Project RISCY Final Report

Shuai Zhang

Contents

Project Overview / Revised Proposal

Introduction

RISC-V is an open-source instruction set architecture developed by UC-Berkley and have been picking up popularity and community over the years. Apart from the open-source instruction encoding, there are also developing tools such as C compiler, simulator etc. available. I have an on-going project called 'Riscy' which is to write my own risc-v core with the ultimate goal is to run a primitive Unix operating system (such as xv6) on a FPGA loaded with my risc-v core. The project is called Riscy is because I morphed RISC-V's V to Y, so it makes easier for me to read. (And because I can see it will consume me hundreds of hours building this project and it's a *ris-ky* decision) The newest version of source code can be found at [0], and the source code when I wrote this report is available in the appendix

To build the project:
Go to /par/de1 and make

Or build from scratch:
git clone https://github.com/Bald-Badger/riscy.git
git submodule update --init –recursive
cd riscy/par/de1
make

Project Timeline:

1. Use proper AXI-Lite interface, indefinite-delay ram for the core instead of a fixed 1 cycle delay (on simulation)
2. Use AXI-Lite bus and interconnect to connect everything (cache, memory, bootloader etc.)
3. Map everything to the De-1 development board, and use the tiny on-chip memory for now
4. fulfill the timing requirement for the core, and run a very basic program (such as blink a led using memory-mapped IO)
5. develop a program that enables the Arm core on the de-1 board to have access to the AXI-Lite bus on the risc-v core, and can perform basic write / read operations to the risc-v memory space via AXI-Lite bus
6. Polish the program so that I can use it as a debugger / data injector to my risc-v core, and able to interact with is using uart serial connecting the de-1 board to the user console.
7. (hopefully) develop an interface that enables the risc-v code to access the hardware memory (SDRAM or DDR3) so that I can run larger programs

Each step was proposed to last for a week starting from the due date of the proposal. I have been strictly following the proposed timeline and accomplished each proposed goal.

A listing of what I did, and lesson learned

Previous status

Before this semester, the status of project RISCY is as follows:

1. Have a brute-force, hard-wired, hard-coded classical five stage pipeline core design running on simulation without noticeable bugs.
2. The system uses a single-cycle delay memory access delay and simulates with pre-loaded code inside a full 32bit address space. Separate instruction and data ram.
3. The simulation was able to pass all single instruction tests provided by the official risc-v-tests repository, link here:

Current Status

For this Semester I accomplished the following: (everything I did not explicitly marked reference are my own work)

1. Completely rewrote the instruction fetch unit and memory pipeline stage. I used to use a rudimentary, simplified wishbone-ish bus connection (which I named as the SIMP bus)
    a. The instruction fetch unit now have a pre-fetch buffer that continuously pre-fetches instructions from the system bus. So that whenever there's a pipeline stall the instruction fetch unit will not waste time on waiting.
    b. The memory pipeline stage now has proper stalling logic that waits the memory read/write to finishing.
2. Unified AXI-Lite system bus for maximum compatibility.
    a. Every module is linked to each other by a unified AXI-Lite interface
    b. Implemented / copied AXI-Lite interface controller for each component, which includes:
        i. Studied the AXI interface specification [2], which took much longer than I expected…
        ii. AXI master for Instruction fetch unit
        iii. AXI master for Data memory read/write unit
        iv. AXI slave for ram for simulation (I copied it online, reference: [1])
        v. AXI slave for UART module
        vi. AXI slave for 7-seg module
        vii. AXI slave for the on-board SDRAM module (I copied it online, reference: [4])
            1. It sounds easy that I copied it online, but in practice it was exceeding hard to get it working. It was a general-proposed SDRAM controller, and I had to specifically tune its parameters and write specific buffers for it to get working. Also, the timing was also a hazard and I fixed it by leading the SDRAM clock by 3ns over the AXI bus clock. I spent way too much time on this part…
            2. Using On-board BRAM was not an option as the generated demo ELF file was as large as 1.3MB, which is exceeding the maximum BRAM available on the FPGA
    c. Implemented a AXI system level crossbar interconnect so that each component can talk to each other without explicit switching.
        i. I did not write this part myself, copied from [1]
3. Test codes are now much more sophisticated that are generated from a legitimate C++ compiler with regular C++ code instead of what was provide by a third-party bareback assembly compiler.
    a. The official Risc-V GNU C/C++ toolchain are available at [3]
    b. Wrote the longest Makefile in my dear life to incorporate the generated machine code with my bare metal core. Everything is statically compiled if possible because I don't have the standard C library installed on my C core.
    c. Avoided using the standard C library as some of them needs system call to operate. I did not plan to implement my own system call for this semester (such as _sbrk(), _write() etc)
    d. Played with the linker script so that the generated code will be placed at the memory position that I wished it to be so that I can control my core to boot from there
4. Developed a bootloader for the entire system on the HPS system. The HPS's AXI bus is connected to the AXI crossbar for the RISC-V core, as well as other peripherals. In this case I can boot load the core with a robust system.
    a. The bootloader can also control the peripherals, in this way I can debug my peripherals without suspecting that the problem is in the peripheral code instead of my CPU core.
    b. The bootloader can also act as a debugger for the system as it's also connected to the SDRAM thanks to the unified AXI interface and AXI crossbar. I can simply send peek

instructions to the HPS, and the bootloader can return the memory value on a certain memory location.

5. AXI peripherals are also developed including:
    a. A custom UART module that supports AXI read and write.
        i. Using 4 of the GPIO pins: TX, RX, and two flow control pins
        ii. Reading from the UART module can return the bytes available in the buffer
        iii. Writing to the UART module can output a byte to the Serial interface
        iv. Details will be introduced later in this report
    b. A custom 7-seg module that supports AXI read and write
        i. Reading from it will return the 7-byte value displayed on the 7seg
        ii. Writing to it will update the 7-byte value based on the value being written
        iii. Details will be introduced later in this report
6. The most important part: mapping everything from simulation to actual synthesizable code.
    a. Got rid of all lazy system tasks such as $signed () or $initial ()
    b. Fixed timing issues by:
        i. reducing excess combinational logic,
        ii. deleting combinational chain,
        iii. optimized forwarding logic,
        iv. pipelining the system even more
        v. memory / buffer fetches are registered
        vi. using unprioritized combinational logic
        vii. fine tuning the Quartus compilation setting for maximum speed
    c. The final system maximum clock speed raised from 23MHz to 51MHz, now running at native clock speed!
7. Finally, it just worked. If I could only write one sentence in my project that would be “it worked” A tic-tac-toe demo was presented in the final presentation, the demo was able to show my RISC-V core is able to:
    a. Read code from memory and executes them proving the CPU works
    b. Write data to AXI peripherals proving my AXI master and slave interface controller works
    c. Read data from user input proving my AXI UART module works
    d. Execute a object oriented C++ code proving that my linker script worked, and successfully statically compiled malloc () in my code
    e. No glitches, proving the timing is good and able to work on 50MHz system clock

What I learned

This have been a huge project I have been working on, and I have dedicated hundreds of my life into this project this semester. Here are some of my thoughts. For the students who are reading this in the future, take these for a grain of salt as these are only my personal thoughts.

1. Although I spend hundreds of hours on this project, it would only take about half of the time If I knew what I should do in the first place
    a. I spend 30+ hours on the SDRAM controller wishing to get it to work by my self. However, the bug was really classic and obvious, and I was able to find the way to solve it inside an intel technical document. (which I should have been doing in the first place)
    b. Some if the bugs I encountered were so specific and rare that even no one asked them on stack overflow. I didn’t dare to post a question on there due to the how toxic the community is on stack overflow. However, I posted my question on Reddit and a whole bunch of nice dudes answered my question in detail. The Reddit guys were pretty chill and way less toxic than the grumpy condescending ones on Stack Overflow. I should’ve asked for help in the first place to experts instead of trying to figure out by myself.
2. SDRAM is very hard to work with. Use BRAM if possible

    a. Each SDRAM have its own timing profile and it's very hard to tune. The open source SDRAM control core I used took a lot of parameters for granted and didn't provide an intuitive way to tune them. (All the parameters are hard-coded in the code)
    b. The Quartus Standard version have a really nice snd working SDRAM controller. However it only works as stand-alone module and does not gets along well with System Designer. This means you must instantiate it outside of your .qsys file. Too bad, by the time I know this I already finished debugging the opensource SDRAM controller.
3. Plan your time wisely. There was a week that I became obsessed with this project because everything has been going on really smoothly. As a result, I slept very little and didn't have time to work on my other assignments. That was a stressful week.
4. Don't trust open-source software, especially the ones are not actively maintained.
    a. To avoid re-inventing the wheels, I copied a bunch of code on GitHub and tried to incorporate them with my code. However, after some desperate debugging, I found out that the open-source code themselves are buggy despite the fact that they have a really fancy Readme file. I ended up with ditching the majority of the code I copied and only keeping the ones I thoroughly debugged.
5. Stack Overflow is a really toxic community
6. Stack Exchange is a lightly less but still toxic community
7. It you report a bug or issue on GitHub about their open-sourced code, don't expect them to be thankful.
8. Reddit is a surprising less toxic community on answering noob questions.

Project Design Document

Algorithms:

Frankly, there is not a whole lot of fancy algorithms involved in this project. Everything is retrospective and old-fashioned. Instead, I will just list the key specs of this project

1. CPU Core:
    a. In-Order Classical 5 stage pipeline
    b. Support RV32IMA instruction set
        i. 32-bit word size, small endian, byte addressable
        ii. I: base integer operation ISA
        iii. M: integer multiplication / division operation ISA, not verified and evaluated, nor used
        iv. A: atomic instruction set, not verified, evaluated, nor used
    c. Full forwarding to solve hazard
        i. Side notes 1: this is a BAD idea because it hurts timing a LOT. Should have gone with plain stalling…
        ii. Side notes 2: the forwarding / stalling logic is so complicated that I once though I should have gone with out-of-order core design…
    d. Parameterizable instruction fetch buffer
    e. Parameterizable memory read/write buffer (implemented but not verified)
    f. Parameterizable system cache (implemented but buggy)
        i. Can be either unified or separate I / D cache
        ii. 2-Way set associative cache, parameterizable depth
        iii. Write-back policy, LRU victim eviction policy
    g. Atomic instruction support
    h. Unified system bus, can be either AXI4 or AXI-Lite
        i. AXIL to AXI and AXI to AXIL bridge available
    i. Pure System Verilog, platform independent, no priority IP (except for the mult/div unit)
    j. Formally verified (better than nothing)

2. SoC System:
    a. 512Mb on-board SDRAM
        i. Controller with AXI wrapper
    b. Unified bus everywhere
        i. All buses are either AXI or AXI-Lite, made bus translation overhead much smaller.
    c. Support most on-board peripheral:
        i. All GPIO
        ii. UART for serial I/O
        iii. SW and push button
        iv. 7-seg for system state indicator

Resource Budgets for the while SoC system

Logic: around 11K logic element at 50MHz timing requirement, 13K if 40MHz
Memory block: 0 (set all buffer depth to 0), infinite if I just ramp up the numbers
DSP block: 23 if support mult/div instruction, 0 if not

The Hardware / Software Interface / memory layout

R: readable, W: writeable, X: execuable

1. SDRAM module: (0x0000_0000 – 0x03FF_FFFF)
    a. R/W/X 512Mb, 64MB, 16M Words
2. 7-Seg module: (0x0400_0000 – 0x0400_001F)
    a. 8 32bit registers, lowest 8 bits are valid, others have undefined behavior.
    b. All registers are R/W
    c. 0x0400_0000: seg digit #0
    d. 0x0400_0004: seg digit #1
    e. 0x0400_0008: seg digit #2
    f. 0x0400_000C: seg digit #3
    g. 0x0400_0010: seg digit #4
    h. 0x0400_0014: seg digit #5
    i. 0x0400_0018: seg digit #6
    j. 0x0400_001F: undefined
    k. Writing to digit 0 – 6's lowest 8 bit will set the seg number
    l. Reading from digit 0-6 's lowest 8 bit will read from the seg number
3. The UART module: (0x0401_0000 – 0x0401_000F)
    a. 2 32bit registers, others have undefined behaviors
    b. (R/W) 0x0401_0000: UART data register
        i. Reading from this register will read the top-most byte from UART RX FIFO, and the data is placed in the lowest byte in the 32-bit word. If the RX FIFO is empty, then it will return undefined number.
        ii. Writing to this register will write a single byte from the lowest byte from the data bus to the UART TX FIFO, and it will be automatically sent to the serial interface if the host can accept more data (indicated by a low on flow control wire) Writing a full FIFO will loss the data without explicit warning or bus error
    c. (R) 0x0401_0004: UART RX FIFO LEN register
        i. Reading from this register will return the length of the RX FIFO. A value of 0 means the FIFO is empty and no data available. Writing to this register will have undefined behavior so don't do it. Maybe the world will end idk

Functional descriptions of each block

Quick introduction on RISC-V

RISC-V ISA specified 40 basic instructions that every designer should implement, the instructions are shown as follows, the detailed specification can be found at [5]:

31 27	26 25	24 20	19 15	14 12	11 7	6 0	
funct7		rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]			rs1	funct3	rd	opcode	I-type
imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]					rd	opcode	U-type
imm[20|10:1|11|19:12]					rd	opcode	J-type

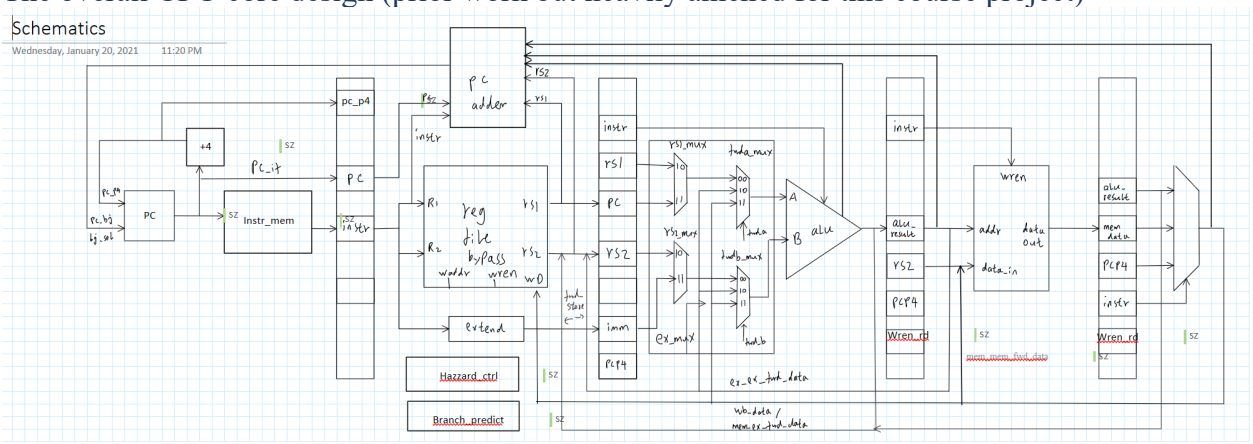
RV32I Base Instruction Set

imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND
fm | pred | succ		rs1	000	rd	0001111	FENCE
000000000000		00000	000	00000	1110011	ECALL
000000000001		00000	000	00000	1110011	EBREAK

The Full RISC-V Base ISA

The processor is able to perform the most basic operation and flow control with the support of the 40 instructions above. The programmer is also allow to implemented a list of extensions that will make the core more powerful such as the A extension that allows atomic instruction, and the M extension that allows integer multiplication and division.

The overall CPU core design (prior work but heavily amened for this course project)

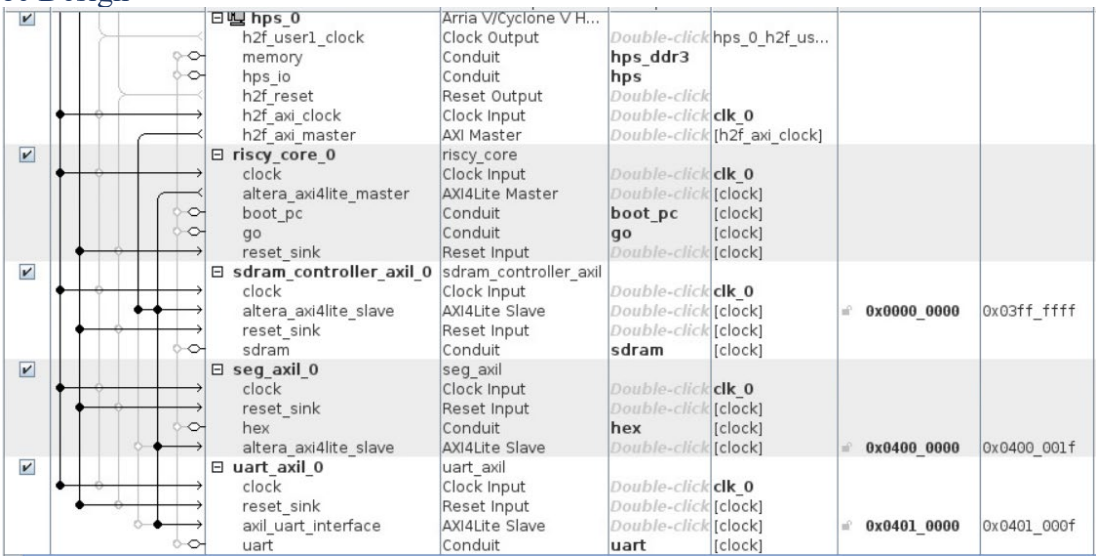


Noted that this image will probably be outdate by the time you see this, the project is constantly evolving after this course.

As shown in the image above, we are using a classical five stage pipeline design, each pipeline block is separated by a stage status register shown as long vertical rectangle above.

1. The fetch stage:
    a. Continuously fetches instruction from the system AXI bus and store the instruction inside the instruction buffer. The fetch unit fetches instruction one by one (single issue) from the instruction buffer and passes the instruction to later pipeline stage. A flush would clear the instruction buffer and set the program counter to its correct position.
2. The Decode Stage
    a. Decodes a full instruction to specific sub-control signals for later pipeline units to use
    b. Fetch operands from registor
    c. Also translates complex instruction into simpler instructions
        i. E.g. translate a atomic read to a regular read, and stall the pipeline waiting for the atomic read finish
3. The Execute stage
    a. Computes all Arithmetic operation
4. The Mem stage
    a. Handles all memory load and store operations, as well as translating the operations to AXI bus.
5. The write-back stage
    a. Writes back memory from either ALU or memory stage back to Decode stage

The Soc Design

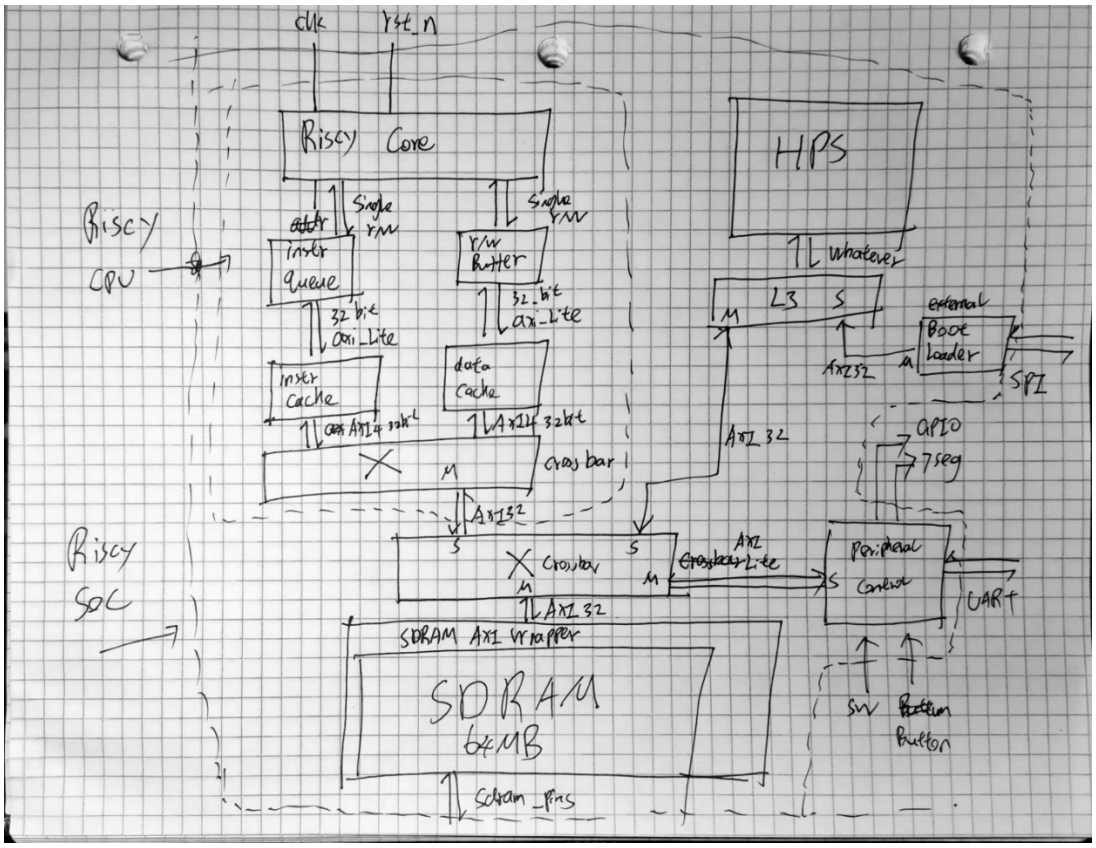


the SoC design

The SoC composed of:

1. The HPS module
    a. The hard processor system in the FPGA, acts as the bootloader and debugger in this project
    b. Issues trusted, verified, reference AXI bus operations
2. The Riscy module
    a. Clk: unified 50MHz Clock
    b. AXI4LiteMaster: unified AXI master for bus reading and writing, the instruction bus and the data bus are merged by a AXI crossbar I copied from [1]
    c. Reset_sink: unified asynchronous reset signal, active high
    d. Boot_pc: external reset vector, hard connected to the 10 switch on the FPGA board. A boot pc value of 0 means boot form the 0^{th} word in memory space (0x0000_0000), a boot pc value of 3 means boot from the 3^{rd} word in memory space (0x0000_1100)
    e. Go: the go signal, tells the core's FSM to start executing until the core runs a EBREAK instruction
3. SDRAM Controller
    a. The SDRAM Controller I copied and fine-tuned from [4]
4. Seg_axil:
    a. The AXI interface 7-seg module, functional description mentioned above
5. UART_axil
    a. The AXIL-to-UART interface, functional description mentioned above

The overall system can also be represented by this sketch:
Noted that the instruction cache and data cache are implemented but buggy, so not presented in the final demo. Their code is available in GitHub



Overall system sketch

The AXI Controller

I implemented an AXI-Lite master interface and AXI-Lite slave interface. An AXI interface is a data bus proposed by Arm aiming to provide a system fast yet simple data transition. It's the de facto bus implementation on nearly all embedded system powered by arm core.

