

Table 4.149: Crossbar Parameters

Parameters	Description
DATA_WIDTH	The number of bits per word, per input port.
IN_PORTS	The total number of crossbar ports.
OUT_PORTS	The total number of crosbar output ports.
OUT_PORT_BITS	The log2 of OUT_PORTS, rounded up to the next integer.

Table 4.150: Crossbar Ports

Port	Direction	Width	Description
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	Synchronous, active high reset signal.
in_data	Input	$DATA\_WIDTH * IN\_PORTS$	The input data words to each input port of the crossbar.

Table 4.150: Crossbar Ports

Port	Direction	Width	Description
req_ports	Input	$OUT\_PORT\_BITS * IN\_PORTS$	The port an input word requests to exit the crossbar on.
grants	Input	IN_PORTS	An active high signal to indicate a valid data word has received a time slot to travers the crossbar by an external controller.
out_data	Output	$DATA\_WIDTH * OUT\_PORTS$	The output data words exiting the crossbar on each output port.
valid	Output	OUT_PORTS	An active high signal to indicate if each output word is valid.

4.4.2 Fast Router

Figures 4.1 and 4.2 depict the micro-architecture of the "fast router" design. This version of the router is not deadlock free. Future versions will include an updated VC allocation stage to improve performance and prevent deadlocks.

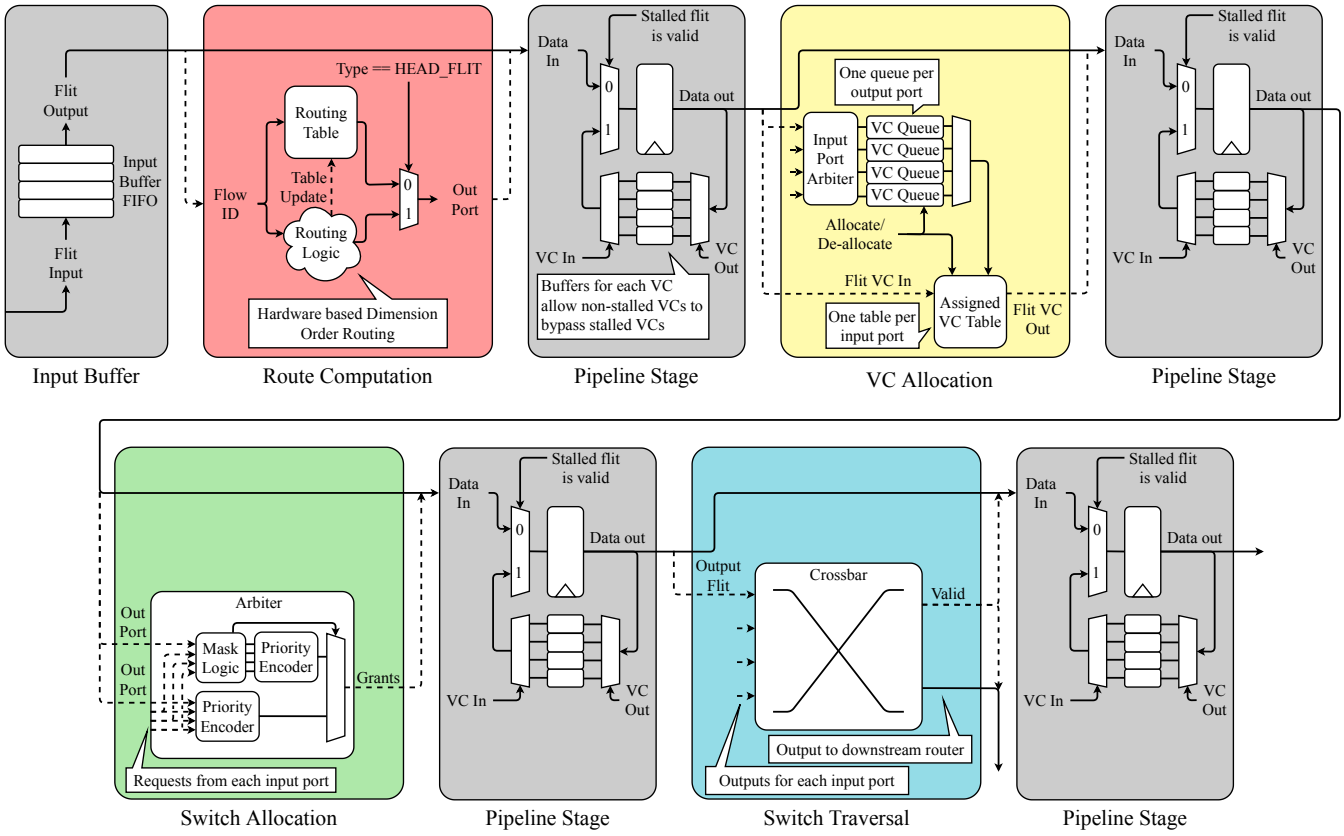

Figure 4.1: Detailed block diagram of a single pipeline within the Fast Router.

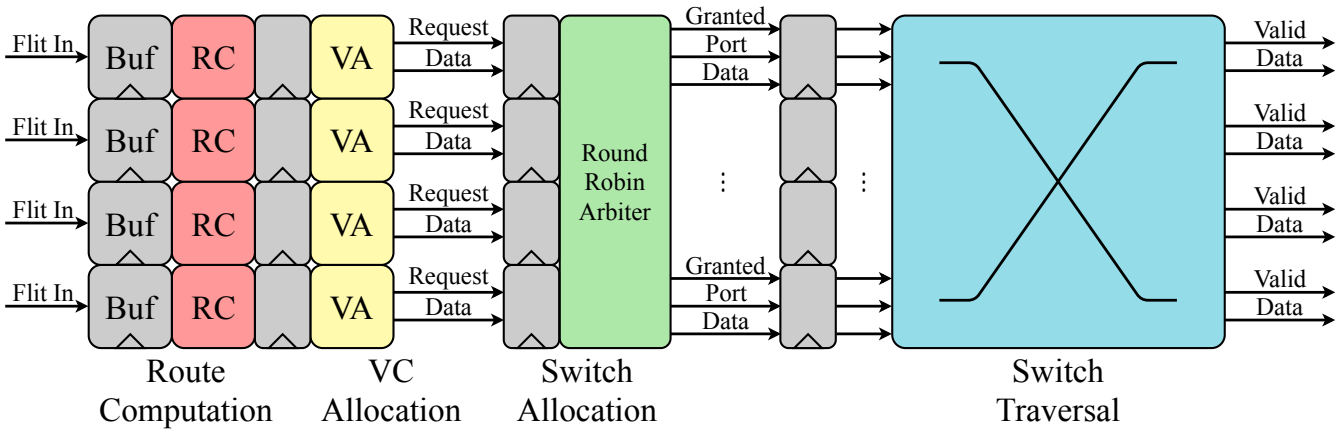

Figure 4.2: Block diagram of the Fast Router with its parallel pipelines for each input port.

4.5 Node Designs

In The Trireme Platform, "node" modules are used to build Networks-on-Chip.

4.5.1 Base Node Modules

These modules are common to more than one type of node.

4.5.1.1 BRAM Node Bridge Module

The **bram_node_bridge** module takes data from the node interface and distributes it to the BRAM or a node, as appropriate.

Table 4.151: BRAM Node Bridge Module Parameters

Parameters	Description
NODE_ID	A unique identifier for each node.
NODE_DATA_WIDTH	The width of data for the node.
CORE_DATA_WIDTH	The width of data in the core.
ADDRESS_BITS	The number of bits in the address.
MSG_BITS	The number of bits in the message.
ID_BITS	The number of bits in the NODE_ID.
LOCAL_ADDRESS_BITS	The number of bits in the local-to-the-node address.
SCAN_CYCLES_MIN	Prevents scan logic from printing for cycles less than this threshold.
SCAN_CYCLES_MAX	Prevents scan logic from printing for cycels greater than this threshold.