The main difference between an AXI-Lite bus and an AXI bus is that AXI-Lite lacks burst transaction (multiple word transaction), with the trade-off of being very easy to implement and debug.

An AXI transaction must go through a number of handshakes in order, the same time, but not in inverse order

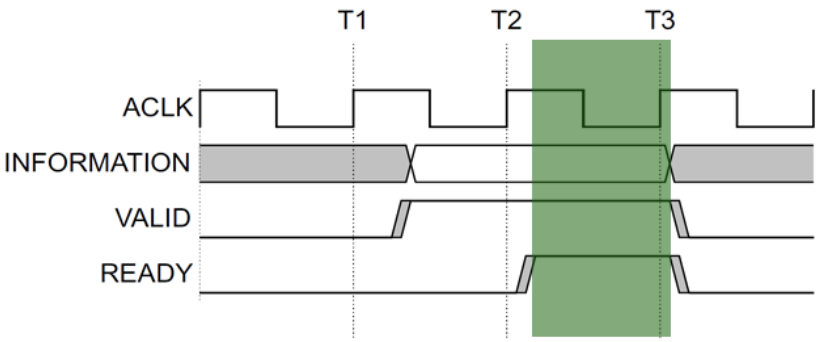
For a read it must have

1. Read addr channel handshake
2. Read data channel handshake

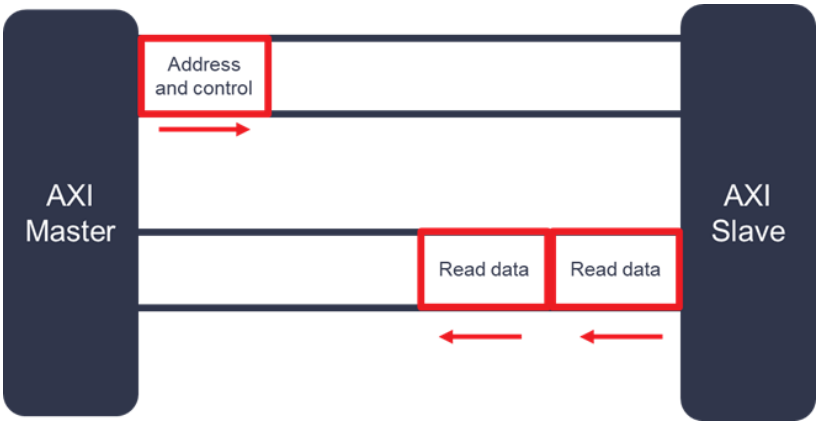
For a write it must have

1. Write addr channel handshake
2. Write data channel handshake
3. Write resp channel handshake

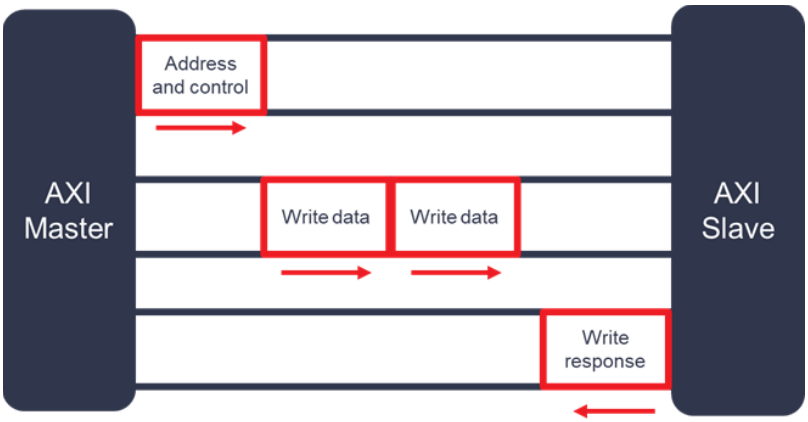
A channel handshake is occurred by both the 'valid' and 'ready' are asserted on the same positive clock edge



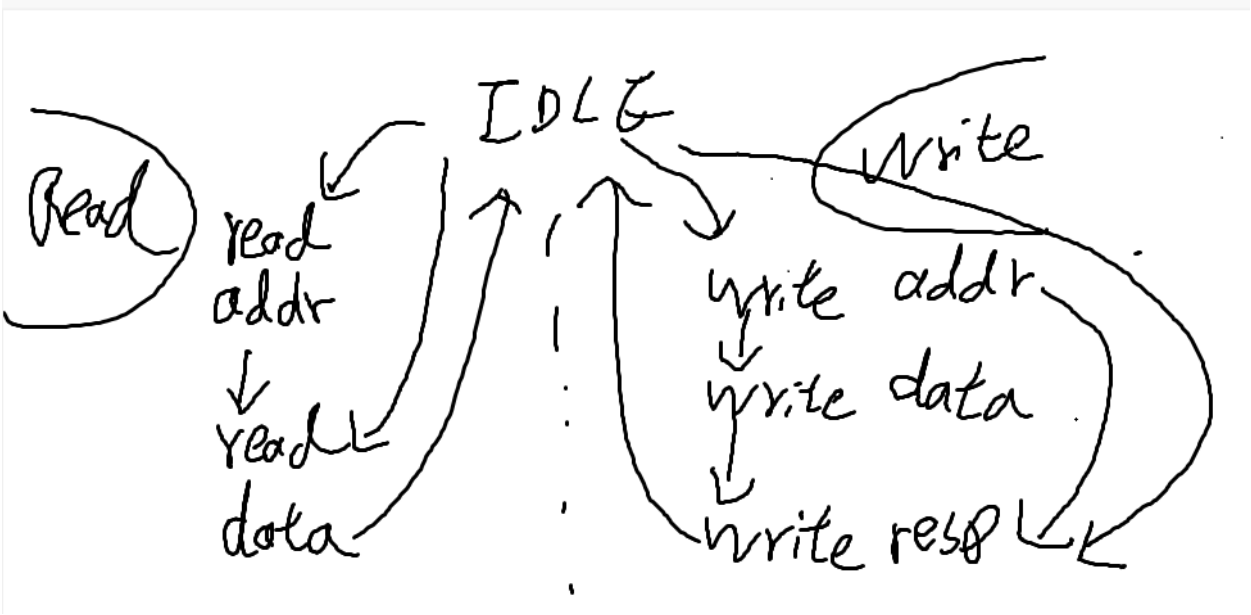
An AXI handshake



An AXI read with read address channel and read data channel



An AXI write with write addr, write data, and write resp channel
The AXI specification is rather complex and there are some edge case that needs to be considered, for more information please refer to [2] [6]
Here's a sketch of the state machine for the AXI Controller I wrote
The detailed code can be found in /rtl/mem /mem_system_axil.sv:54



A draft of the AXIL controller FSM

The bootloader:

1. The ELF file in compiler on my personal computer with the official RISC-V GNU toolchain
2. A python script is run to locate the entry point of the program
3. I manually dial the entry point using the switch on the FPGA board, encoded as 10-bit switch value
4. The ELF file is copied to the HPS using SCP
5. The bootloader on HPS calls mmap and maps the physical address of the SDRAM to the HPS user's virtual address space, mmap returns a pointer
6. The bootloader reads the ELF file word by word (32 bit) and mirror the ELF file into the SDRAM
7. The bootloader then reads the ELF file out from SDRAM and checks it there's bit error
8. I press the reset button (button[3])
9. I press the go button (button[0])
10. The core is now running the ELF file!

Epilogue

This semester has been very thrilling working with this project. I started this project around year ago and I would never have accomplished so much progress on this project without the push of this course. This report has only been a brief introduction of the project. This project is so huge that it would take hundreds of pages to specifically introduce each module' s functions. So, this report only take a glimpse of the project RISCY All the source code are on GitHub and are open source.

In the future I'm planning to change the in-order core to out-of-order core, as well as writing dedicated system call in C for it, so that I could run standard C library

Reference

0. Project RISCY, GitHub: https://github.com/Bald-Badger/riscy
1. Verilog AXI components: https://github.com/alexforencich/verilog-axi
2. AXI bus specification: https://developer.arm.com/documentation/ihi0022/e/AMBA-AXI3-and-AXI4-Protocol-Specification
3. official Risc-V GNU C toolchain: https://github.com/riscv-collab/riscv-gnu-toolchain
4. Third party SDRAM controller: https://github.com/ultraembedded/core_sdram_axi4
5. RISC-V spec: https://riscv.org/technical/specifications/
6. AXI transactions: https://support.xilinx.com/s/article/1053914?language=en_US

Code

Before looking at the boring long list of code, I would really recommend check out the GitHub Repo instead of looking at the following report: https://github.com/Bald-Badger/riscy/

the code on GitHub is really organized and well-formatted.

This have been an on-going project and have hundreds of files that I have changed or wrote this semester. I wrote around 1MB of code in plain text for this project and I tried to copy and paste the ones I actually write or changed this semester below

Soc_system_top.sv:

```
// ============================================================
// Copyright (c) 2013 by Terasic Technologies Inc.
// ============================================================
//
// Modified 2019 by Stephen A. Edwards
//
// Permission:
//
//		Terasic grants permission to use and modify this code for use
//		in synthesis for all Terasic Development Boards and Altera
//		Development Kits made by Terasic.		Other use of this code,
//		including the selling ,duplication, or modification of any
//		portion is strictly prohibited.
//
// Disclaimer:
//
//		This VHDL/Verilog or C/C++ source code is intended as a design
//		reference which illustrates how these types of functions can be
//		implemented.		It is the user's responsibility to verify their
//		design for consistency and functionality through the use of
//		formal verification methods.		Terasic provides no warranty
//		regarding the use or functionality of this code.
//
// ============================================================
//
//		Terasic Technologies Inc

//		9F., No.176, Sec.2, Gongdao 5th Rd, East Dist, Hsinchu City, 30070. Taiwan
//
//
//                                                                      web: http://www.terasic.com/
//                                                                      email: support@terasic.com
module soc_system_top (

 ///////// ADC /////////
 inout                       ADC_CS_N,
 output                      ADC_DIN,
 input                       ADC_DOUT,
 output                      ADC_SCLK,

 ///////// AUD /////////
 input                       AUD_ADCDAT,
 inout                       AUD_ADCLRCK,
 inout                       AUD_BCLK,
 output                      AUD_DACDAT,
 inout                       AUD_DACLRCK,
 output                      AUD_XCK,

 ///////// CLOCK2 /////////
 input                       CLOCK2_50,

 ///////// CLOCK3 /////////
 input                       CLOCK3_50,
```

```
//////// CLOCK4 ////////
input                       CLOCK4_50,

//////// CLOCK ////////
input                       CLOCK_50,

//////// DRAM ////////
output [12:0]       DRAM_ADDR,
output [1:0]        DRAM_BA,
output                      DRAM_CAS_N,
output                      DRAM_CKE,
output                      DRAM_CLK,
output                      DRAM_CS_N,
inout [15:0]        DRAM_DQ,
output                      DRAM_LDQM,
output                      DRAM_RAS_N,
output                      DRAM_UDQM,
output                      DRAM_WE_N,

//////// FAN ////////
output                      FAN_CTRL,

//////// FPGA ////////
output                      FPGA_I2C_SCLK,
inout                       FPGA_I2C_SDAT,

//////// GPIO ////////
inout [35:0]        GPIO_0,
inout [35:0]        GPIO_1,

//////// HEX0 ////////
output [6:0]        HEX0,

//////// HEX1 ////////
output [6:0]        HEX1,

//////// HEX2 ////////
output [6:0]        HEX2,

//////// HEX3 ////////
output [6:0]        HEX3,

//////// HEX4 ////////
output [6:0]        HEX4,

//////// HEX5 ////////
output [6:0]        HEX5,

//////// HPS ////////
inout                       HPS_CONV_USB_N,
output [14:0]       HPS_DDR3_ADDR,
output [2:0]        HPS_DDR3_BA,
output                      HPS_DDR3_CAS_N,
output                      HPS_DDR3_CKE,
output                      HPS_DDR3_CK_N,
output                      HPS_DDR3_CK_P,
output                      HPS_DDR3_CS_N,
output [3:0]        HPS_DDR3_DM,
inout [31:0]        HPS_DDR3_DQ,
inout [3:0]HPS_DDR3_DQS_N,
inout [3:0]HPS_DDR3_DQS_P,
output                      HPS_DDR3_ODT,
output                      HPS_DDR3_RAS_N,
output                      HPS_DDR3_RESET_N,
input                       HPS_DDR3_RZQ,
output                      HPS_DDR3_WE_N,
output                      HPS_ENET_GTX_CLK,
inout                       HPS_ENET_INT_N,
output                      HPS_ENET_MDC,
```

```
inout                           HPS_ENET_MDIO,
input                           HPS_ENET_RX_CLK,
input [3:0]HPS_ENET_RX_DATA,
input                           HPS_ENET_RX_DV,
output [3:0]            HPS_ENET_TX_DATA,
output                          HPS_ENET_TX_EN,
inout                           HPS_GSENSOR_INT,
inout                           HPS_I2C1_SCLK,
inout                           HPS_I2C1_SDAT,
inout                           HPS_I2C2_SCLK,
inout                           HPS_I2C2_SDAT,
inout                           HPS_I2C_CONTROL,
inout                           HPS_KEY,
inout                           HPS_LED,
inout                           HPS_LTC_GPIO,
output                          HPS_SD_CLK,
inout                           HPS_SD_CMD,
inout [3:0]HPS_SD_DATA,
output                          HPS_SPIM_CLK,
input                           HPS_SPIM_MISO,
output                          HPS_SPIM_MOSI,
inout                           HPS_SPIM_SS,
input                           HPS_UART_RX,
output                          HPS_UART_TX,
input                           HPS_USB_CLKOUT,
inout [7:0]HPS_USB_DATA,
input                           HPS_USB_DIR,
input                           HPS_USB_NXT,
output                          HPS_USB_STP,

////////// IRDA //////////
input                           IRDA_RXD,
output                          IRDA_TXD,

////////// KEY //////////
input [3:0]KEY,

////////// LEDR //////////
output [9:0]            LEDR,

////////// PS2 //////////
inout                           PS2_CLK,
inout                           PS2_CLK2,
inout                           PS2_DAT,
inout                           PS2_DAT2,

////////// SW //////////
input [9:0]SW,

////////// TD //////////
input                           TD_CLK27,
input [7:0]TD_DATA,
input                           TD_HS,
output                          TD_RESET_N,
input                           TD_VS,


////////// VGA //////////
output [7:0]            VGA_B,
output                          VGA_BLANK_N,
output                          VGA_CLK,
output [7:0]            VGA_G,
output                          VGA_HS,
output [7:0]            VGA_R,
output                          VGA_SYNC_N,
output                          VGA_VS
);

        // wire define
        logic       but_rst_n, rst_n;
```

```
logic        osc_clk, clk;
logic        locked;
logic        go;

logic [3:0] key_dbc;   // debounced key
logic [3:0] key_edg;   // debounced key
logic [3:0] key_rise;   // debounced key
logic [3:0] key_fall;   // debounced key

logic        uart_tx, uart_rx, uart_cts, uart_rts;


soc_system soc_system0(
            .clk_clk                                                    ( clk ),
            .reset_reset_n                                              ( rst_n ),
            .go_go                                                              ( go ),
            .boot_pc_boot_pc                                    ( SW ),

            .hps_ddr3_mem_a                                             ( HPS_DDR3_ADDR ),
            .hps_ddr3_mem_ba                                    ( HPS_DDR3_BA ),
            .hps_ddr3_mem_ck                                    ( HPS_DDR3_CK_P ),
            .hps_ddr3_mem_ck_n                                  ( HPS_DDR3_CK_N ),
            .hps_ddr3_mem_cke                                   ( HPS_DDR3_CKE ),
            .hps_ddr3_mem_cs_n                                  ( HPS_DDR3_CS_N ),
            .hps_ddr3_mem_ras_n                                         ( HPS_DDR3_RAS_N ),
            .hps_ddr3_mem_cas_n                                         ( HPS_DDR3_CAS_N ),
            .hps_ddr3_mem_we_n                                          ( HPS_DDR3_WE_N ),
            .hps_ddr3_mem_reset_n                               ( HPS_DDR3_RESET_N ),
            .hps_ddr3_mem_dq                                    ( HPS_DDR3_DQ ),
            .hps_ddr3_mem_dqs                                   ( HPS_DDR3_DQS_P ),
            .hps_ddr3_mem_dqs_n                                         ( HPS_DDR3_DQS_N ),
            .hps_ddr3_mem_odt                                   ( HPS_DDR3_ODT ),
            .hps_ddr3_mem_dm                                    ( HPS_DDR3_DM ),
            .hps_ddr3_oct_rzqin                                 ( HPS_DDR3_RZQ ),

            .hps_hps_io_emac1_inst_TX_CLK       ( HPS_ENET_GTX_CLK ),
            .hps_hps_io_emac1_inst_TXD0         ( HPS_ENET_TX_DATA[0] ),
            .hps_hps_io_emac1_inst_TXD1         ( HPS_ENET_TX_DATA[1] ),
            .hps_hps_io_emac1_inst_TXD2         ( HPS_ENET_TX_DATA[2] ),
            .hps_hps_io_emac1_inst_TXD3         ( HPS_ENET_TX_DATA[3] ),
            .hps_hps_io_emac1_inst_RXD0         ( HPS_ENET_RX_DATA[0] ),
            .hps_hps_io_emac1_inst_MDIO         ( HPS_ENET_MDIO ),
            .hps_hps_io_emac1_inst_MDC          ( HPS_ENET_MDC ),
            .hps_hps_io_emac1_inst_RX_CTL       ( HPS_ENET_RX_DV ),
            .hps_hps_io_emac1_inst_TX_CTL       ( HPS_ENET_TX_EN ),
            .hps_hps_io_emac1_inst_RX_CLK       ( HPS_ENET_RX_CLK ),
            .hps_hps_io_emac1_inst_RXD1         ( HPS_ENET_RX_DATA[1] ),
            .hps_hps_io_emac1_inst_RXD2         ( HPS_ENET_RX_DATA[2] ),
            .hps_hps_io_emac1_inst_RXD3         ( HPS_ENET_RX_DATA[3] ),

            .hps_hps_io_sdio_inst_CMD           ( HPS_SD_CMD ),
            .hps_hps_io_sdio_inst_D0            ( HPS_SD_DATA[0] ),
            .hps_hps_io_sdio_inst_D1            ( HPS_SD_DATA[1] ),
            .hps_hps_io_sdio_inst_CLK           ( HPS_SD_CLK ),
            .hps_hps_io_sdio_inst_D2            ( HPS_SD_DATA[2] ),
            .hps_hps_io_sdio_inst_D3            ( HPS_SD_DATA[3] ),

            .hps_hps_io_usb1_inst_D0            ( HPS_USB_DATA[0] ),
            .hps_hps_io_usb1_inst_D1            ( HPS_USB_DATA[1] ),
            .hps_hps_io_usb1_inst_D2            ( HPS_USB_DATA[2] ),
            .hps_hps_io_usb1_inst_D3            ( HPS_USB_DATA[3] ),
            .hps_hps_io_usb1_inst_D4            ( HPS_USB_DATA[4] ),
            .hps_hps_io_usb1_inst_D5            ( HPS_USB_DATA[5] ),
            .hps_hps_io_usb1_inst_D6            ( HPS_USB_DATA[6] ),
            .hps_hps_io_usb1_inst_D7            ( HPS_USB_DATA[7] ),
            .hps_hps_io_usb1_inst_CLK           ( HPS_USB_CLKOUT ),
            .hps_hps_io_usb1_inst_STP           ( HPS_USB_STP ),
            .hps_hps_io_usb1_inst_DIR           ( HPS_USB_DIR ),
            .hps_hps_io_usb1_inst_NXT           ( HPS_USB_NXT ),
```

```
        .hps_hps_io_spim1_inst_CLK          ( HPS_SPIM_CLK ),
        .hps_hps_io_spim1_inst_MOSI         ( HPS_SPIM_MOSI ),
        .hps_hps_io_spim1_inst_MISO         ( HPS_SPIM_MISO ),
        .hps_hps_io_spim1_inst_SS0          ( HPS_SPIM_SS ),

        .hps_hps_io_uart0_inst_RX           ( HPS_UART_RX ),
        .hps_hps_io_uart0_inst_TX           ( HPS_UART_TX ),

        .hps_hps_io_i2c0_inst_SDA           ( HPS_I2C1_SDAT ),
        .hps_hps_io_i2c0_inst_SCL           ( HPS_I2C1_SCLK ),

        .hps_hps_io_i2c1_inst_SDA           ( HPS_I2C2_SDAT ),
        .hps_hps_io_i2c1_inst_SCL           ( HPS_I2C2_SCLK ),

        .hps_hps_io_gpio_inst_GPIO09  ( HPS_CONV_USB_N ),
        .hps_hps_io_gpio_inst_GPIO35  ( HPS_ENET_INT_N ),
        .hps_hps_io_gpio_inst_GPIO40  ( HPS_LTC_GPIO ),

        .hps_hps_io_gpio_inst_GPIO48  ( HPS_I2C_CONTROL ),
        .hps_hps_io_gpio_inst_GPIO53  ( HPS_LED ),
        .hps_hps_io_gpio_inst_GPIO54  ( HPS_KEY ),
        .hps_hps_io_gpio_inst_GPIO61  ( HPS_GSENSOR_INT ),

        // 7-seg LEDs
        .hex_hex0                           ( HEX0 ),
        .hex_hex1                           ( HEX1 ),
        .hex_hex2                           ( HEX2 ),
        .hex_hex3                           ( HEX3 ),
        .hex_hex4                           ( HEX4 ),
        .hex_hex5                           ( HEX5 ),

        // 64MB SDRAM
        .sdram_addr                                 ( DRAM_ADDR ),
        .sdram_ba                           ( DRAM_BA ),
        .sdram_cas_n                        ( DRAM_CAS_N ),
        .sdram_cke                                  ( DRAM_CKE ),
        .sdram_clk_clk                      ( ),
        .sdram_cs_n                                 ( DRAM_CS_N ),
        .sdram_dq                           ( DRAM_DQ ),
        .sdram_dqm                                  ( {DRAM_UDQM, DRAM_LDQM} ),
        .sdram_ras_n                        ( DRAM_RAS_N ),
        .sdram_we_n                                 ( DRAM_WE_N ),

        // UART, connected to GPIO
        .uart_rx                            (uart_rx),
        .uart_tx                            (uart_tx),
        .uart_cts                           (uart_cts),  // high: master cannot take input
        .uart_rts                           (uart_rts)   // high: we cannot take input
);


// PLL module
pll     my_pll (
        .refclk                                     ( osc_clk ),
        .rst                                        ( ~but_rst_n ),
        .outclk_0                           ( clk ),
        .outclk_1                           ( DRAM_CLK ),
        .locked                                     ( locked )
);


// debounce the 4 input keys
genvar dbcr_gen;
generate
        for (dbcr_gen = 0; dbcr_gen < 4; dbcr_gen++) begin : debouncer_gen_loop
                debouncer dbc_gen (
                        .clk    (osc_clk),
                        .in             (KEY[dbcr_gen]),
                        .out    (key_dbc[dbcr_gen]),
                        .edj    (key_edg[dbcr_gen]),
```

```
                .rise           (key_rise[dbcr_gen]),
                .fall           (key_fall[dbcr_gen])
            );
        end
endgenerate


always_comb begin : ctrl_sig_assign
        but_rst_n   = key_dbc[3];           // pressed key is 0
        osc_clk             = CLOCK_50;
        rst_n               = (but_rst_n & locked);
        go                          = key_fall[0];
end


always_comb begin : gpio_0_connect
        GPIO_0 = 36'bZ;
end


always_comb begin : uart_signal_assign
        GPIO_1[1]= 1'b0;                            // connect to master GND
        uart_rx             = GPIO_1[3];            // connect to master tx
        GPIO_1[5]= uart_tx;             // connect to master rx
        uart_cts    = GPIO_1[7];            // connect to master rts
        GPIO_1[9]= uart_rts;            // connect to master cts
end


assign LEDR[0] = rst_n;
assign LEDR[1] = 1'b0;
assign LEDR[2] = 1'b0;
assign LEDR[3] = 1'b0;
assign LEDR[4] = 1'b0;
assign LEDR[5] = 1'b0;
assign LEDR[6] = 1'b0;
assign LEDR[7] = 1'b0;
assign LEDR[8] = 1'b0;
assign LEDR[9] = 1'b0;

// The following quiet the "no driver" warnings for output
// pins and should be removed if you use any of these peripherals

assign ADC_CS_N = SW[1] ? SW[0] : 1'bZ;
assign ADC_DIN = SW[0];
assign ADC_SCLK = SW[0];

assign AUD_ADCLRCK = SW[1] ? SW[0] : 1'bZ;
assign AUD_BCLK = SW[1] ? SW[0] : 1'bZ;
assign AUD_DACDAT = SW[0];
assign AUD_DACLRCK = SW[1] ? SW[0] : 1'bZ;
assign AUD_XCK = SW[0];

assign FAN_CTRL = SW[0];

assign FPGA_I2C_SCLK = SW[0];
assign FPGA_I2C_SDAT = SW[1] ? SW[0] : 1'bZ;

assign IRDA_TXD = SW[0];

assign PS2_CLK = SW[1] ? SW[0] : 1'bZ;
assign PS2_CLK2 = SW[1] ? SW[0] : 1'bZ;
assign PS2_DAT = SW[1] ? SW[0] : 1'bZ;
assign PS2_DAT2 = SW[1] ? SW[0] : 1'bZ;

assign TD_RESET_N = SW[0];

assign {VGA_R, VGA_G, VGA_B} = { 24{ SW[0] } };
assign {VGA_BLANK_N, VGA_CLK,
        VGA_HS, VGA_SYNC_N, VGA_VS} = { 5{ SW[0] } };
```

```
endmodule : soc_system_top



// synopsys translate_off
`timescale 1ns / 1ps
// synopsys translate_on

import defines::*;
import axi_defines::*;
import mem_defines::*;

module riscy_core_axil_qsys  (
        input   logic                           clk,
        input   logic                           rst,
        input   logic                           go,
        input   logic   [9:0]           boot_pc,

        /*
        * AXI lite master interface
        */
        output  logic   [31:0]          awaddr,
        output  logic   [2:0]           awprot,
        output  logic                           awvalid,
        input   logic                           awready,
        output  logic   [31:0]          wdata,
        output  logic   [3:0]           wstrb,
        output  logic                           wvalid,
        input   logic                           wready,
        input   logic   [1:0]           bresp,
        input   logic                           bvalid,
        output  logic                           bready,
        output  logic   [31:0]          araddr,
        output  logic   [2:0]           arprot,
        output  logic                           arvalid,
        input   logic                           arready,
        input   logic   [31:0]          rdata,
        input   logic   [1:0]           rresp,
        input   logic                           rvalid,
        output  logic                           rready
);

        axil_interface axil_bus ();

        proc_axil proc (
                .clk                                    (clk),
                .rst_n                                  (~rst),
                .go                                             (go),
                .boot_pc                        (boot_pc),
                .axil_bus_master        (axil_bus.axil_master)
        );


        always_comb begin : interface_linking
                awaddr                                  = axil_bus.axil_awaddr;
                awprot                                  = axil_bus.axil_awprot;
                awvalid                                 = axil_bus.axil_awvalid;
                axil_bus.axil_awready   = awready;
                wdata                                   = axil_bus.axil_wdata;
                wstrb                                   = axil_bus.axil_wstrb;
                wvalid                                  = axil_bus.axil_wvalid;
                axil_bus.axil_wready  = wready;
                axil_bus.axil_bresp             = bresp;
                axil_bus.axil_bvalid    = bvalid;
                bready                                  = axil_bus.axil_bready;
                araddr                                  = axil_bus.axil_araddr;
                arprot                                  = axil_bus.axil_arprot;
                arvalid                                 = axil_bus.axil_arvalid;
```

```
                axil_bus.axil_arready = arready;
                axil_bus.axil_rdata   = rdata;
                axil_bus.axil_rresp   = rresp;
                axil_bus.axil_rvalid  = rvalid;
                rready                = axil_bus.axil_rready;
        end

endmodule : riscy_core_axil_qsys
```

```
// reference: https://electronics.stackexchange.com/questions/505911/debounce-circuit-design-in-verilog

module debouncer
#(
	parameter MAX_COUNT = 1000000		// around 0.02s
)
(
	input logic clk,
	input logic in,				// Asynchronous and noisy input.
	output logic out,	// Debounced and filtered output.
	output logic edj,	// Goes high for 1 clk cycle on either edge of output. Note: used "edj" because "edge" is a keyword.
	output logic rise,	// Goes high for 1 clk cycle on the rising edge of output.
	output logic fall	// Goes high for 1 clk cycle on the falling edge of output.
);

	localparam COUNTER_BITS = $clog2(MAX_COUNT);

	logic in_ff, in_ff0;

	logic [COUNTER_BITS - 1 : 0] counter;
	logic w_edj;
	logic w_rise;
	logic w_fall;

	always_ff @( posedge clk ) begin : double_flop
		in_ff0 <= in;
		in_ff <= in_ff0;
	end

	always @(posedge clk)
	begin
		counter <= 0;  // Freeze counter by default to reduce switching losses when input and output are equal.
		edj <= 0;
		rise <= 0;
		fall <= 0;
		if (counter == MAX_COUNT - 1)  // If successfully debounced, notify what happened.
		begin
			out <= in_ff;
			edj <= w_edj;		// Goes high for 1 clk cycle on either edge.
			rise <= w_rise;  // Goes high for 1 clk cycle on the rising edge.
			fall <= w_fall;  // Goes high for 1 clk cycle on the falling edge.
		end
		else if (in_ff != out)  // Hysteresis.
		begin
			counter <= counter + 1;  // Only increment when input and output differ.
		end
	end

	// Edge detect.
	assign w_edj = in_ff ^ out;
	assign w_rise = in_ff & ~out;
	assign w_fall = ~in_ff & out;

endmodule : debouncer
```

```
import defines::*;

// synopsys translate_off
`timescale 1 ns / 1 ps
// synopsys translate_on

module dff_wrap #(
        WIDTH = XLEN
) (
        input   logic   clk,
        input   logic   rst_n,
        input   logic   [WIDTH-1:0]     d,
        output  logic   [WIDTH-1:0]     q
);

        reg  [WIDTH-1:0]  state;

        `ifdef SYNTHESIZE
                assign    q = state;
        `else
                assign #(1) q = state;
        `endif

        always_ff @ (posedge clk or negedge rst_n)
                if (!rst_n)
                        state <= 0;
                else
                        state <= d;

endmodule
```

```
import defines::*;

// synopsys translate_off
`timescale 1 ns / 1 ps
// synopsys translate_on

// quartus have dffe primitive but not parameterizable, i dont like it

module dffe_wrap_unsyn #(
        WIDTH = XLEN
) (
        input clk,
        input en,
        input rst_n,
        input logic[WIDTH-1:0] d,
        output logic[WIDTH-1:0] q
);

 dff_wrap #(.WIDTH(WIDTH)) dff_inst (
         // Output
        .q(q),
        // Input
        .d(({WIDTH{en}}&d) | (q&~{WIDTH{en}})),
        .clk(clk),
        .rst_n(rst_n)
);

endmodule : dffe_wrap_unsyn
```

```
import defines::*;

// synopsys translate_off
`timescale 1 ns / 1 ps
// synopsys translate_on

module dffe_wrap #(
        parameter WIDTH = XLEN,
        parameter GEN_TARGET = INDEPNDENT
) (
        input clk,
        input en,
        input rst_n,
        input logic[WIDTH-1:0] d,
        output logic[WIDTH-1:0] q
);


        genvar i;   // number of dffe
        generate
                if (GEN_TARGET == ALTERA) begin
                        for (i = 0; i < WIDTH;i++) begin : dffe_generate_loop
                                dffe dffe_gen(
                                        .d              (d[i]),
                                        .clk    (clk),
                                        .clrn   (rst_n),
                                        .prn    (1'b1),
                                        .ena    (en),
                                        .q              (q[i])
                                );
                        end