Table 4.152: BRAM Node Bridge Module Ports

Port	Direction	Width	Description
clock	Input	1	The clock signal for synchronous logic.
reset	Input	1	Synchronous, active high reset signal. Sets x0 in the register file to zero and resets scan logic.

Table 4.152: BRAM Node Bridge Module Ports

Port	Direction	Width	Description
msg_in	Input	MSG_BITS	The message into the bridge from the node interface.
address_in	Input	ADDRESS_BITS	The address into the bridge from the node interface.
data_in	Input	NODE_DATA_WIDTH	The data into the bridge from the node interface.
src_id_in	Input	ID_BITS	The ID of the source node for the message, address, and data into the bridge from the node interface.
interface_busy_in	Output	1	A signal to indicate that the interface is busy.
bram_wr_en	Output	1	A signal to enable writing to the BRAM.
bram_byte_en	Output	CORE_DATA_WIDTH/8	A signal to select which byte(s) are accessed. ???
bram_address_out	Output	LOCAL_ADDRESS_BITS	The address sent to the BRAM from the bridge.
bram_data_in	Input	CORE_DATA_WIDTH	The data into the bridge from the BRAM.
msg_out	Output	MSG_BITS	The message from the bridge to the node.
address_out	Output	ADDRESS_BITS	The address from the bridge to the node.
data_out	Output	NODE_DATA_WIDTH	The data from the bridge to the node.
dest_id_out	Output	ID_BITS	The ID of the destination node for the message, address, and data from the bridge.
interface_busy_out	Input	1	A signal to indicate that the interface is busy.
scan	Input	1	An active high signal to print debug information in simulation.

4.5.1.2 Debug Core Module

4.5.1.3 Debug Link Module

4.5.1.4 Depacketizer Module

4.5.1.5 Injection Core Module

4.5.1.6 Interface Arbiter Module

4.5.1.7 Interface Splitter Module

4.5.1.8 NOC Node Bridge Module

4.5.1.9 Packetizer Module

4.5.1.10 Packetizer Multi-VC Module

4.6 Top Level Designs

Top level designs in The Trireme Platform contain common, core, memory, router or node modules within them.

4.6.0.1 Single Cycle Top Module

The Single Cycle Top Module instantiates the single cycle core, the memory interface, and the single cycle memory subsystem. It represents a complete processor environment.

4.6.0.2 Single Cycle Top Module with BRAM

The Single Cycle Top Module with BRAM ("single_cycle_BRAM_top") instantiates the single cycle core, the memory interface, and the dual port BRAM memory subsystem. It represents a complete processor environment.

4.6.0.3 Single Cycle Top Module with Cache

The Single Cycle Top Module with Cache ("single_cycle_cache_top") instantiates the single cycle core, the memory interface, the cache hierarchy, the main memory interface, and the main memory. It represents a complete processor environment.

4.6.0.4 Five Stage Top Module

The Five Stage Top Module instantiates the five stage core, the memory interface, and the single cycle memory subsystem. It represents a complete processor environment.

4.6.0.5 Five Stage Top Module with BRAM

The Five Stage Top Module with BRAM ("five_stage_BRAM_top") instantiates the five stage core, the memory interface, and the dual port BRAM memory subsystem. It represents a complete processor environment.

4.6.0.6 Five Stage Top Module with Cache

The Five Stage Top Module with Cache ("five_stage_cache_top") instantiates the five stage core, the memory interface, the cache hierarchy, the main memory interface, and the main memory. It represents a complete processor environment.

4.6.0.7 Seven Stage Top Module with BRAM

The Seven Stage Top Module with BRAM ("seven_stage_BRAM_top") instantiates the seven stage core, the memory interface, and the dual port BRAM memory subsystem. It represents a complete processor environment.

4.6.0.8 Seven Stage Top Module with Cache

The Seven Stage Top Module with Cache ("seven_stage_cache_top") instantiates the seven stage core, the memory interface, the cache hierarchy, the main memory interface, and the main memory. It represents a complete processor environment.