                end else if (GEN_TARGET == INDEPNDENT) begin
                        dffe_wrap_unsyn # (
                                .WIDTH  (WIDTH)
                        ) dffe (
                                .clk    (clk),
                                .en             (en),
                                .rst_n  (rst_n),
                                .d              (d),
                                .q              (q)
                        );
                end
        endgenerate

endmodule : dffe_wrap
```

```
`timescale 1ns / 1ps

// credit: https://github.com/Danishazmi29/Verilog-Code-of-Synchronus-FIFO-Design-with-verilog-test-code.git
// with modification

module fifo
	#(
		parameter  BUF_WIDTH		= 3,		// default: 2^3 = 8 entries
		parameter  DATA_WIDTH		= 32		// 32 bit fifo
	) (
	input	logic 							clk,
	input	logic 							rst,
	input	logic	[DATA_WIDTH-1:0]	buf_in,
	input	logic 							wr_en,
	input	logic 							rd_en,
	output	logic	[DATA_WIDTH-1:0]	buf_out,
	output	logic 							buf_empty,
	output	logic 							buf_full,
	output	logic 							buf_almost_full,
	output	logic	[BUF_WIDTH :0]		fifo_counter
);

localparam BUF_SIZE = (1<<BUF_WIDTH);


logic [BUF_WIDTH-1:0]		rd_ptr, wr_ptr;		// pointer to read and write addresses
logic [DATA_WIDTH-1:0]		buf_mem[0 : BUF_SIZE -1] /* synthesis ramstyle = "logic" */;


always_comb begin
	buf_empty = (fifo_counter == 0);// Checking for whether buffer is empty or not
	buf_full = (fifo_counter == BUF_SIZE);	//Checking for whether buffer is full or not
	buf_almost_full = (fifo_counter == BUF_SIZE - 1) || buf_full;
end


//Setting FIFO counter value for different situations of read and write operations.
always_ff @(posedge clk) begin
	if( rst )
		fifo_counter <= 0;					// Reset the counter of FIFO
	else if( (!buf_full && wr_en) && ( !buf_empty && rd_en ) ) //When doing read and write operation simultaneously
		fifo_counter <= fifo_counter;	// At this state, counter value will remain same.
	else if( !buf_full && wr_en )		// When doing only write operation
		fifo_counter <= fifo_counter + 1;
	else if( !buf_empty && rd_en )		//When doing only read operation
		fifo_counter <= fifo_counter - 1;
	else
	  fifo_counter <= fifo_counter;		// When doing nothing.
end


// seriously, don't touch
// my other logic expects 1 cycle delay
// my other logic expects buf_out holds until next read
always_ff @( posedge clk or posedge rst) begin
	if( rst )
		buf_out <= 0;				//On reset output of buffer is all 0.
	else begin
		if( rd_en && !buf_empty )
			buf_out <= buf_mem[rd_ptr];	//Reading the data from buffer location indicated by read pointer
		else
			buf_out <= buf_out;
	end
end


always_ff @(posedge clk) begin
	if( wr_en && !buf_full )
		buf_mem[ wr_ptr ] <= buf_in;		//Writing data input to buffer location indicated by write pointer
```

```
        else
                buf_mem[ wr_ptr ] <= buf_mem[ wr_ptr ];
end


always_ff @(posedge clk) begin
        if( rst ) begin
          wr_ptr <= 0;                  // Initializing write pointer
          rd_ptr <= 0;                  //Initializing read pointer
        end else begin
                if( !buf_full && wr_en )
                        wr_ptr <= wr_ptr + 1;           // On write operation, Write pointer points to next location
                else
                        wr_ptr <= wr_ptr;

                if( !buf_empty && rd_en )
                        rd_ptr <= rd_ptr + 1;           // On read operation, read pointer points to next location to be read
                else
                        rd_ptr <= rd_ptr;
        end
end

endmodule : fifo
```

```
/*
when en is high, io maps to o
when en is low, i maps to io and o
*/

module iobuf # (
        parameter WIDTH = 1
) (
        input                                                   en,         // 3-State enable input
        input       [WIDTH - 1 : 0] i,      // buffer input
        output      [WIDTH - 1 : 0] o,      // buffer output
        inout       [WIDTH - 1 : 0] io      // buffer inout
);

genvar index;

generate
        for (index=0; index < WIDTH; index = index + 1) begin : iobuf_gen
                bufif0 (io[index], i[index], en);
                buf (o[index], io[index]);
        end
endgenerate

endmodule
```

```
import defines::*;

module decode (
	// general
	input logic			clk,
	input logic			rst_n,

	// input
	input data_t			pc,
	input instr_t			instr,
	input data_t			wd, // write back data
	input r_t			waddr,
	input logic			wren,
	input data_t			ex_data,
	input data_t			mem_data,
	input data_t			wb_data,
	input id_fwd_sel_t	fwd_rs1,
	input id_fwd_sel_t	fwd_rs2,

	// output
	output data_t			pc_bj,
	output logic			pc_sel,
	output data_t			rs1,
	output data_t			rs2,
	output data_t			imm,
	output logic			branch_taken
);

	pc_adder pc_adder_inst (
		// input
		.instr			(instr),
		.pc				(pc),
		.rs1			(rs1),
		.rs2			(rs2),
		.ex_data		(ex_data),
		.mem_data		(mem_data),
		.wb_data		(wb_data),
		.fwd_rs1		(fwd_rs1),
		.fwd_rs2		(fwd_rs2),

		// output
		.pc_bj			(pc_bj),
		.pc_sel			(pc_sel),
		.branch_taken	(branch_taken)
	);

	registers registers_inst (
		// general
		.clk			(clk),
		.rst_n			(rst_n),

		// input
		.instr			(instr),
		.rd_data		(wd),
		.rd_wren		(wren),
		.rd_addr		(waddr),

		// output
		.rs1_data		(rs1),
		.rs2_data		(rs2)
	);

	extend extend_inst (
		// input
		.instr			(instr),

		// output
		.imm			(imm)
	);
```

```
endmodule : decode
```

```
import defines::*;

module extend (
   input     instr_t      instr,
             output       data_t       imm
);

             always_comb begin
                          imm = get_imm(instr);
             end

endmodule : extend
```

```
// computes the branch / jump target PC

import defines::*;

module pc_adder (
	input	instr_t			instr,
	input	data_t			pc,
	input	data_t			rs1,
	input	data_t			rs2,
	input	data_t			ex_data,
	input	data_t			mem_data,
	input	data_t			wb_data,
	input	id_fwd_sel_t	fwd_rs1,
	input	id_fwd_sel_t	fwd_rs2,

	output	data_t			pc_bj,
	output	logic			pc_sel,
	output	logic			branch_taken
);

	localparam taken		= 1'b1;
	localparam not_taken = 1'b0;

	opcode_t opcode;
	funct3_t funct3;

	always_comb begin
		opcode = instr.opcode;
		funct3 = instr.funct3;
	end

	data_t op1, op2;
	always_comb begin : branch_forward_mux

		case (fwd_rs1)
			RS_ID_SEL:		op1 = rs1;
			EX_ID_SEL:		op1 = ex_data;
			MEM_ID_SEL:		op1 = mem_data;
			WB_ID_SEL:		op1 = wb_data;
			default:	op1 = NULL;
		endcase

		case (fwd_rs2)
			RS_ID_SEL:		op2 = rs2;
			EX_ID_SEL:		op2 = ex_data;
			MEM_ID_SEL:		op2 = mem_data;
			WB_ID_SEL:		op2 = wb_data;
			default:	op2 = NULL;
		endcase

		/*
		op1 =	(fwd_rs1 == RS_ID_SEL)		? rs1 :
				(fwd_rs1 == EX_ID_SEL)		? ex_data :
				(fwd_rs1 == MEM_ID_SEL)	? mem_data :
				(fwd_rs1 == WB_ID_SEL)		? wb_data :
				NULL;

		op2 =	(fwd_rs2 == RS_ID_SEL)		? rs2 :
				(fwd_rs2 == EX_ID_SEL)		? ex_data :
				(fwd_rs2 == MEM_ID_SEL)	? mem_data :
				(fwd_rs2 == WB_ID_SEL)		? wb_data :
				NULL;
		*/
	end

/*
	logic [XLEN+1:0] rs_diff_unsign;
	logic [XLEN:0] rs_diff_sign;
	logic beq_take;
	logic bne_take;
```

```
        logic blt_take;
        logic bltu_take;
        logic bge_take;
        logic bgeu_take;

        always_comb begin
                rs_diff_unsign          = ({1'b0, op2} - {1'b0, op1});
                rs_diff_sign            = $signed(op2) - $signed(op1);
                beq_take                = (rs_diff_sign == 34'b0);
                bne_take                = ~beq_take;
                blt_take                = $signed(rs_diff_sign[XLEN:0]) > 0;
                bltu_take               = $signed(rs_diff_unsign[XLEN:0]) > 0;
                bge_take                = ~blt_take;
                bgeu_take               = ~bltu_take;
        end
*/

        logic beq_take;
        logic bne_take;
        logic blt_take;
        logic bltu_take;
        logic bge_take;
        logic bgeu_take;

        always_comb begin : branck_taken_assign
                beq_take   = (op1 == op2);
                bne_take   = (op1 != op2);
                blt_take   = ($signed(op1) < $signed(op2));
                bltu_take  = ($unsigned(op1) < $unsigned(op2));
                bge_take   = ($signed(op1) >= $signed(op2));
                bgeu_take  = ($unsigned(op1) >= $unsigned(op2));
        end


        always_comb begin
                branch_taken =
                        (
                                (funct3 == BEQ && beq_take)             ? taken :
                                (funct3 == BNE && bne_take)     ? taken :
                                (funct3 == BLT && blt_take)     ? taken :
                                (funct3 == BLTU && bltu_take) ? taken :
                                (funct3 == BGE && bge_take)     ? taken :
                                (funct3 == BGEU && bgeu_take)           ? taken :
                                not_taken
                        ) && (opcode == B);
        end

        /*
        possible combos:
        1. pc + imm (branch and JAL)
        2. rs1 + imm (JALR)
        */

        // for B and JAL, the imm is counted in multiple of 2 bytes
        // for JALR, the imm is counted in multuple of single byte
        data_t imm;
        always_comb begin
                imm = NULL;
                unique case (opcode)
                        B:                              imm = {{20{instr[31]}}, instr[7], instr[30:25], instr[11:8], 1'b0};
                        JAL:                    imm = {{12{instr[31]}}, instr[19:12], instr[20], instr[30:21], 1'b0};
                        JALR:                   imm = {{20{instr[31]}}, instr[31:20]};
                        default:        imm         = NULL;
                endcase
        end

        data_t pc_add, pc_add_imm, op1_add_imm;

        always_comb begin
                pc_add_imm = pc + imm;
```

```
        op1_add_imm = op1 + imm;

        pc_add =   (branch_taken)                   ? pc_add_imm  :
                                        (opcode == JAL)                 ? pc_add_imm  :
                                        (opcode == JALR)    ? op1_add_imm :
                                        NULL;

        // JALR should mask the last bit to 0
        pc_bj = (opcode == JALR) ? {pc_add[31:1], 1'b0} : pc_add;

        // 1 for branch/jump, 0 for pc + 4
        pc_sel = (branch_taken)                 ? 1'b1 :
                                (opcode == JAL)                 ? 1'b1 :
                                (opcode == JALR)    ? 1'b1 :
                                1'b0;
        // pc_nxt = (pc_sel) ? pc_bj : (pc + 32'd4);
    end

endmodule : pc_adder
```

```
import defines::*;
import mem_defines::*;

module reg_bypass (
        input   logic   clk,
        input   logic   rst_n,

        input   data_t  rd_data,
        input   logic   rd_wren,
        input   r_t             rd_addr,

        input   r_t             rs1_addr,
        input   r_t             rs2_addr,
        input   logic   rs1_rden,
        input   logic   rs2_rden,

        output  data_t  rs1_data,
        output  data_t  rs2_data
);

        logic [XLEN-1:0] registers [0:31] /* synthesis ramstyle = "logic" */;
        integer i;

        always_ff @(negedge clk or negedge rst_n) begin
                if (~rst_n) begin
                        for (i = 0; i < 32; i++) begin
                                registers[i] <= NULL;
                        end
                end else begin
                        for (i = 0; i < 32; i++) begin
                                if ((i == rd_addr) && (rd_wren)) begin
                                        registers[i] <= rd_data;
                                end else begin
                                        registers[i] <= registers[i];
                                end
                        end
                end
        end


        // bypass logic
        logic bypass_rs1;
        logic bypass_rs2;

always_comb begin
        bypass_rs1 = (rd_wren && rs1_rden) && (rs1_addr == rd_addr);
        bypass_rs2 = (rd_wren && rs2_rden) && (rs2_addr == rd_addr);

        rs1_data = rs1_addr == ZERO     ? NULL:
                                        bypass_rs1                      ? rd_data :
                                        rs1_rden                ? registers[rs1_addr]:
                                        NULL;
        rs2_data = rs2_addr == ZERO     ? NULL:
                                        bypass_rs2                      ? rd_data :
                                        rs2_rden                ? registers[rs2_addr]:
                                        NULL;
end

endmodule : reg_bypass
```

```
import defines::*;

module reg_ctrl (
	input	instr_t	instr,

	output	r_t		rs1_addr,
	output	r_t		rs2_addr,
	output	logic	rs1_rden,
	output	logic	rs2_rden
);

	opcode_t opcode;
	always_comb begin
		opcode = opcode_t'(instr.opcode);

/*
		rs1_rden = (opcode == JALR)	? ENABLE :
					(opcode == B)		? ENABLE :
					(opcode == LOAD)	? ENABLE :
					(opcode == STORE)	? ENABLE :
					(opcode == I)		? ENABLE :
					(opcode == R)		? ENABLE :
					(opcode == MEM)	? ENABLE :
					DISABLE;

		rs2_rden = (opcode == B)		? ENABLE :
					(opcode == STORE)	? ENABLE :
					(opcode == R)		? ENABLE :
					DISABLE;
*/

		case (opcode)
			JALR:		rs1_rden = ENABLE;
			B:			rs1_rden = ENABLE;
			LOAD:		rs1_rden = ENABLE;
			STORE:		rs1_rden = ENABLE;
			I:			rs1_rden = ENABLE;
			R:			rs1_rden = ENABLE;
			MEM:		rs1_rden = ENABLE;
			default:	rs1_rden = DISABLE;
		endcase

		case (opcode)
			B:			rs2_rden = ENABLE;
			STORE:		rs2_rden = ENABLE;
			R:			rs2_rden = ENABLE;
			default:	rs2_rden = DISABLE;
		endcase

		rs1_addr =	(rs1_rden) ? instr.rs1 : r_t'(ZERO); // read from X0 when no read operation
		rs2_addr =	(rs2_rden) ? instr.rs2 : r_t'(ZERO);
	end

endmodule : reg_ctrl
```

```
// register file

import defines::*;

module registers (
	input	logic	clk,
	input	logic	rst_n,

	input	data_t	instr,

	input	data_t	rd_data,
	input	logic	rd_wren,
	input	r_t		rd_addr,

	output	data_t	rs1_data,
	output	data_t	rs2_data
);

	r_t		rs1_addr, rs2_addr;
	logic	rs1_rden, rs2_rden;

	reg_bypass reg_bypass_inst (
		.clk		(clk),
		.rst_n		(rst_n),
		.rd_data	(rd_data),
		.rd_wren	(rd_wren),
		.rd_addr	(rd_addr),
		.rs1_addr	(rs1_addr),
		.rs2_addr	(rs2_addr),
		.rs1_rden	(rs1_rden),
		.rs2_rden	(rs2_rden),
		.rs1_data	(rs1_data),
		.rs2_data	(rs2_data)
	);

	reg_ctrl reg_ctrl_inst (
		.instr		(instr),
		.rs1_addr	(rs1_addr),
		.rs2_addr	(rs2_addr),
		.rs1_rden	(rs1_rden),
		.rs2_rden	(rs2_rden)
	);

endmodule : registers
```

```
package alu_defines;

`ifndef _alu_define_svh_
`define _alu_define_svh_

localparam DIV_LATENCY = 12;
localparam MUL_LATENCY = 6;

typedef enum logic[1:0] {
        a_add_b,
        a_sub_b,
        b_add_a,
        b_sub_a
} add_sub_op_t;

typedef enum logic {
        unsigned_op     = 1'b0,
        signed_op = 1'b1
} sign_t;

typedef enum logic {
        logical      = 1'b0,
        arithmetic  = 1'b1
} shift_type_t;

typedef enum logic[2:0] {
        and_result_sel,
        or_result_sel,
        xor_result_sel,
        set_result_sel,
        shift_result_sel,
        add_sub_result_sel
} result_sel_t;

typedef enum logic[2:0] {
        normal_div,
        normal_rem,
        overflow_div,
        div_by_0_div
} div_out_case_t;

`endif

endpackage : alu_defines
```

```
import defines::*;
import alu_defines::*;

module alu (
	input logic		clk,	// for multi-cycle computation
	input instr_t		instr,
	input data_t		a_in,
	input data_t		b_in,

	output data_t		c_out,
	output logic		rd_wr,
	output logic		div_result_valid,
	output logic		mul_result_valid
);



	data_t			and_result,
				or_result,
				xor_result,
				set_result,
				shift_result,
				add_sub_result,
				mult_result,
				div_rem_result;

	logic [4:0]		shamt;
	shift_type_t		shift_type;
	logic			sub_func;
	opcode_t		opcode;
	funct3_t		funct3;


	always_comb begin
		funct3 =	instr.funct3;
		opcode =	instr.opcode;
		shamt  =	instr.rs2;
		shift_type = shift_type_t'(instr[30]); // 0 for logical, 1 for arith
		sub_func = (opcode == R) & instr[30];
	end


	logic			invA, invB, plus1;		// for add/suber
	logic			set_flag;


	always_comb begin : ander
		and_result = (opcode == R) ?	a_in & b_in :
								a_in & get_imm(instr);
	end

	always_comb begin : orer
		or_result = (opcode == R) ?	a_in | b_in :
							a_in | get_imm(instr);
	end

	always_comb begin : xorer
		xor_result = (opcode == R) ?	a_in ^ b_in :
								a_in ^ get_imm(instr);
	end

	always_comb begin : seter
		set_result = set_flag ? 32'b1 : NULL;
	end

	always_comb begin : shifter
		shift_result = NULL;
		unique case ({shift_type, funct3, opcode})
			{logical, SLL, R}:	shift_result = a_in	<<	b_in[4:0];
			{logical, SRL, R}:	shift_result = a_in	>>	b_in[4:0];
```

```
                {arithmetic, SRA, R}: shift_result = $signed(a_in)      >>>     b_in[4:0];
                {logical, SLLI, I}:   shift_result = a_in                       <<      $unsigned(shamt);
                {logical, SRLI, I}:   shift_result = a_in                       >>      $unsigned(shamt);
                {arithmetic, SRAI, I}: shift_result = $signed(a_in)     >>>     $unsigned(shamt);
                default:                                        shift_result = NULL;
        endcase
end

data_t adder_in1, adder_in2;
logic [XLEN: 0]     adder_out;

logic set_signed_flag;
logic set_unsigned_flag;
//logic adder_msb;
always_comb begin : add_suber
        invA =  ((funct3 == SLT)    ? ENABLE :
                 (funct3 == SLTI)   ? ENABLE :
                 (funct3 == SLTU)   ? ENABLE :
                 (funct3 == SLTIU)  ? ENABLE :
                 DISABLE) & (opcode == I); // only I type should need to invert A;
        invB =  (funct3 == SUB & sub_func) ? ENABLE : DISABLE;

        plus1 = invA | invB;   // A - B = A + ~B + 1
        adder_in1 = invA ? ~a_in : a_in;
        // there should not be any instr in I-type that need to inv B
        // so hopefully no bug here.
        adder_in2 = (opcode == I)       ? get_imm(instr) // TODO: this line might not need
                                                                        : (invB ? ~b_in : b_in);

        adder_out = $unsigned(adder_in1) + $unsigned(adder_in2);
        add_sub_result = plus1 ? (adder_out[XLEN-1:0] + 1) : adder_out[XLEN-1:0];

        // I could not use one single adder to achieve both add, sub, and set
        set_signed_flag = ($signed(a_in) < $signed(b_in)) ? ENABLE : DISABLE;
        set_unsigned_flag = ($unsigned(a_in) < $unsigned(b_in)) ? ENABLE : DISABLE;
        set_flag =  (funct3 == SLT & set_signed_flag) ? ENABLE :
                                (funct3 == SLTU & set_unsigned_flag) ? ENABLE :
                                DISABLE;
end


/*
        // TODO: use generate on flag M_SUPPORT
        multiplier multiplierer (
                .clk        (clk),
                .instr      (instr),
                .a_in       (a_in),
                .b_in       (b_in),
                .valid      (mul_result_valid),
                .c_out      (mult_result)
        );

        // TODO: use generate on flag M_SUPPORT
        divider dividerer (
                .clk        (clk),
                .instr      (instr),
                .a_in       (a_in),
                .b_in       (b_in),
                .valid      (div_result_valid),
                .c_out      (div_rem_result)
        );
*/

        assign div_result_valid = INVALID;
        assign mul_result_valid = INVALID;
        assign mult_result = NULL;
        assign div_rem_result = NULL;

        // opcode = R, instr.funct7 = M_INSTR, div_instr&div_result_valid
        // opcode = R, instr.funct7 = M_INSTR, mul_result_vaild&~div_instr
```

```
// opcode = R, compare cout, when R, rd_wr = enable

// opcode = I, rd_wr = enable
logic div_instr;
always_comb begin : output_sel
        c_out = NULL;
        rd_wr = DISABLE;
        div_instr = funct3[2];
        unique case (opcode)
                R: begin
                        rd_wr = ENABLE;
                        unique case (instr.funct7)
                                M_INSTR: begin
                                        if (div_instr) begin     // div instruction
                                                c_out = div_rem_result;
                                        end else begin
                                                c_out = mult_result;
                                        end
                                end

                                default: begin
                                        unique case (funct3)
                                                ADD:    c_out = add_sub_result;         // same as SUB
                                                AND:    c_out = and_result;
                                                OR:     c_out = or_result;
                                                XOR:    c_out = xor_result;
                                                SLT:    c_out = set_result;
                                                SLTU:   c_out = set_result;
                                                SLL:    c_out = shift_result;
                                                SRL:    c_out = shift_result;   // same as SRA
                                                default:c_out = NULL;
                                        endcase
                                end
                        endcase
                end

                I: begin
                        rd_wr = ENABLE;
                        unique case (funct3)
                                ADDI:           c_out = add_sub_result;         // same as SUB
                                ANDI:           c_out = and_result;
                                ORI:            c_out = or_result;
                                XORI:           c_out = xor_result;
                                SLTI:           c_out = set_result;
                                SLTIU:          c_out = set_result;
                                SLLI:           c_out = shift_result;
                                SRLI:           c_out = shift_result;   // same as SRA
                                default:    c_out = NULL;
                        endcase
                end

                B: begin
                        c_out = NULL;
                        rd_wr = DISABLE;
                end

                LUI: begin
                        c_out = b_in; // should already be extended imm
                        rd_wr = ENABLE;
                end

                AUIPC: begin
                        c_out = add_sub_result;
                        rd_wr = ENABLE;
                end

                JAL: begin
                        c_out = add_sub_result;
                        rd_wr = ENABLE;
                end
```

```
                        JALR: begin
                                c_out = add_sub_result;
                                rd_wr = ENABLE;
                        end

                        LOAD: begin
                                c_out = add_sub_result;
                                rd_wr = ENABLE;
                        end

                        STORE: begin
                                c_out = add_sub_result;
                                rd_wr = DISABLE;
                        end

                        MEM: begin
                                c_out = NULL;
                                rd_wr = DISABLE;
                        end

                        SYS: begin
                                c_out = NULL;
                                rd_wr = DISABLE;
                        end

                        ATOMIC: begin
                                c_out = a_in;
                                rd_wr = ENABLE;
                        end

                        default: begin
                                c_out = NULL;
                                rd_wr = DISABLE;
                        end
                endcase
        end


/*
        opcode_t                opcode_formal;
        funct3_t                funct3_formal;
        shift_type_t            shift_type_formal;
        logic [4:0]     shamt_formal;
        logic                           sub_func_formal;

        data_t                          c_out_formal;
        logic                           rd_wr_formal;
        logic                           div_instr_formal;


        always_comb begin : formal
                opcode_formal = opcode_t'(instr[6:0]);
                funct3_formal = instr[14:12];

                shamt_formal  = instr[24:20];
                shift_type_formal = shift_type_t'(instr[30]);
                sub_func_formal = (opcode == R) & instr[30];
        end


        assert final((opcode_formal == opcode)
                        &&(funct3 == funct3_formal)
                        &&(shamt == shamt_formal)
                        &&(shift_type == shift_type_formal)
                        &&(sub_func == sub_func_formal))
        else  $display("checker failed at instr/opcode");

        logic[63:0]  mult_a_in;
        logic[63:0]  mult_b_in;
```

```
logic carry_bit, carry_bit_i;
data_t c_gold, c_gold_i;

logic[63:0]  mult_raw;

logic[31:0]  mult_result_formal;

logic mul_result_valid_formal, div_result_valid_formal;

assign mult_a_in = {{32{a_in[31]}}, {a_in[31:0]}};

assign mult_b_in = {{32{b_in[31]}}, {b_in[31:0]}};

assign mult_raw  = mult_a_in * mult_b_in;

always_comb begin: mult
        case (funct3_formal)
                MUL:            mult_result_formal = mult_raw[31:0];
                MULH:           mult_result_formal = mult_raw[63:32];
                MULHSU:                 mult_result_formal = mult_raw[63:32];
                MULHU:          mult_result_formal = mult_raw[63:32];
                default:    mult_result_formal = NULL;
        endcase
end

always_comb begin: output_sel_formal
        c_out_formal = NULL;
        rd_wr_formal = rd_wr;

        mul_result_valid_formal = mul_result_valid;

        div_result_valid_formal = div_result_valid;

        div_instr_formal = funct3_formal[2]; //for div, =1   for mult, = 0

        {carry_bit, c_gold} = a_in + b_in;

        {carry_bit_i, c_gold_i} = a_in + data_t'({ {20{instr[31]}} , instr[31:20]});

        case (opcode_formal)
                R: begin
                        case(instr[31:25]) //function 7
                                M_INSTR: begin
                                        if (~div_instr)begin
                                                c_out_formal = mult_result_formal;
                                        end else begin
                                                c_out_formal = a_in/b_in;
                                        end
                                        // assertion needs here
                                end
                                //c_out_formal use as golden value
                                default: begin
                                        case (funct3_formal)

                                                ADD:  c_out_formal = c_gold;
                                                SUB:  c_out_formal = $signed(a_in) - $signed(b_in);
                                                AND:  c_out_formal = a_in & b_in;
                                                OR:   c_out_formal = a_in | b_in;
                                                XOR:  c_out_formal = a_in ^ b_in;
                                                SLT:   c_out_formal = ($signed(a_in) < $signed(b_in)) ? 32'b1 :
32'b0;

                                                SLTU: c_out_formal = (a_in < b_in) ? 32'b1 : 32'b0;
                                                SLL:  c_out_formal = a_in << b_in[4:0];
                                                SRL:  c_out_formal = a_in >> b_in[4:0];
                                                default: c_out_formal = NULL;
                                        endcase
                                end
                        endcase
```

```
                        end

                        I: begin
                                unique case (funct3_formal)
                                        ADDI: c_out_formal = c_gold_i;
                                        ANDI: c_out_formal = a_in & (data_t'({ {20{instr[31]}} , instr[31:20]}));
                                        ORI:  c_out_formal = a_in | data_t'({ {20{instr[31]}} , instr[31:20]});
                                        XORI: c_out_formal = a_in ^ data_t'({ {20{instr[31]}} , instr[31:20]});
                                        SLTI:       c_out_formal    =    ($signed(a_in)    <    $signed(data_t'({    {20{instr[31]}}    ,
instr[31:20]}))) ? 32'b1: 32'b0;
                                        SLTIU: c_out_formal = (a_in < data_t'({ {20{instr[31]}} , instr[31:20]})) ? 32'b1: 32'b0;
                                        SLLI:  c_out_formal = a_in << instr[24:20];
                                        SRLI:  c_out_formal = a_in >> instr[24:20];
                                        default: c_out_formal = NULL;
                                endcase

                                //$diaplay("for I type, c_out is: %d", c_out);
                                //$display("for I type, c_out_formal is: %d", c_out_formal);

                                //$display("STATEMENT 1 :: time is %0t",$time);

                        end

                        LUI:begin
                                c_out_formal = {instr[31:12], 12'b0};
                        end

                        default: begin
                                c_out_formal = NULL;
                        end



                        //LOAD
                        //STORE

                endcase
        end
//assume cover

        property I_type_output;
                @(posedge clk) (opcode_formal == I) |-> ((rd_wr == rd_wr_formal)
                                        && (c_out_formal == c_out));
        endproperty

        assert property(I_type_output);

        //else begin
                //$display("I-type output value not match, time: %t", $time);
                //$display("c out is %d, formal is %d", c_out, c_out_formal);
                                        //#100
                //$stop();
        //end
        property I_type_funct3_instr;
                        @(posedge clk) (((opcode_formal == I) && ((funct3_formal == SLLI) || (funct3_formal == SRLI))) |->
                                        (instr[31:25] == 7'b0);
        endproperty

        assert property(I_type_funct3_instr);
        //assert property(@(posedge clk)(((opcode_formal == I) && ((funct3_formal == SLLI) || (funct3_formal == SRLI))) |->
                                        //(instr[31:25] == 7'b0))
        //else begin
                //$display("I-type SLLI/SRLI instr not meet requirment");
        //end
        property R_type_output_mul;
                @(posedge clk)((opcode_formal == R) && (instr[31:25] == M_INSTR) && (~div_instr)) |->
                                        ##[6:6] ((c_out_formal == c_out) && (mul_result_valid && ~div_result_valid) &&
(rd_wr));
        endproperty
```

```
        assert property(R_type_output_mul);
        //assert property(@(posedge clk)((opcode_formal == R) && (instr[31:25] == M_INSTR) && (~div_instr)) |->
                                                        //##[6:6] ((c_out_formal == c_out) && (mul_result_valid && ~div_result_valid) &&
(rd_wr)))
        //else begin
                //$display("R-type output value not match, time: %t", $time);
                //$display("c out is %d, formal is %d, mul_result_vaild is %d, div_result_valid is %d", c_out, c_out_formal, mul_result_valid,
div_result_valid);
        //end
        property R_type_output_div;
                @(posedge clk)((opcode_formal == R) && (instr[31:25] == M_INSTR) && (div_instr)) |->
                                                        ##[12:12] ((c_out_formal == c_out) && (~mul_result_valid && div_result_valid) &&
(rd_wr));
        endproperty

        assert property(R_type_output_div);

        //assert property(@(posedge clk)((opcode_formal == R) && (instr[31:25] == M_INSTR) && (div_instr)) |->
                                                        //##[12:12] ((c_out_formal == c_out) && (~mul_result_valid && div_result_valid) &&
(rd_wr)))
        //else begin
                //$display("R-type output value not match, time: %t", $time);
                //$display("c out is %d, formal is %d, mul_result_vaild is %d, div_result_valid is %d", c_out, c_out_formal, mul_result_valid,
div_result_valid);
        //end
        property R_type_output;
                        @(posedge clk)((opcode_formal == R) && (instr[31:25] != M_INSTR)) |->
                                                        ((c_out_formal == c_out) && (rd_wr));
        endproperty

        assert property(R_type_output);

        //assert property(@(posedge clk)((opcode_formal == R) && (instr[31:25] != M_INSTR)) |->
                                                        //((c_out_formal == c_out) && (rd_wr)))
        //else begin
                //$display("R-type output value not match, time: %t", $time);
                //$display("c out is %d, formal is %d, rd_wr is %d", c_out, c_out_formal, rd_wr);
        //end
        property LUI_type_output;
                @(posedge clk)(opcode_formal == LUI) |-> ((rd_wr) && (c_out_formal == c_out));
        endproperty

        assert property(LUI_type_output);


        //assert property(@(posedge clk)(opcode_formal == LUI) |-> ((rd_wr) && (c_out_formal == c_out)))
        //else begin
                //$display("LUI output not match, time: %t", $time);
                //$display("c out is %d, formal is %d, rd is %d", c_out, c_out_formal, rd_wr);
        //end
        property AUIPC_type_output;
                @(posedge clk)((opcode_formal == AUIPC) |-> (rd_wr));
        endproperty

        assert property(AUIPC_type_output);

        //assert property(@(posedge clk)((opcode_formal == AUIPC) |-> (rd_wr)))
        //else begin
                //$display("AUIPC-type output value not match, rd_wr value is %d", rd_wr);
        //end
        property JAL_type_output;
                @(posedge clk)((opcode_formal == JAL) |-> (rd_wr));
        endproperty

        assert property(JAL_type_output);

        //assert property(@(posedge clk)(opcode_formal == JAL) |-> (rd_wr))
        //else begin
                //$display("JAL-type output value not match, rd_wr value is %d", rd_wr);
        //end
```

```
property JALR_type_output;
        @(posedge clk)((opcode_formal == JALR) |-> (rd_wr));
endproperty

assert property(JALR_type_output);

//assert property(@(posedge clk)(opcode_formal == JALR) |-> (rd_wr))
//else begin
        //$display("JALR-type output value not match, rd_wr value is %d", rd_wr);
//end
property B_type_output;
        @(posedge clk)(opcode_formal == B) |->
                                        ((c_out_formal == NULL) && (~rd_wr))
endproperty

assert property(B_type_output);
//assert property(@(posedge clk)(opcode_formal == B) |->
                                        //((c_out_formal == NULL) && (~rd_wr)))
//else begin
        //$display("B-type output value not match, time: %t", $time);
        //$display("c out is %d, formal is %d, rd_wr is %d", c_out, c_out_formal, rd_wr);
//end

assert property(@(posedge clk)(opcode_formal == I) |-> ((rd_wr == rd_wr_formal)
                                        && (c_out_formal == c_out)))
else begin
        $display("I-type output value not match, time: %t", $time);
        $display("c out is %d, formal is %d", c_out, c_out_formal);
                                        //#100
        //$stop();
end

assert property(@(posedge clk)(((opcode_formal == I) && ((funct3_formal == SLLI) || (funct3_formal == SRLI)))) |->
                                        (instr[31:25] == 7'b0))
else begin
        $display("I-type SLLI/SRLI instr not meet requirment");
end

assert property(@(posedge clk)((opcode_formal == R) && (instr[31:25] == M_INSTR) && (~div_instr)) |->
                                        ##[6:6] ((c_out_formal == c_out) && (mul_result_valid && ~div_result_valid) &&
(rd_wr)))
else begin
        $display("R-type output value not match, time: %t", $time);
        $display("c out is %d, formal is %d, mul_result_vaild is %d, div_result_valid is %d", c_out, c_out_formal, mul_result_valid,
div_result_valid);
end

assert property(@(posedge clk)((opcode_formal == R) && (instr[31:25] == M_INSTR) && (div_instr)) |->
                                        ##[12:12] ((c_out_formal == c_out) && (~mul_result_valid && div_result_valid) &&
(rd_wr)))
else begin
        $display("R-type output value not match, time: %t", $time);
        $display("c out is %d, formal is %d, mul_result_vaild is %d, div_result_valid is %d", c_out, c_out_formal, mul_result_valid,
div_result_valid);
end

assert property(@(posedge clk)((opcode_formal == R) && (instr[31:25] != M_INSTR)) |->
                                        ((c_out_formal == c_out) && (rd_wr)))
else begin
        $display("R-type output value not match, time: %t", $time);
        $display("c out is %d, formal is %d, rd_wr is %d", c_out, c_out_formal, rd_wr);
end


assert property(@(posedge clk)(opcode_formal == LUI) |-> ((rd_wr) && (c_out_formal == c_out)))
else begin
        $display("LUI output not match, time: %t", $time);
        $display("c out is %d, formal is %d, rd is %d", c_out, c_out_formal, rd_wr);
end
```

```
	assert property(@(posedge clk)((opcode_formal == AUIPC) |-> (rd_wr)))
	else begin
		$display("AUIPC-type output value not match, rd_wr value is %d", rd_wr);
	end

	assert property(@(posedge clk)(opcode_formal == JAL) |-> (rd_wr))
	else begin
		$display("JAL-type output value not match, rd_wr value is %d", rd_wr);
	end

	assert property(@(posedge clk)(opcode_formal == JALR) |-> (rd_wr))
	else begin
		$display("JALR-type output value not match, rd_wr value is %d", rd_wr);
	end

	assert property(@(posedge clk)(opcode_formal == B) |->
							((c_out_formal == NULL) && (~rd_wr)))
	else begin
		$display("B-type output value not match, time: %t", $time);
		$display("c out is %d, formal is %d, rd_wr is %d", c_out, c_out_formal, rd_wr);
	end
*/

endmodule
```

```
import defines::*;
import alu_defines::*;

module divider (
        input   logic   clk,
        input   instr_t instr,
        input   data_t  a_in,
        input   data_t  b_in,

        output  logic   valid,
        output  data_t  c_out
);

logic [39:0]        divide_result;
logic [39:0]        remainder;

logic [39:0]        div_a_in, div_b_in;

data_t                      overflow_out;
data_t                      div_by_0_out;

reg [3:0]           div_counter;
logic                       div_instr;

always_comb begin : div_instr_flag_assign
        div_instr = instr.funct3[2];
end

assign valid = (div_counter == DIV_LATENCY);

always_ff @(posedge clk) begin : div_counter_update
        if (valid) begin
                div_counter <= 4'b0;   // reset counter
        end else if ( (instr.funct7 == M_INSTR) && div_instr ) begin
                div_counter <= (div_counter + 4'b1);
        end else begin
                div_counter <= 4'b0;   // reset counter
        end
end


always_comb begin : div_a_in_assign
        unique case (instr.funct3)
                DIV:    div_a_in = {{8{a_in[31]}}, a_in[31:0]};
                DIVU:   div_a_in = {8'b0, a_in[31:0]};
                REM:    div_a_in = {{8{a_in[31]}}, a_in[31:0]};
                REMU:   div_a_in = {8'b0, a_in[31:0]};
                default:div_a_in = 40'b0;
        endcase
end


always_comb begin : div_b_in_assign
        unique case (instr.funct3)
                DIV:    div_b_in = {{8{b_in[31]}}, b_in[31:0]};
                DIVU:   div_b_in = {8'b0, b_in[31:0]};
                REM:    div_b_in = {{8{b_in[31]}}, b_in[31:0]};
                REMU:   div_b_in = {8'b0, b_in[31:0]};
                default:div_b_in = 40'b0;
        endcase
end


div divide_signed_inst (
                .clock          ( clk ),
                .denom          ( div_b_in ),
                .numer          ( div_a_in ),
                .quotient   ( divide_result ),
                .remain         ( remainder )
);
```

```
div_out_case_t div_out_case, div_out_case_reg;
data_t overflow_result, div_by_0_result;
always_ff @( posedge clk ) begin : special_case_reg
	if (div_counter == 4'b0) begin
		overflow_result		<= overflow_out;
		div_by_0_result		<= div_by_0_out;
		div_out_case_reg	<= div_out_case;
	end else begin
		overflow_result		<= overflow_result;
		div_by_0_result		<= div_by_0_result;
		div_out_case_reg	<= div_out_case_reg;
	end
end


always_comb begin : c_out_assign
	unique case (div_out_case_reg)
		normal_div		: c_out = divide_result[(XLEN-1):0];
		normal_rem		: c_out = remainder[(XLEN-1):0];
		overflow_div	: c_out = overflow_result;
		div_by_0_div	: c_out = div_by_0_result;
		default			: c_out = NULL;
	endcase
end


always_comb begin : div_sel
	unique case (instr.funct3)
		DIV:	begin
			if (b_in == 0) begin
				div_out_case = div_by_0_div;
			end else if (a_in == 32'h1000_0000 && b_in == 32'hFFFF_FFFF) begin
				div_out_case = overflow_div;
			end else begin
				div_out_case = normal_div;
			end
		end

		DIVU:	begin
			if (b_in == 0) begin
				div_out_case = div_by_0_div;
			end else begin
				div_out_case = normal_div;
			end
		end

		REM:	begin
			if (b_in == 0) begin
				div_out_case = div_by_0_div;
			end else if (a_in == 32'h1000_0000 && b_in == 32'hFFFF_FFFF) begin
				div_out_case = overflow_div;
			end else begin
				div_out_case = normal_rem;
			end

		end

		REMU:	begin
			if (b_in == 0) begin
				div_out_case = div_by_0_div;
			end else begin
				div_out_case = normal_rem;
			end
		end

		default: begin
				div_out_case = normal_rem;
		end
```

```
        endcase
end


// beahvour spec: riscv-spec p45
always_comb begin : div_by_0_sel
        unique case (instr.funct3)
                DIVU:           div_by_0_out = {XLEN{1'b1}};
                REMU:           div_by_0_out = a_in;
                DIV:            div_by_0_out = {XLEN{1'b1}};
                REM:            div_by_0_out = a_in;
                default:    div_by_0_out = NULL;
        endcase
end


always_comb begin : overflow_sel
        unique case (instr.funct3)
                DIV:            overflow_out = {{1'b1}, {(XLEN-1){1'b0}}};
                REM:            overflow_out = 1'b0;
                default:    overflow_out = NULL;
        endcase
end

endmodule : divider
```

```
import defines::*;

module ex_mux (
	input instr_t			instr,
	input data_t			pc,
	input data_t			rs1,
	input data_t			rs2,
	input data_t			imm,
	input data_t			ex_ex_fwd_data,
	input data_t			mem_ex_fwd_data,

	input ex_fwd_sel_t	fwd_a,
	input ex_fwd_sel_t	fwd_b,

	output data_t			a_out,
	output data_t			b_out
);

	// rs1_mux, rs2_mux ctrl signal types
	localparam null_sel = 2'b00;
	localparam rs1_sel	= 2'b10;
	localparam pc_sel	= 2'b11;
	localparam rs2_sel	= 2'b10;
	localparam imm_sel	= 2'b11;


	logic [1:0] rs1_mux_sel, rs2_mux_sel;
	data_t rs1_mux_out, rs2_mux_out;

	always_comb begin : rs1_mux
		unique case (rs1_mux_sel)
			null_sel:	rs1_mux_out = NULL;
			rs1_sel:	rs1_mux_out = rs1;
			pc_sel:			rs1_mux_out = pc;
			default:	rs1_mux_out = NULL;
		endcase
	end


	always_comb begin : rs2_mux
		unique case (rs2_mux_sel)
			null_sel:	rs2_mux_out = NULL;
			rs2_sel:	rs2_mux_out = rs2;
			imm_sel:	rs2_mux_out = imm;
			default:	rs2_mux_out = NULL;
		endcase
	end


	always_comb begin : fwd_a_mux
		unique case (fwd_a)
			RS_EX_SEL:			a_out = rs1_mux_out;
			MEM_EX_SEL:			a_out = ex_ex_fwd_data;
			WB_EX_SEL:			a_out = mem_ex_fwd_data;
			default:		a_out = NULL;
		endcase
	end


	always_comb begin : fwd_b_mux
		unique case (fwd_b)
			RS_EX_SEL:			b_out = rs2_mux_out;
			MEM_EX_SEL:			b_out = ex_ex_fwd_data;
			WB_EX_SEL:			b_out = mem_ex_fwd_data;
			default:		b_out = NULL;
		endcase
	end
```

```
	always_comb begin : rs_mux_sel_ctrl
		rs1_mux_sel = 2'b00;
		rs2_mux_sel = 2'b00;
		unique case (instr.opcode)
			R: begin //DONE
				rs1_mux_sel = rs1_sel;
				rs2_mux_sel = rs2_sel;
			end

			I: begin //DONE
				rs1_mux_sel = rs1_sel;
				rs2_mux_sel = imm_sel;
			end

			B: begin
				rs1_mux_sel = rs1_sel;
				rs2_mux_sel = rs2_sel;
			end

			LUI: begin// imm + 0
				rs1_mux_sel = null_sel;
				rs2_mux_sel = imm_sel;
			end

			AUIPC: begin		// pc + imm
				rs1_mux_sel = pc_sel;
				rs2_mux_sel = imm_sel;
			end

			JAL: begin// pc + imm, imm = 4
				rs1_mux_sel = pc_sel;
				rs2_mux_sel = imm_sel;
			end

			JALR: begin		// pc + imm, imm = 4
				rs1_mux_sel = pc_sel;
				rs2_mux_sel = imm_sel;
			end

			LOAD: begin		// rs1 + imm DONE
				rs1_mux_sel = rs1_sel;
				rs2_mux_sel = imm_sel;
			end

			STORE: begin // rs1 + imm DONE
				rs1_mux_sel = rs1_sel;
				rs2_mux_sel = imm_sel;
			end

			MEM: begin		// riscv-spec.pdf page 27, ignore rs and rd
				rs1_mux_sel = null_sel;
				rs2_mux_sel = null_sel;
			end

			SYS: begin		// no alu need
				rs1_mux_sel = null_sel;
				rs2_mux_sel = null_sel;
			end

			default: begin
				rs1_mux_sel = null_sel;
				rs2_mux_sel = null_sel;
			end
		endcase
	end


/*
	opcode_t		opcode_formal;
```

```
always_comb begin: formal
	opcode_formal = opcode_t'(instr[6:0]);
end

always_comb begin: a_out_formal
	ex_mux_a_out_general: assert ((a_out == NULL)
                                                                || (a_out == rs1_mux_out) ||
                                                                (a_out == ex_ex_fwd_data) ||
                                                                (a_out == mem_ex_fwd_data)) begin
		$display("ex-mux: a_out general case pass");
	end else begin
		$display("WB: a_out general case doesn't pass");
	end

end //automatic

always_comb begin: b_out_formal
	ex_mux_b_out_general: assert ((b_out == NULL)
                                                                || (b_out == rs2_mux_out) ||
                                                                (b_out == ex_ex_fwd_data) ||
                                                                (b_out == mem_ex_fwd_data)) begin
		$display("ex-mux: b_out general case pass");
	end else begin
		$display("WB: b_out general case doesn't pass");
	end

end //automatic

always_comb begin: rs1_mux_out_formal
	ex_mux_rs1_out_general: assert ((rs1_mux_out == NULL)
                                                                ||(rs1_mux_out == rs1) ||
                                                                        (rs1_mux_out == pc)) begin
		$display("ex-mux: ex_mux_rs1_out_general case pass");
	end else begin
		$display("WB: ex_mux_rs1_out_general case doesn't pass");
	end

end //automatic

always_comb begin: rs2_mux_out_formal
	ex_mux_rs2_out_general: assert ((rs2_mux_out == NULL)
                                                                ||(rs2_mux_out == rs2) ||
                                                                        (rs2_mux_out == imm)) begin
		$display("ex-mux: ex_mux_rs2_out_general general case pass");
	end else begin
		$display("WB: ex_mux_rs2_out_general case doesn't pass");
	end

end //automatic

always_comb begin: opcode_rs1_mux_out_formal
	case (opcode_formal)
		R: begin
			R_rs1_mux_output: assert (rs1_mux_out == rs1) begin
				$display("ex-mux: R type rs1_mux_output pass");
			end else begin
				$display("ex-mux: R type rs1_mux_output don't pass");
			end
		end

		I: begin
			I_rs1_mux_output: assert (rs1_mux_out == rs1) begin
				$display("ex-mux: I type rs1_mux_output pass");
			end else begin
				$display("ex-mux: I type rs1_mux_output don't pass");
			end
		end

		B: begin
			B_rs1_mux_output: assert (rs1_mux_out == rs1) begin
```

```
                $display("ex-mux: B type rs1_mux_output pass");
            end else begin
                $display("ex-mux: B type rs1_mux_output don't pass");
            end
end

LUI: begin
            LUI_rs1_mux_output: assert (rs1_mux_out == NULL) begin
                $display("ex-mux: LUI type rs1_mux_output pass");
            end else begin
                $display("ex-mux: LUI type rs1_mux_output don't pass");
            end
end

AUIPC: begin
            AUIPC_rs1_mux_output: assert (rs1_mux_out == pc) begin
                $display("ex-mux: AUIPC type rs1_mux_output pass");
            end else begin
                $display("ex-mux: AUIPC type rs1_mux_output don't pass");
            end
end

JAL: begin
            JAL_rs1_mux_output: assert (rs1_mux_out == pc) begin
                $display("ex-mux: JAL type rs1_mux_output pass");
            end else begin
                $display("ex-mux: JAL type rs1_mux_output don't pass");
            end
end

JALR: begin
            JALR_rs1_mux_output: assert (rs1_mux_out == pc) begin
                $display("ex-mux: JALR type rs1_mux_output pass");
            end else begin
                $display("ex-mux: JALR type rs1_mux_output don't pass");
            end
end

LOAD: begin
            LOAD_rs1_mux_output: assert (rs1_mux_out == rs1) begin
                $display("ex-mux: LOAD type rs1_mux_output pass");
            end else begin
                $display("ex-mux: LOAD type rs1_mux_output don't pass");
            end
end

STORE: begin
            STORE_rs1_mux_output: assert (rs1_mux_out == rs1) begin
                $display("ex-mux: STORE type rs1_mux_output pass");
            end else begin
                $display("ex-mux: STORE type rs1_mux_output don't pass");
            end
end

MEM: begin
            MEM_rs1_mux_output: assert (rs1_mux_out == NULL) begin
                $display("ex-mux: MEM type rs1_mux_output pass");
            end else begin
                $display("ex-mux: MEM type rs1_mux_output don't pass");
            end
end

SYS: begin
            SYS_rs1_mux_output: assert (rs1_mux_out == NULL) begin
                $display("ex-mux: SYS type rs1_mux_output pass");
            end else begin
                $display("ex-mux: SYS type rs1_mux_output don't pass");
            end
end
```

```
		endcase
	end

	always_comb begin: opcode_rs2_mux_out_formal
		case (opcode_formal)
			R: begin
				R_rs2_mux_output: assert (rs2_mux_out == rs2) begin
					$display("ex-mux: R type rs2_mux_output pass");
				end else begin
					$display("ex-mux: R type rs2_mux_output don't pass");
				end
			end

			I: begin
				I_rs2_mux_output: assert (rs2_mux_out == imm) begin
					$display("ex-mux: I type rs2_mux_output pass");
				end else begin
					$display("ex-mux: I type rs2_mux_output don't pass");
				end
			end

			B: begin
				B_rs2_mux_output: assert (rs2_mux_out == rs2) begin
					$display("ex-mux: B type rs2_mux_output pass");
				end else begin
					$display("ex-mux: B type rs2_mux_output don't pass, rs2_mux_out %d, rs2 %d",
rs2_mux_out, rs2);
				end
			end

			LUI: begin
				LUI_rs2_mux_output: assert (rs2_mux_out == imm) begin
					$display("ex-mux: LUI type rs1_mux_output pass");
				end else begin
					$display("ex-mux: LUI type rs1_mux_output don't pass");
				end
			end

			AUIPC: begin
				AUIPC_rs2_mux_output: assert (rs2_mux_out == imm) begin
					$display("ex-mux: AUIPC type rs2_mux_output pass");
				end else begin
					$display("ex-mux: AUIPC type rs2_mux_output don't pass");
				end
			end

			JAL: begin
				JAL_rs2_mux_output: assert ((rs2_mux_out == imm) && (imm == 4)) begin
					$display("ex-mux: JAL type rs2_mux_output pass");
				end else begin
					$display("ex-mux: JAL type rs2_mux_output don't pass");
				end
			end

			JALR: begin
				JALR_rs2_mux_output: assert ((rs2_mux_out == imm) && (imm == 4)) begin
					$display("ex-mux: JALR type rs2_mux_output pass");
				end else begin
					$display("ex-mux: JALR type rs2_mux_output don't pass");
				end
			end

			LOAD: begin
				LOAD_rs2_mux_output: assert (rs2_mux_out == imm) begin
					$display("ex-mux: LOAD type rs2_mux_output pass");
				end else begin
					$display("ex-mux: LOAD type rs2_mux_output don't pass");
				end
			end
```

```
                                STORE: begin
                                        STORE_rs2_mux_output: assert (rs2_mux_out == imm) begin
                                                $display("ex-mux: STORE type rs2_mux_output pass");
                                        end else begin
                                                $display("ex-mux: STORE type rs2_mux_output don't pass");
                                        end
                                end

                                MEM: begin
                                        MEM_rs2_mux_output: assert (rs2_mux_out == NULL) begin
                                                $display("ex-mux: MEM type rs2_mux_output pass");
                                        end else begin
                                                $display("ex-mux: MEM type rs2_mux_output don't pass");
                                        end
                                end

                                SYS: begin
                                        SYS_rs2_mux_output: assert (rs2_mux_out == NULL) begin
                                                $display("ex-mux: SYS type rs2_mux_output pass");
                                        end else begin
                                                $display("ex-mux: SYS type rs2_mux_output don't pass");
                                        end
                                end

                        endcase
                end
*/

endmodule : ex_mux
```

```
import defines::*;

module execute (
        input   logic                   clk,

        // ctrl
        input   ex_fwd_sel_t    fwd_a,
        input   ex_fwd_sel_t    fwd_b,

        // input
        input   data_t                  rs1,
        input   data_t                  pc,
        input   data_t                  rs2,
        input   data_t                  imm,
        input   instr_t                 instr,

        // fwd data
        input   data_t                  ex_ex_fwd_data,
        input   data_t                  mem_ex_fwd_data,

        // output
        output  data_t                  alu_result,
        output  logic                   rd_wren,
        output  logic                   execute_busy
);

        data_t  a, b;
        logic   div_result_valid, mul_result_valid;
        logic   div_busy, mul_busy;
        logic   div_instr, mul_instr;

        always_comb begin : execute_busy_assign
                div_instr       =       (instr.funct3[2]);
                div_busy        =       (instr.funct7 == M_INSTR) &&
                                                (div_instr) &&
                                                (~div_result_valid) &&
                                                (instr.opcode == R);
                mul_instr       =       ~div_instr;
                mul_busy        =       (instr.funct7 == M_INSTR) &&
                                                (mul_instr) &&
                                                (~mul_result_valid) &&
                                                (instr.opcode == R);
                execute_busy    =       mul_busy || div_busy;
        end


        ex_mux ex_mux_inst (
                // input
                .instr                  (instr),
                .pc                             (pc),
                .rs1                    (rs1),
                .rs2                    (rs2),
                .imm                    (imm),
                .ex_ex_fwd_data         (ex_ex_fwd_data),
                .mem_ex_fwd_data        (mem_ex_fwd_data),
                .fwd_a                  (fwd_a),
                .fwd_b                  (fwd_b),

                // output
                .a_out                  (a),
                .b_out                  (b)
        );


        alu alu_inst (
                // clk for multi-cycle computation
                .clk                    (clk),

                // input
```

```
        .instr                  (instr),
        .a_in                   (a),
        .b_in                   (b),

        // output
        .c_out                  (alu_result),
        .rd_wr                  (rd_wren),
        .div_result_valid       (div_result_valid),
        .mul_result_valid       (mul_result_valid)
);

/*
property mul_div_not_equal;
        @(posedge clk) ((instr.opcode == R) && (instr.funct7 == M_INSTR)) |-> (div_result_valid && (~mul_result_valid))
endproperty

assert property (mul_div_not_equal) //make sure div_result_valid is not equal when multly and divide
else begin
        $display("property mul and div not match");
end;

property property_ex_busy;
        @(posedge clk) ((instr.opcode !=R) |-> (execute_busy == 0))
endproperty

assert property (property_ex_busy) //make sure when instr.opcode !=R, exexcute_busy will keep 0
else begin
        $display("property_ex_busy not match");
end;

property property_ex_busy_div;
        @(posedge clk) (((instr.opcode == R) && (instr.funct7 == M_INSTR) && (instr.funct3[2] == 1)) |->
                                                ##[0:11] (execute_busy == 1))
endproperty

assert property(property_ex_busy_div) //make sure when divide, under 12 cycles, exectue_busy is always 1
else begin
        $display("property_ex_busy_div not match, execute_busy is %d", execute_busy);
end;

property property_ex_busy_mul;
        @(posedge clk) (((instr.opcode == R) && (instr.funct7 == M_INSTR) && (instr.funct3[2] == 0)) |->
                                                ##[0:5] (execute_busy == 1))
endproperty

assert property(property_ex_busy_mul) //make sure when multiply, under 6 cycles, exectue_busy is always 1
else begin
        $display("property_ex_busy_mul not match, execute_busy is %d", execute_busy);
end;

*/

endmodule : execute
```

```
import defines::*;
import alu_defines::*;

module multiplier (
	input	logic	clk,
	input	instr_t	instr,
	input	data_t	a_in,
	input	data_t	b_in,

	output	logic	valid,
	output	data_t	c_out
);

logic [79:0] mult_result;
logic [39:0] mult_a_in, mult_b_in;


reg [3:0]	mul_counter;
logic		mul_instr;

always_comb begin : mul_instr_flag_assign
	mul_instr = ~instr.funct3[2];
end


assign valid = (mul_counter == MUL_LATENCY);

always_ff @(posedge clk) begin : mul_counter_update
	if (valid) begin
		mul_counter <= 4'b0; // reset counter
	end else if ( (instr.funct7 == M_INSTR) && mul_instr ) begin
		mul_counter <= (mul_counter + 4'b1);
	end else begin
		mul_counter <= 4'b0; // reset counter
	end
end


always_comb begin : mult_a_in_assign
	unique case (instr.funct3)
		MUL:		mult_a_in = {{8{a_in[31]}}, a_in[31:0]};
		MULH:		mult_a_in = {{8{a_in[31]}}, a_in[31:0]};
		MULHSU:		mult_a_in = {{8{a_in[31]}}, a_in[31:0]};
		MULHU:		mult_a_in = {8'b0, a_in[31:0]};
		default:	mult_a_in = 40'b0;
	endcase
end


always_comb begin : mult_b_in_assign
	unique case (instr.funct3)
		MUL:		mult_b_in = {{8{b_in[31]}}, b_in[31:0]};
		MULH:		mult_b_in = {{8{b_in[31]}}, b_in[31:0]};
		MULHSU:		mult_b_in = {8'b0, b_in[31:0]};
		MULHU:		mult_b_in = {8'b0, b_in[31:0]};
		default:	mult_b_in = 40'b0;
	endcase
end


mult mult_signed_inst (
		.clock	( clk ),
		.dataa	( mult_a_in ),
		.datab	( mult_b_in ),
		.result	( mult_result )
);


always_comb begin : mult_sel
	unique case (instr.funct3)
```

```
                MUL:            c_out = mult_result[31:0];
                MULH:           c_out = mult_result[63:32];
                MULHSU:                 c_out = mult_result[63:32];
                MULHU:          c_out = mult_result[63:32];
                default:    c_out = NULL;
        endcase
end

endmodule : multiplier
```

```
import defines::*;

module branch_predict (
        input   instr_t     instr,
        output  logic       taken
);

        always_comb begin : predictor
                unique case (PREDICTOR)
                        P_TAKEN:        taken = TAKEN;
                        P_N_TAKEN:      taken = NOT_TAKEN;
                        BTFNT:          taken =     (instr.opcode != B)    ? NOT_TAKEN :     // no action if not branch instrution
                                                                           (instr[XLEN-1])        ? TAKEN :
        // PC pffset is negative value, branch take
                                                                           NOT_TAKEN;
                        // PC offset is positive value, branch (not) take
                        default:    taken = NOT_TAKEN;
                endcase
        end

endmodule : branch_predict
```

```
// Seriously, this code is a shit-whole
// Rewrite this mudule with better FSM in the future

import defines::*;
import axi_defines::*;

module fetch_axil # (
	parameter INSTR_QUE_ADDR_WIDTH = 4
) (
	// general input
	input	logic 					clk,
	input	logic 					rst_n,

	// input
	input	data_t 					pc_bj,
	input	logic 					pc_sel,
	input	logic 					stall,
	input	logic 					flush,
	input	logic 					go,
	input	logic [9:0]			boot_pc_extrn,
	input	instr_t 				instr_w,

	// output
	output	data_t 					pc_out,
	output	instr_t 				instr,
	output	logic 					instr_valid,

	// AXI Lite bus interface
	axil_interface.axil_master		axil_bus
);

	// state explained in latter FSM logic
	typedef enum logic[2:0] {
		DEBUG,
		FETCH,
		STALL,
		EBREAK_WAIT
	} state_t;

	state_t state, nxt_state;

	always_ff @(posedge clk or negedge rst_n) begin
		if (~rst_n)
			state <= DEBUG;
		else
			state <= nxt_state;
	end

	typedef struct packed {
		data_t	instr;
		data_t	pc;
	} instr_queue_entry_t;

	logic [INSTR_QUE_ADDR_WIDTH:0] fifo_counter;
	logic ebreak, ebreak_clear;
	logic done;

	logic buf_empty, buf_full, buf_almost_full;

	logic ifu_rden;
	logic ifu_valid;

	// pc control logic
	data_t pc, pc_p4;

	assign pc_p4 = pc + 32'd4;

	logic pc_en;
	logic update_pc;
	assign update_pc = done;
```

```
always_ff @(posedge clk or negedge rst_n) begin
        if (~rst_n)
                pc_en <= CLEAR;
        else if (go)
                pc_en <= SET;
        else if (state == EBREAK_WAIT)
                pc_en <= CLEAR;
        else if (state == DEBUG)
                pc_en <= CLEAR;
        else
                pc_en <= pc_en;
end


data_t pc_nxt, pc_bj_ff;
logic flush_flag, flush_flag_delay;

always_ff @(posedge clk or negedge rst_n) begin
        if (~rst_n)
                flush_flag <= CLEAR;
        else if (flush && ~done)
                flush_flag <= SET;
        else if (done)
                flush_flag <= CLEAR;
        else
                flush_flag <= flush_flag;
end

always_ff @(posedge clk) begin
        flush_flag_delay <= flush_flag;
end

always_ff @(posedge clk or negedge rst_n) begin
        if (~rst_n)
                pc_bj_ff <= NULL;
        else if (pc_sel)
                pc_bj_ff <= pc_bj;
        else
                pc_bj_ff <= pc_bj_ff;
end


always_comb begin
        if (done && flush)
                pc_nxt = pc_bj;
        else
                pc_nxt = flush_flag ? pc_bj_ff : pc_p4;
end


always_ff @(posedge clk or negedge rst_n) begin
        if (~rst_n) begin
                pc <= (boot_pc_extrn * 4);
        end else if (pc_en && update_pc) begin
                pc <= pc_nxt;
        end else begin
                pc <= pc;
        end
end
// end pc control logic


// instruction wires
instr_queue_entry_t instr_fifo_in, instr_fifo_out;
data_t instr_mem_sys;
data_t instr_plain;
data_t instr_switch;    // switch endianess
logic instr_valid_early;
assign instr_plain = data_t'(instr_mem_sys);
```

```
assign instr_fifo_in = instr_queue_entry_t'({instr_mem_sys, pc});

// BUG: assert ebreak = instr_d == EBREAK
assign ebreak =         (ENDIANESS == BIG_ENDIAN) ?
                                        (instr_plain == EBREAK) :
                                        (swap_endian(instr_plain) == EBREAK);
assign ebreak_clear = (ENDIANESS == LITTLE_ENDIAN) ?
                                                                        (data_t'(instr_w) == EBREAK) :
                                                                        (swap_endian(data_t'(instr_w)) == EBREAK);
assign instr_valid_early = ((~buf_empty) && (~stall) && (~flush) && (~flush_flag) && (~flush_flag_delay));
// end instruction wires

data_t instr_debug;
assign instr_debug = data_t'(instr);

// delay instr_valid for one cycle because fifo have 1 cycle read delay
always_ff @( posedge clk, negedge rst_n ) begin
        if (~rst_n)
                instr_valid <= 1'b0;
        else if (flush)
                instr_valid <= 1'b0;
        else if (flush_flag_delay)
                instr_valid <= 1'b0;
        else if (stall)
                instr_valid <=instr_valid;
        else
                instr_valid <= instr_valid_early;
end


always_comb begin
        nxt_state                       = DEBUG;
        ifu_rden                        = DISABLE;
        ifu_valid                       = INVALID;
        unique case (state)

                // init state, the CPU stall due to a EBREAK instruction
                // hand over the control flow to debugger
                DEBUG: begin
                        if (go) begin
                                nxt_state   = FETCH;
                        end else begin
                                nxt_state   = DEBUG;
                        end
                end

                FETCH: begin
                                ifu_rden = ENABLE;
                                ifu_valid = VALID;
                        if (done && buf_almost_full) begin
                                nxt_state   = STALL;
                        end else if (done && ebreak) begin
                                nxt_state   = EBREAK_WAIT;
                        end else begin
                                nxt_state   = FETCH;
                        end
                end

                // instruction issue almoat full, stall fetch
                STALL: begin
                        if (~buf_almost_full) begin
                                nxt_state   = FETCH;
                        end else begin
                                nxt_state   = STALL;
                        end
                end

                EBREAK_WAIT: begin
                        if (ebreak_clear) begin
                                nxt_state   = DEBUG;
```

```
                                        end else begin
                                                nxt_state   = EBREAK_WAIT;
                                        end
                                end

                                default: begin
                                        nxt_state   = DEBUG;
                                end
                        endcase
                end


                always_comb begin : switch_endian
                        instr_switch =          (ENDIANESS == BIG_ENDIAN) ? instr_t'(instr_fifo_out.instr) :
                                instr_t'(swap_endian(data_t'(instr_fifo_out.instr)));
                end


                always_comb begin
                        instr = instr_valid ? instr_switch : NOP;
                        pc_out = instr_valid ? instr_fifo_out.pc : NULL;
                end


                mem_sys_axil_wrapper instr_fetcher (
                        .clk                    (clk),
                        .rst_n                  (rst_n),

                        .addr                   (pc),
                        .data_in        (NULL),
                        .wr                             (DISABLE),
                        .rd                             (ifu_rden),
                        .valid                  (ifu_valid),
                        .be                             (4'b1111),

                        .data_out       (instr_mem_sys),
                        .done                   (done),

                        .axil_bus       (axil_bus)
                );


                fifo #(
                        .BUF_WIDTH                      (INSTR_QUE_ADDR_WIDTH),
                        .DATA_WIDTH                     (XLEN * 2)              // fits both instr and pc
                ) instr_queue (
                        .clk                            (clk),
                        .rst                            ((~rst_n) || flush),
                        .buf_in                         (instr_fifo_in),
                        .buf_out                (instr_fifo_out),
                        .wr_en                          (done),
                        .rd_en                          (~stall),
                        .buf_empty                      (buf_empty),
                        .buf_full               (buf_full),
                        .buf_almost_full        (buf_almost_full),
                        .fifo_counter           (fifo_counter)
                );

endmodule : fetch_axil
```

```
import defines::*;

module fetch (
	// general input
	input	logic	clk,
	input	logic	rst_n,

	// input
	input	data_t	pc_bj,
	input	logic	pc_sel,
	input	logic	stall,
	input	logic	flush,
	input	logic	go,
	input	logic	mispredict,

	// output
	output	data_t	pc_p4_out,
	output	data_t	pc_out,
	output	instr_t	instr,
	output	logic	taken,
	output	logic	instr_valid
);
	instr_t instr_raw;
	assign instr =		(~instr_valid)		? NOP :
								(flush)			? NOP : // mask the output as if flushed
												instr_raw;

	always_ff @(posedge clk or negedge rst_n) begin
		if (~rst_n)
			instr_valid <= INVALID;
		else
			instr_valid <= go;
	end

	data_t pc, pc_p4;

	always_ff @(posedge clk) begin : pc_delay
		if (stall)
			pc_out <= pc_out;
		else
			pc_out <= pc;
	end

	assign pc_p4_out = pc_out + 4;

	pc pc_inst (
		// input
		.clk	(clk),
		.rst_n	(rst_n),
		.pc_bj	(pc_bj),
		.pc_sel	(pc_sel),
		.stall	(stall),

		// output
		.pc			(pc),
		.pc_p4	(pc_p4)
	);


	instr_mem instr_mem_inst (
		.clk	(clk),
		.rst_n	(rst_n),
		.rden	(~flush),
		.stall	(stall),
		.addr	(pc),
		.instr	(instr_raw)
	);


	// not implemented yet
	branch_predict branch_predictor (
```

```
                .instr      (instr),
                .taken      (taken)
        );

endmodule : fetch
```

```
module axi_crossbar_2x1_wrapper (
	input logic clk,
	input logic rst,
	axi_interface s00,
	axi_interface s01,
	axi_interface m00
);

	axi_crossbar_2x1 crossbar (
		.clk                    (clk),
		.rst                    (rst),

		// s00
		.s00_axi_awid           (s00.s_axi_awid),
		.s00_axi_awaddr         (s00.s_axi_awaddr),
		.s00_axi_awlen          (s00.s_axi_awlen),
		.s00_axi_awsize         (s00.s_axi_awsize),
		.s00_axi_awburst        (s00.s_axi_awburst),
		.s00_axi_awlock         (s00.s_axi_awlock),
		.s00_axi_awcache        (s00.s_axi_awcache),
		.s00_axi_awprot         (s00.s_axi_awprot),
		.s00_axi_awqos          (s00.s_axi_awqos),
		.s00_axi_awuser         (s00.s_axi_awuser),
		.s00_axi_awvalid        (s00.s_axi_awvalid),
		.s00_axi_awready        (s00.s_axi_awready),
		.s00_axi_wdata          (s00.s_axi_wdata),
		.s00_axi_wstrb          (s00.s_axi_wstrb),
		.s00_axi_wlast          (s00.s_axi_wlast),
		.s00_axi_wuser          (s00.s_axi_wuser),
		.s00_axi_wvalid         (s00.s_axi_wvalid),
		.s00_axi_wready         (s00.s_axi_wready),
		.s00_axi_bid            (s00.s_axi_bid),
		.s00_axi_bresp          (s00.s_axi_bresp),
		.s00_axi_buser          (s00.s_axi_buser),
		.s00_axi_bvalid         (s00.s_axi_bvalid),
		.s00_axi_bready         (s00.s_axi_bready),
		.s00_axi_arid           (s00.s_axi_arid),
		.s00_axi_araddr         (s00.s_axi_araddr),
		.s00_axi_arlen          (s00.s_axi_arlen),
		.s00_axi_arsize         (s00.s_axi_arsize),
		.s00_axi_arburst        (s00.s_axi_arburst),
		.s00_axi_arlock         (s00.s_axi_arlock),
		.s00_axi_arcache        (s00.s_axi_arcache),
		.s00_axi_arprot         (s00.s_axi_arprot),
		.s00_axi_arqos          (s00.s_axi_arqos),
		.s00_axi_aruser         (s00.s_axi_aruser),
		.s00_axi_arvalid        (s00.s_axi_arvalid),
		.s00_axi_arready        (s00.s_axi_arready),
		.s00_axi_rid            (s00.s_axi_rid),
		.s00_axi_rdata          (s00.s_axi_rdata),
		.s00_axi_rresp          (s00.s_axi_rresp),
		.s00_axi_rlast          (s00.s_axi_rlast),
		.s00_axi_ruser          (s00.s_axi_ruser),
		.s00_axi_rvalid         (s00.s_axi_rvalid),
		.s00_axi_rready         (s00.s_axi_rready),

		// s01
		.s01_axi_awid           (s01.s_axi_awid),
		.s01_axi_awaddr         (s01.s_axi_awaddr),
		.s01_axi_awlen          (s01.s_axi_awlen),
		.s01_axi_awsize         (s01.s_axi_awsize),
		.s01_axi_awburst        (s01.s_axi_awburst),
		.s01_axi_awlock         (s01.s_axi_awlock),
		.s01_axi_awcache        (s01.s_axi_awcache),
		.s01_axi_awprot         (s01.s_axi_awprot),
		.s01_axi_awqos          (s01.s_axi_awqos),
		.s01_axi_awuser         (s01.s_axi_awuser),
		.s01_axi_awvalid        (s01.s_axi_awvalid),
		.s01_axi_awready        (s01.s_axi_awready),
		.s01_axi_wdata          (s01.s_axi_wdata),
```

```
		.s01_axi_wstrb		(s01.s_axi_wstrb),
		.s01_axi_wlast		(s01.s_axi_wlast),
		.s01_axi_wuser		(s01.s_axi_wuser),
		.s01_axi_wvalid		(s01.s_axi_wvalid),
		.s01_axi_wready		(s01.s_axi_wready),
		.s01_axi_bid		(s01.s_axi_bid),
		.s01_axi_bresp		(s01.s_axi_bresp),
		.s01_axi_buser		(s01.s_axi_buser),
		.s01_axi_bvalid		(s01.s_axi_bvalid),
		.s01_axi_bready		(s01.s_axi_bready),
		.s01_axi_arid		(s01.s_axi_arid),
		.s01_axi_araddr		(s01.s_axi_araddr),
		.s01_axi_arlen		(s01.s_axi_arlen),
		.s01_axi_arsize		(s01.s_axi_arsize),
		.s01_axi_arburst	(s01.s_axi_arburst),
		.s01_axi_arlock		(s01.s_axi_arlock),
		.s01_axi_arcache	(s01.s_axi_arcache),
		.s01_axi_arprot		(s01.s_axi_arprot),
		.s01_axi_arqos		(s01.s_axi_arqos),
		.s01_axi_aruser		(s01.s_axi_aruser),
		.s01_axi_arvalid	(s01.s_axi_arvalid),
		.s01_axi_arready	(s01.s_axi_arready),
		.s01_axi_rid		(s01.s_axi_rid),
		.s01_axi_rdata		(s01.s_axi_rdata),
		.s01_axi_rresp		(s01.s_axi_rresp),
		.s01_axi_rlast		(s01.s_axi_rlast),
		.s01_axi_ruser		(s01.s_axi_ruser),
		.s01_axi_rvalid		(s01.s_axi_rvalid),
		.s01_axi_rready		(s01.s_axi_rready),

/*
 * AXI master interface
 */
		.m00_axi_awid		(m00.m_axi_awid),
		.m00_axi_awaddr		(m00.m_axi_awaddr),
		.m00_axi_awlen		(m00.m_axi_awlen),
		.m00_axi_awsize		(m00.m_axi_awsize),
		.m00_axi_awburst	(m00.m_axi_awburst),
		.m00_axi_awlock		(m00.m_axi_awlock),
		.m00_axi_awcache	(m00.m_axi_awcache),
		.m00_axi_awprot		(m00.m_axi_awprot),
		.m00_axi_awqos		(m00.m_axi_awqos),
		.m00_axi_awregion	(m00.m_axi_awregion),
		.m00_axi_awuser		(m00.m_axi_awuser),
		.m00_axi_awvalid	(m00.m_axi_awvalid),
		.m00_axi_awready	(m00.m_axi_awready),
		.m00_axi_wdata		(m00.m_axi_wdata),
		.m00_axi_wstrb		(m00.m_axi_wstrb),
		.m00_axi_wlast		(m00.m_axi_wlast),
		.m00_axi_wuser		(m00.m_axi_wuser),
		.m00_axi_wvalid		(m00.m_axi_wvalid),
		.m00_axi_wready		(m00.m_axi_wready),
		.m00_axi_bid		(m00.m_axi_bid),
		.m00_axi_bresp		(m00.m_axi_bresp),
		.m00_axi_buser		(m00.m_axi_buser),
		.m00_axi_bvalid		(m00.m_axi_bvalid),
		.m00_axi_bready		(m00.m_axi_bready),
		.m00_axi_arid		(m00.m_axi_arid),
		.m00_axi_araddr		(m00.m_axi_araddr),
		.m00_axi_arlen		(m00.m_axi_arlen),
		.m00_axi_arsize		(m00.m_axi_arsize),
		.m00_axi_arburst	(m00.m_axi_arburst),
		.m00_axi_arlock		(m00.m_axi_arlock),
		.m00_axi_arcache	(m00.m_axi_arcache),
		.m00_axi_arprot		(m00.m_axi_arprot),
		.m00_axi_arqos		(m00.m_axi_arqos),
		.m00_axi_arregion	(m00.m_axi_arregion),
		.m00_axi_aruser		(m00.m_axi_aruser),
		.m00_axi_arvalid	(m00.m_axi_arvalid),
		.m00_axi_arready	(m00.m_axi_arready),
```

```
                .m00_axi_rid            (m00.m_axi_rid),
                .m00_axi_rdata          (m00.m_axi_rdata),
                .m00_axi_rresp          (m00.m_axi_rresp),
                .m00_axi_rlast          (m00.m_axi_rlast),
                .m00_axi_ruser          (m00.m_axi_ruser),
                .m00_axi_rvalid         (m00.m_axi_rvalid),
                .m00_axi_rready         (m00.m_axi_rready)
        );

endmodule : axi_crossbar_2x1_wrapper
```

```
import defines::*;
import axi_defines::*;
`timescale 1ns / 1ps

module axi_ram_sv_wrapper #
(
   // Width of data bus in bits
   parameter DATA_WIDTH = 32,
   // Width of address bus in bits
   parameter ADDR_WIDTH = 16,
   // Width of wstrb (width of data bus in words)
   parameter STRB_WIDTH = (DATA_WIDTH/8),
   // Extra pipeline register on output
   parameter PIPELINE_OUTPUT = 1
) (
	input logic clk,
	input logic rst,
	axi_interface axi_bus
);

	axi_ram # (
		.DATA_WIDTH			(DATA_WIDTH),
		.ADDR_WIDTH			(ADDR_WIDTH),
		.STRB_WIDTH			(DATA_WIDTH/8),
		.PIPELINE_OUTPUT	(PIPELINE_OUTPUT)
	) axi_ram (
		.clk				(clk),
		.rst				(rst),
		.s_axi_awid			(axi_bus.s_axi_awid),
		.s_axi_awaddr		(axi_bus.s_axi_awaddr),
		.s_axi_awlen		(axi_bus.s_axi_awlen),
		.s_axi_awsize		(axi_bus.s_axi_awsize),
		.s_axi_awburst		(axi_bus.s_axi_awburst),
		.s_axi_awlock		(axi_bus.s_axi_awlock),
		.s_axi_awcache		(axi_bus.s_axi_awcache),
		.s_axi_awprot		(axi_bus.s_axi_awprot),
		.s_axi_awvalid		(axi_bus.s_axi_awvalid),
		.s_axi_awready		(axi_bus.s_axi_awready),
		.s_axi_wdata		(axi_bus.s_axi_wdata),
		.s_axi_wstrb		(axi_bus.s_axi_wstrb),
		.s_axi_wlast		(axi_bus.s_axi_wlast),
		.s_axi_wvalid		(axi_bus.s_axi_wvalid),
		.s_axi_wready		(axi_bus.s_axi_wready),
		.s_axi_bid			(axi_bus.s_axi_bid),
		.s_axi_bresp		(axi_bus.s_axi_bresp),
		.s_axi_bvalid		(axi_bus.s_axi_bvalid),
		.s_axi_bready		(axi_bus.s_axi_bready),
		.s_axi_arid			(axi_bus.s_axi_arid),
		.s_axi_araddr		(axi_bus.s_axi_araddr),
		.s_axi_arlen		(axi_bus.s_axi_arlen),
		.s_axi_arsize		(axi_bus.s_axi_arsize),
		.s_axi_arburst		(axi_bus.s_axi_arburst),
		.s_axi_arlock		(axi_bus.s_axi_arlock),
		.s_axi_arcache		(axi_bus.s_axi_arcache),
		.s_axi_arprot		(axi_bus.s_axi_arprot),
		.s_axi_arvalid		(axi_bus.s_axi_arvalid),
		.s_axi_arready		(axi_bus.s_axi_arready),
		.s_axi_rid			(axi_bus.s_axi_rid),
		.s_axi_rdata		(axi_bus.s_axi_rdata),
		.s_axi_rresp		(axi_bus.s_axi_rresp),
		.s_axi_rlast		(axi_bus.s_axi_rlast),
		.s_axi_rvalid		(axi_bus.s_axi_rvalid),
		.s_axi_rready		(axi_bus.s_axi_rready)
	);

endmodule : axi_ram_sv_wrapper
```

```
// WARNING! for single R/W only

module axil_axi_adapter # (
	parameter 			USER_ID 		= 0
)
(
	axil_interface.axil_slave	axil_bus,	// slave axil bus
	axi_interface 					axi_bus 		// master axi bus
);

	// wire linking
	assign axi_bus.m_axi_awid 		= USER_ID;		// don't care
	assign axi_bus.m_axi_awaddr 		= axil_bus.s_axil_awaddr;
	assign axi_bus.m_axi_awlen 		= 0;		// single burst
	assign axi_bus.m_axi_awsize 		= 3'b010;	// 4 byte per single burst (1 word)
	assign axi_bus.m_axi_awburst	= 2'b00;	// fixed burst len ? the spec say should be 2'b01, INCR
	assign axi_bus.m_axi_awlock 		= 0;		// normal
	assign axi_bus.m_axi_awcache	= 4'b0011;		// Normal Non-cacheable Bufferable
	assign axi_bus.m_axi_awprot 		= axil_bus.s_axil_awprot;
	assign axi_bus.m_axi_awqos 		= 0;
	assign axi_bus.m_axi_awuser 		= 0;
	assign axi_bus.m_axi_awvalid	= axil_bus.s_axil_awvalid;
	assign axi_bus.m_axi_wdata 		= axil_bus.s_axil_wdata;
	assign axi_bus.m_axi_wstrb 		= axil_bus.s_axil_wstrb;
	//assign axi_bus.m_axi_wlast 		= (axil_bus.s_axil_wvalid || axi_bus.m_axi_wready);	// TODO: BUG?
	assign axi_bus.m_axi_wlast 		= 1'b1;
	assign axi_bus.m_axi_wuser 		= 0;
	assign axi_bus.m_axi_wvalid 		= axil_bus.s_axil_wvalid;
	// assign axi_bus.m_axi_buser 		= 0; // don't care
	assign axi_bus.m_axi_bready 		= axil_bus.s_axil_bready;
	assign axi_bus.m_axi_arid 		= USER_ID;		// don't care
	assign axi_bus.m_axi_araddr 		= axil_bus.s_axil_araddr;
	assign axi_bus.m_axi_arlen 		= 0;		// single
	assign axi_bus.m_axi_arsize 		= 3'b010;	// 4 byte burst (1 word)
	assign axi_bus.m_axi_arburst	= 2'b00;	// fixed burst len
	assign axi_bus.m_axi_arlock 		= 0;		// normal
	assign axi_bus.m_axi_arcache	= 4'b0011; // Normal Non-cacheable Bufferable
	assign axi_bus.m_axi_arqos 		= 0;
	assign axi_bus.m_axi_aruser 		= 0;
	assign axi_bus.m_axi_arprot 		= axil_bus.s_axil_arprot;
	assign axi_bus.m_axi_arvalid	= axil_bus.s_axil_arvalid;
	//assign axi_bus.m_axi_ruser 		= 0; // don't care
	assign axi_bus.m_axi_rready 		= axil_bus.s_axil_rready;

	assign axil_bus.s_axil_awready	= axi_bus.m_axi_awready;
	assign axil_bus.s_axil_wready	= axi_bus.m_axi_wready;
	// assign axi_bus.m_axi_bid 		= 0;		// don't care
	assign axil_bus.s_axil_bresp	= axi_bus.m_axi_bresp;
	assign axil_bus.s_axil_bvalid	= axi_bus.m_axi_bvalid;
	assign axil_bus.s_axil_arready	= axi_bus.m_axi_arready;
	// assign axi_bus.m_axi_rid 		= 0;		// don't care
	assign axil_bus.s_axil_rdata	= axi_bus.m_axi_rdata;
	assign axil_bus.s_axil_rresp	= axi_bus.m_axi_rresp;
	//assign axi_bus.s_axi_rlast 		= ;		// don't care
	assign axil_bus.s_axil_rvalid	= axi_bus.m_axi_rvalid;

endmodule : axil_axi_adapter
```

```
import defines::*;
import axi_defines::*;

module axil_dummy_master (
	axil_interface.axil_master		m00
);

	always_comb begin : dummy_master_signal
		m00.axil_awaddr			= NULL;
		m00.axil_awprot			= 3'b0;
		m00.axil_awvalid	= INVALID;
		m00.axil_wdata			= NULL;
		m00.axil_wstrb			= 4'b0;
		m00.axil_wvalid			= INVALID;
		m00.axil_bready			= INVALID;
		m00.axil_araddr			= NULL;
		m00.axil_arprot			= 3'b0;
		m00.axil_arvalid	= INVALID;
		m00.axil_rready			= INVALID;
	end

endmodule : axil_dummy_master


module axil_dummy_slave (
	axil_interface.axil_slave		s00
);

	always_comb begin : dummy_slave_signal
		s00.axil_awready	= INVALID;
		s00.axil_wready			= INVALID;
		s00.axil_bresp			= RESP_OKAY;
		s00.axil_bvalid			= INVALID;
		s00.axil_arready	= INVALID;
		s00.axil_rdata			= NULL;
		s00.axil_rresp			= RESP_OKAY;
		s00.axil_rvalid			= INVALID;
	end

endmodule : axil_dummy_slave
```

```
interface axil_interface # (
	// Width of address bus in bits
	parameter ADDR_WIDTH		= 32
) ();

	// Width of AXI interface data bus in bits
	localparam DATA_WIDTH		= 32;
	// Width of data bus in words
	localparam STRB_WIDTH		= (DATA_WIDTH/8);

	// bus signals
	logic	[ADDR_WIDTH-1:0]axil_awaddr;
	logic	[2:0]			axil_awprot;
	logic					axil_awvalid;
	logic					axil_awready;
	logic	[DATA_WIDTH-1:0]axil_wdata;
	logic	[STRB_WIDTH-1:0] axil_wstrb;
	logic					axil_wvalid;
	logic					axil_wready;
	logic	[1:0]			axil_bresp;
	logic					axil_bvalid;
	logic					axil_bready;
	logic	[ADDR_WIDTH-1:0]axil_araddr;
	logic	[2:0]			axil_arprot;
	logic					axil_arvalid;
	logic					axil_arready;
	logic	[DATA_WIDTH-1:0]axil_rdata;
	logic	[1:0]			axil_rresp;
	logic					axil_rvalid;
	logic					axil_rready;

	modport axil_master (
		output		axil_awaddr,
		output		axil_awprot,
		output		axil_awvalid,
		input		axil_awready,
		output		axil_wdata,
		output		axil_wstrb,
		output		axil_wvalid,
		input		axil_wready,
		input		axil_bresp,
		input		axil_bvalid,
		output		axil_bready,
		output		axil_araddr,
		output		axil_arprot,
		output		axil_arvalid,
		input		axil_arready,
		input		axil_rdata,
		input		axil_rresp,
		input		axil_rvalid,
		output		axil_rready
	);

	modport axil_slave (
		input		axil_awaddr,
		input		axil_awprot,
		input		axil_awvalid,
		output		axil_awready,
		input		axil_wdata,
		input		axil_wstrb,
		input		axil_wvalid,
		output		axil_wready,
		output		axil_bresp,
		output		axil_bvalid,
		input		axil_bready,
		input		axil_araddr,
		input		axil_arprot,
		input		axil_arvalid,
		output		axil_arready,
		output		axil_rdata,
```

```
                output                                  axil_rresp,
                output                                  axil_rvalid,
                input                                   axil_rready
        );

endinterface : axil_interface
```

```
import defines::*;
import axi_defines::*;
`timescale 1ns / 1ps

module axil_ram_sv_wrapper #
(
   // Width of data bus in bits
   parameter DATA_WIDTH = 32,
   // Width of address bus in bits
   parameter ADDR_WIDTH = 16,
   // Width of wstrb (width of data bus in words)
   parameter STRB_WIDTH = (DATA_WIDTH/8),
   // Extra pipeline register on output
   parameter PIPELINE_OUTPUT = 1,
            // load the elf file into the mem on simulation
            parameter bootload = 0
) (
            input logic clk,
            input logic rst,
            axil_interface.axil_slave axil_bus
);

            axil_ram # (
                        .DATA_WIDTH                 (DATA_WIDTH),
                        .ADDR_WIDTH                 (ADDR_WIDTH),
                        .STRB_WIDTH                 (DATA_WIDTH/8),
                        .PIPELINE_OUTPUT    (PIPELINE_OUTPUT),
                        .bootload                   (bootload)
            ) axil_ram (
                        .clk                                (clk),
                        .rst                                (rst),
                        .s_axil_awaddr              (axil_bus.axil_awaddr),
                        .s_axil_awprot              (axil_bus.axil_awprot),
                        .s_axil_awvalid             (axil_bus.axil_awvalid),
                        .s_axil_awready             (axil_bus.axil_awready),
                        .s_axil_wdata               (axil_bus.axil_wdata),
                        .s_axil_wstrb               (axil_bus.axil_wstrb),
                        .s_axil_wvalid              (axil_bus.axil_wvalid),
                        .s_axil_wready              (axil_bus.axil_wready),
                        .s_axil_bresp               (axil_bus.axil_bresp),
                        .s_axil_bvalid              (axil_bus.axil_bvalid),
                        .s_axil_bready              (axil_bus.axil_bready),
                        .s_axil_araddr              (axil_bus.axil_araddr),
                        .s_axil_arprot              (axil_bus.axil_arprot),
                        .s_axil_arvalid             (axil_bus.axil_arvalid),
                        .s_axil_arready             (axil_bus.axil_arready),
                        .s_axil_rdata               (axil_bus.axil_rdata),
                        .s_axil_rresp               (axil_bus.axil_rresp),
                        .s_axil_rvalid              (axil_bus.axil_rvalid),
                        .s_axil_rready              (axil_bus.axil_rready)
            );

endmodule : axil_ram_sv_wrapper
```

```
interface sdram_interface #(
        // by bits it means "pin couont"
        // sdram's addr is input as row and col
        parameter ADDR_BITS     = 13,
        parameter BA_BITS   = 2,
        parameter DQ_BITS   = 16,
        parameter DQM_BITS      = 2
) ();

        // all outputs except sdram_data_io is output
        wire                                    sdram_clk;
        wire                                    sdram_cke;
        wire [DQM_BITS - 1 : 0]         sdram_dqm;
        wire                                    sdram_cas_n;
        wire                                    sdram_ras_n;
        wire                                    sdram_we_n;
        wire                                    sdram_cs_n;
        wire [BA_BITS - 1 : 0]          sdram_ba;
        wire [ADDR_BITS - 1 : 0]    sdram_addr;
        wire [DQ_BITS - 1 : 0]          sdram_dq; // inout

endinterface : sdram_interface
```

```
// axil slave to simp master

import defines::*;
import pref_defines::*;
import axi_defines::*;

/*
here I define the simp interface:
input	logics:
	addr
	data_in
	wr
	rd
	valid
	be

output	logics
	data_out
	done

valid must stay high until done is asserted
handshake is valid && done
done only assert for one cycle
on a read; addr, rd, valid, be must hold until done
on a write; addr, wr, valid, be must hold until done
data_out must valid when handshake
must not read-when-write
slave must finally assert done regardless of r/w success
*/

module axil2simp # (
	parameter  ADDR_WIDTH = 16
) (
	// Inputs
	input	logic                                   clk,
	input	logic                                   rst,

	// axi
	input	logic                                   awvalid_i,
	input	logic	[ADDR_WIDTH - 1:0]      awaddr_i,
	input	logic                                   wvalid_i,
	input	logic	[31:0]                  wdata_i,
	input	logic	[ 3:0]                  wstrb_i,
	input	logic                                   bready_i,
	input	logic                                   arvalid_i,
	input	logic	[ADDR_WIDTH - 1:0]      araddr_i,
	input	logic                                   rready_i,

	output	logic                                   awready_o,
	output	logic                                   wready_o,
	output	logic                                   bvalid_o,
	output	logic	[1:0]                   bresp_o,
	output	logic                                   arready_o,
	output	logic                                   rvalid_o,
	output	logic	[31:0]                  rdata_o,
	output	logic	[ 1:0]                  rresp_o,

	// unused AXI signal
	input	logic	[ 2:0]                  awprot_i,
	input	logic	[ 2:0]                  arprot_i,

	// simp master interface
	output	logic	[31:0]                  simp_addr,
	output	logic	[31:0]                  simp_data_in,
	output	logic                                   simp_wr,
	output	logic                                   simp_rd,
	output	logic                                   simp_valid,
	output	logic	[ 3:0]                  simp_be,

	input	logic	[31:0]                  simp_data_out,
```

```
        input   logic                                                   simp_done
);

        typedef enum logic[2:0] {
                IDLE,
                W_DATA,
                W_RESP,
                W_SIMP,
                R_RESP,
                R_DONE
        } state_t;

        state_t state, nxt_state;

        always_ff @( posedge clk, posedge rst ) begin
                if (rst)
                        state <= IDLE;
                else
                        state <= nxt_state;
        end

        // five channels' handshake
        logic read_addr_handshake, read_data_handshake;
        logic write_addr_handshake, write_data_handshake, write_resp_handshake;

        always_comb begin
                read_addr_handshake             = arready_o && arvalid_i;
                read_data_handshake             = rready_i && rvalid_o;
                write_addr_handshake            = awready_o && awvalid_i;
                write_data_handshake= wready_o && wvalid_i;
                write_resp_handshake= bready_i && bvalid_o;
        end

        logic simp_handshake;
        assign simp_handshake = simp_valid && simp_done;

        data_t simp_addr_reg;
        data_t simp_data_out_reg;

        always_ff @( posedge clk ) begin
                if (read_addr_handshake)
                        simp_addr_reg <= araddr_i;
                else if (write_addr_handshake)
                        simp_addr_reg <= awaddr_i;
                else
                        simp_addr_reg <= simp_addr_reg;
        end

        always_ff @( posedge clk ) begin
                if (write_data_handshake)
                        simp_data_in <= wdata_i;
                else
                        simp_data_in <= simp_data_in;
        end

        always_ff @( posedge clk ) begin
                if (simp_valid && simp_done && simp_rd)
                        simp_data_out_reg <= simp_data_out;
                else
                        simp_data_out_reg <= simp_data_out_reg;
        end

        always_comb begin : fsm
                nxt_state               = IDLE;
                awready_o               = 1'b0;
                wready_o                = 1'b0;
                bvalid_o                = INVALID;
                bresp_o                         = RESP_OKAY;
                arready_o               = 1'b0;
                rvalid_o                = INVALID;
```

```
rdata_o                 = NULL;
rresp_o                 = RESP_OKAY;

simp_addr       = NULL;
simp_wr                 = DISABLE;
simp_rd                 = DISABLE;
simp_valid      = VALID;
simp_be                 = 4'b0;

unique case (state)

        IDLE:   begin
                aready_o                = 1'b1;
                if (arvalid_i) begin
                        nxt_state       = R_RESP;
                end else if (awvalid_i) begin
                        awready_o       = 1'b1;
                        if (wvalid_i) begin
                                wready_o  = 1'b1;
                                nxt_state   = W_SIMP;
                        end else begin
                                nxt_state   = W_DATA;
                        end
                end else begin
                        nxt_state       = IDLE;
                end
        end

        W_DATA:begin
                if (wvalid_i) begin
                        wready_o = 1'b1;
                        nxt_state = W_SIMP;
                end begin
                        nxt_state = W_DATA;
                end
        end

        W_SIMP:  begin
                simp_addr       = simp_addr_reg;
                simp_wr                 = ENABLE;
                simp_valid      = VALID;
                simp_be                 = wstrb_i;

                if (simp_handshake) begin
                        nxt_state   = W_RESP;
                end else begin
                        nxt_state   = W_SIMP;
                end
        end

        W_RESP: begin
                bvalid_o = VALID;
                if (write_resp_handshake) begin
                        nxt_state = IDLE;
                end else begin
                        nxt_state = W_RESP;
                end
        end

        R_RESP:  begin
                simp_addr       = simp_addr_reg;
                simp_rd                 = ENABLE;
                simp_valid      = VALID;
                simp_be                 = 4'b1111;
                if (simp_handshake) begin
                        nxt_state   = R_DONE;
                end else begin
                        nxt_state   = R_RESP;
                end
        end
```

```
                        R_DONE: begin
                                rvalid_o                = VALID;
                                rdata_o                         = simp_data_out_reg;
                                if (rready_i)
                                        nxt_state   = IDLE;
                                else
                                        nxt_state   = R_DONE;
                        end

                        default:begin
                                nxt_state   = IDLE;
                                awready_o= 1'b0;
                                wready_o  = 1'b0;
                                bvalid_o    = INVALID;
                                bresp_o                 = RESP_OKAY;
                                arready_o  = 1'b0;
                                rvalid_o    = INVALID;
                                rdata_o                 = NULL;
                                rresp_o                 = RESP_OKAY;
                        end

                endcase
        end

endmodule : axil2simp
```

```
import defines::*;
import mem_defines::*;

module cache (
        // input nets
        input logic                             clk,
        input logic                             en,
        input index_t                           index,
        input logic                             rd,
        input logic                             wr_data,
        input logic                             wr_flag,
        input flag_line_t               flag_line_in,
        input data_line_t               data_line_in,
        input data_line_en_t    data_line_en,

        // output nets
        output flag_line_t              flag_line_out,
        output data_line_t              data_line_out
);


ram_48b_512wd cache_flag_block (
        .address        (index),
        .clock                  (clk),
        .data                   (flag_line_in),
        .rden                   (rd && en),
        .wren                   (wr_flag && en),
        .q                              (flag_line_out)
);


ram_256b_512wd cache_data_block (
        .address        (index),
        .byteena        (data_line_en),
        .clock                  (clk),
        .data                   (data_line_in),
        .rden                   (rd && en),
        .wren                   (wr_data && en),
        .q                              (data_line_out)
);

endmodule : cache
```

```
/*
This is the module that monitors if any other hart have changed
the value in some memory location that holds a lock
*/

import defines::*;
import mem_defines::*;

module exclusive_monitor #(
        RES_ENTRY_CNT = MAX_NEST_LOCK
) (
        input   logic           clk,
        input   logic           rst_n,
   input instr_a_t      instr,
        input   data_t          addr,
        input   logic           mem_wr,
        input   logic           update,     // up only one cycle
        output  logic           success
);

        reservation_set_t       reservation_set                 [0 : MAX_NEST_LOCK - 1];    // a stack of reservation stack
        logic [$clog2(MAX_NEST_LOCK) - 1 : 0]  pointer;                                     // pointer to the newest
reservation stack
        logic set_valid, clear_valid;
        logic push, pop;
        opcode_t  opcode;
        funct5_t  funct5;
        logic                   is_atomic, is_lr, is_sc;
        logic                   addr_hit;

        always_comb begin : ctrl_signal_assign
                opcode          =       instr.opcode;
                funct5          =       instr.funct5;
                is_atomic  =    opcode == ATOMIC;
                is_lr           =       is_atomic && funct5 == LR;
                is_sc           =       is_atomic && funct5 == SC;
                addr_hit   =    addr[XLEN-1:2] == reservation_set[pointer].addr[XLEN-1:2];
                set_valid  =    is_lr;
                clear_valid=    mem_wr && addr_hit && update;
                push            =       is_lr;
                pop                     =       is_sc;
                success         =       is_sc &&
                                        addr_hit &&
                                        reservation_set[pointer].valid;
        end


        always_ff @(posedge clk, negedge rst_n) begin
                if (~rst_n) begin
                        pointer <= 3'b111;
                end else if (push && update) begin
                        pointer <= pointer + 3'b1;
                end else if (pop && update) begin
                        pointer <= pointer - 3'b1;
                end else begin
                        pointer <= pointer;
                end
        end


        integer i;
        always_ff @(posedge clk, negedge rst_n) begin
                for (i = 0; i < MAX_NEST_LOCK; i++) begin
                        if (~rst_n) begin
                                reservation_set[i].addr                         <= NULL;
                                reservation_set[i].valid                <= INVALID;
                        end else if (set_valid && update) begin
                                if (i[$clog2(MAX_NEST_LOCK) - 1:0] == pointer + 3'b1) begin
                                        reservation_set[i].addr                 <= addr;
                                        reservation_set[i].valid        <= VALID;
```

```
                        end else begin
                                reservation_set[i].addr                 <= reservation_set[i].addr;
                                reservation_set[i].valid        <= reservation_set[i].valid;
                        end
                // an regular st/ld without 'update' bit set can also trigger clear valid
                end else if (clear_valid) begin
                        if (i[$clog2(MAX_NEST_LOCK) - 1:0] == pointer) begin
                                reservation_set[i].addr                 <= reservation_set[i].addr;
                                reservation_set[i].valid        <= INVALID;
                        end else begin
                                reservation_set[i].addr                 <= reservation_set[i].addr;
                                reservation_set[i].valid        <= reservation_set[i].valid;
                        end
                end else begin
                        reservation_set[i].addr                 <= reservation_set[i].addr;
                        reservation_set[i].valid                <= reservation_set[i].valid;
                end
        end
    end

// TODO: constraint: ld must followed by a st at same addr, ld must not followed by another ld

endmodule : exclusive_monitor
```

```
package mem_defines;
import defines::*;

`ifndef _mem_defines_
`define _mem_defines_

// note: EP4cE10 FPGA have 46 M9K blocks

/*
        Cache model:

        cache flag line:
        valid0 - dirty0 - tag0 - valid1 - dirty1 - tag1 - LRU - X
        |----------------------flag----48b---------------------|
          1      1     19     1      1     19    1   5

        cache data line:
        data0w0 - data0w1 - data0w2 - data0w3 - data1w0 - data1w1 - data1w2 - data1w3
        |----------------------------data----256b-----------------------------------|
          32      32     32      32      32      32      32      32

        writing policy:         write back - write back to memory when evict
                                             write allocate - miss-write are being written into cache
        replacement policy:     LRU: evict the least recent used way

        Address representation:
        31-------------------13 12----------4    3-2         1-0
                                tag(19)         index(9)   word_off(2)  byte_off(2)
*/

localparamMEM_ACCESS_TIMEOUT    = 128;                          // 128 cycles

localparamDEBUG_MEM_SYS         = DISABLE;
localparamDEBUG_SDRAM                   = DISABLE;

localparamtag_len                       = 19;
localparamindex_len             = 9;
localparamword_off              = 2;
localparambyte_off              = 2;
localparamdata_line_len         = 256;
localparamflag_line_len         = 48;
localparamempty_data_line       = 256'b0;
localparamempty_flag_line       = 48'b0;
localparamsdram_addr_len        = 24;           // 2^24 words
localparamsdram_word                    = 16;           // 16 bit word
localparam  sdram_access_len    = 10'd8;    // 8 16-bit word each access
localparamDIRTY                         = 1'b1;
localparamCLEAN                         = 1'b0;

localparamword_align_mask       = 32'hffff_fffc;

// atomic operation defines
localparamSC_FAIL_ECODE         = 32'b1;    // if sc fail, write this value to rd
localparamSC_SUCCESS_CODE               = NULL;

localparamMAX_NEST_LOCK = 8;                                    // max nested lock aquire length,
                                                                                                        //      cases
that over 2 is very rare

typedef logic [tag_len - 1 : 0]         tag_t;
typedef logic [index_len - 1 : 0]   index_t;
typedef logic [4 : 0]                   x5_t;

typedef enum logic[2:0] {
        REGULAR_LD_OUT,
        LOAD_CONDITIONAL_OUT,
        STORE_CONDITIONAL_SUC_OUT,
        STORE_CONDITIONAL_FAIL_OUT,
        AMO_INSTR_OUT,
        NULL_OUT
```

```
} mem_out_sel_t;

typedef struct packed {
        tag_t           tag;
        index_t         index;
        logic[1:0]  word_off;
        logic[1:0]  byte_off;
} cache_addr_t;

typedef struct packed{
        logic           valid0;
        logic           dirty0;
        tag_t           tag0;
        logic           valid1;
        logic           dirty1;
        tag_t           tag1;
        logic           lru;
        x5_t            x5;
} flag_line_t;

typedef struct packed {
        data_t          data0w0;
        data_t          data0w1;
        data_t          data0w2;
        data_t          data0w3;
        data_t          data1w0;
        data_t          data1w1;
        data_t          data1w2;
        data_t          data1w3;
} data_line_t;

typedef struct {
        flag_line_t flag;
        data_line_tdata;
} cache_line_t;

typedef enum logic [2:0] {
        CACHE_IDLE              = 3'b000,   // does nothing
        COMP_READ               = 3'b110,   // load instr
        COMP_WRITE              = 3'b111,   // store instr
        ACCESS_READ             = 3'b100,   // cache to ram
        ACCESS_WRITE    = 3'b101,   // ram to cache
        CACHE_ERR_1             = 3'b001,   // should not occur
        CACHE_ERR_2             = 3'b010,   // should not occur
        CACHE_ERR_3             = 3'b011    // should not occur
} cache_access_mode_t;

typedef logic[sdram_addr_len - 1 : 0]       sdram_addr_t;
typedef logic[sdram_word - 1:0]                     sdram_wd_t;
typedef struct packed {
        sdram_wd_t      w0;
        sdram_wd_t      w1;
        sdram_wd_t      w2;
        sdram_wd_t      w3;
        sdram_wd_t      w4;
        sdram_wd_t      w5;
        sdram_wd_t      w6;
        sdram_wd_t      w7;
} sdram_8_wd_t;

typedef enum logic[3:0] {
        RD_DISABLE = 4'b0000,
        RDW0 = 4'b1000,
        RDW1 = 4'b1001,
        RDW2 = 4'b1010,
        RDW3 = 4'b1011,
        RDW4 = 4'b1100,
        RDW5 = 4'b1101,
        RDW6 = 4'b1110,
        RDW7 = 4'b1111
```

```
} rd_index_t;


localparam be_none   = 32'h0000_0000;
localparam be_w0     = 32'hffff_0000;
localparam be_w1     = 32'h0000_ffff;
localparam be_all    = 32'hffff_ffff;

typedef enum logic[1:0] {
        WAY_SEL_NONE    = 2'b00,
        WAY_SEL_W0              = 2'b01,
        WAY_SEL_W1              = 2'b10,
        WAY_SEL_ALL             = 2'b11
} way_sel_t;

typedef struct packed {
        logic en0;
        logic en1;
        logic en2;
        logic en3;
} word_en_t;

typedef struct packed {
        word_en_t w0en;
        word_en_t w1en;
        word_en_t w2en;
        word_en_t w3en;
} word_4_en_t;

typedef struct packed {
        word_4_en_t l0en;
        word_4_en_t l1en;
} data_line_en_t;

// for monitoring conditional load/store
typedef struct packed{
                data_t      addr;
                logic       valid;
} reservation_set_t;

`endif

endpackage : mem_defines
```

```
import defines::*;
import mem_defines::*;
import axi_defines::*;

module mem_sys_axil_wrapper (
        input   logic                           clk,
        input   logic                           rst_n,

        input   cache_addr_t            addr,                   // still 32 bits
        input   data_t                          data_in,
        input   logic                           wr,
        input   logic                           rd,
        input   logic                           valid,
        input   logic                           [BYTES-1:0] be,     // for write only

        output  data_t                          data_out,
        output  logic                           done,

        axil_interface.axil_master      axil_bus
);

        mem_sys_axil mem_sys (
                .clk                            (clk),
                .rst_n                          (rst_n),
                .addr                           (addr),
                .data_in                (data_in),
                .wr                                     (wr),
                .rd                                     (rd),
                .valid                          (valid),
                .be                                     (be),
                .data_out               (data_out),
                .done                           (done),
                .m_axil_awaddr          (axil_bus.axil_awaddr),
                .m_axil_awprot          (axil_bus.axil_awprot),
                .m_axil_awvalid         (axil_bus.axil_awvalid),
                .m_axil_awready         (axil_bus.axil_awready),
                .m_axil_wdata           (axil_bus.axil_wdata),
                .m_axil_wstrb           (axil_bus.axil_wstrb),
                .m_axil_wvalid          (axil_bus.axil_wvalid),
                .m_axil_wready          (axil_bus.axil_wready),
                .m_axil_bresp           (axil_bus.axil_bresp),
                .m_axil_bvalid          (axil_bus.axil_bvalid),
                .m_axil_bready          (axil_bus.axil_bready),
                .m_axil_araddr          (axil_bus.axil_araddr),
                .m_axil_arprot          (axil_bus.axil_arprot),
                .m_axil_arvalid         (axil_bus.axil_arvalid),
                .m_axil_arready         (axil_bus.axil_arready),
                .m_axil_rdata           (axil_bus.axil_rdata),
                .m_axil_rresp           (axil_bus.axil_rresp),
                .m_axil_rvalid          (axil_bus.axil_rvalid),
                .m_axil_rready          (axil_bus.axil_rready)
        );

endmodule : mem_sys_axil_wrapper
```

```
// state machine for handling R/W via axi-lite bus

import defines::*;
import mem_defines::*;
import axi_defines::*;

module mem_sys_axil #(
	parameter ADDR_WIDTH = XLEN,
	parameter DATA_WIDTH = XLEN
) (
	input	logic 					clk,
	input	logic 					rst_n,

	input	cache_addr_t 				addr,		// still 32 bits
	input	data_t 					data_in,
	input	logic 					wr,
	input	logic 					rd,
	input	logic 					valid,
	input	logic 					[BYTES-1:0] be,	// for write only

	output	data_t 					data_out,
	output	logic 					done,

	// AXI-Lite master interface
	output	logic [ADDR_WIDTH - 1:0]	m_axil_awaddr,	// Write address
	output	logic [2:0]				m_axil_awprot,	// Write protection level, see axi_defines.sv
	output	logic 					m_axil_awvalid,	// Write address valid, signaling
valid write address and control information.
	input	logic 					m_axil_awready,	// Write address ready (from
slave), ready to accept an address and associated control signals
	output	logic [DATA_WIDTH-1:0]		m_axil_wdata,	// Write data
	output	logic [(DATA_WIDTH/8)-1:0]	m_axil_wstrb,	// Write data strobe (byte select)
	output	logic 					m_axil_wvalid,	// Write data valid, write data and
strobes are available
	input	logic 					m_axil_wready,	// Write data ready, slave can
accept the write data
	input	logic [1:0]				m_axil_bresp,	// Write response (from slave)
	input	logic 					m_axil_bvalid,	// Write response valid, signaling
a valid write response
	output	logic 					m_axil_bready,	// Write response ready (from
master) can accept a write response
	output	logic [ADDR_WIDTH - 1:0]	m_axil_araddr,	// Read address
	output	logic [2:0]				m_axil_arprot,	// Read protection level, see axi_defines.sv
	output	logic 					m_axil_arvalid,	// Read address valid, signaling
valid read address and control information
	input	logic 					m_axil_arready,	// Read address ready (from slave),
ready to accept an address and associated control signals
	input	logic [DATA_WIDTH-1:0]		m_axil_rdata,	// Read data
	input	logic [1:0]				m_axil_rresp,	// Read response (from slave)
	input	logic 					m_axil_rvalid,	// Read response valid, the
channel is signaling the required read data
	output	logic 					m_axil_rready	// Read response ready (from
master), can accept the read data and response information
	// end AXI-Lite master interface
);

	logic	rst;
	assign rst = ~rst_n;

	logic rden, wren;
	assign rden = rd && valid;
	assign wren = wr && valid;

	typedef enum logic[2:0] {
		IDLE,
		R_ADDR,
		R_DATA,
		W_ADDR,
		W_DATA,
		W_RESP,
```

```
        BAD
} state_t;

state_t state, nxt_state;

always_ff @(posedge clk or negedge rst_n) begin
        if (~rst_n)
                state <= IDLE;
        else
                state <= nxt_state;
end

// five channels' handshake
logic read_addr_handshake, read_data_handshake;
logic write_addr_handshake, write_data_handshake, write_resp_handshake;
logic [1:0] read_handshake;
logic [2:0] write_handshake;

always_comb begin
        read_addr_handshake = m_axil_arready && m_axil_arvalid && rden;
        read_data_handshake = m_axil_rready && m_axil_rvalid && rden;
        write_addr_handshake = m_axil_awready && m_axil_awvalid && wren;
        write_data_handshake = m_axil_wready && m_axil_wvalid && wren;
        write_resp_handshake = m_axil_bready && m_axil_bvalid && wren;
        read_handshake = {read_addr_handshake, read_data_handshake};
        write_handshake = {write_addr_handshake, write_data_handshake, write_resp_handshake};
end

always_comb begin
        data_out        = NULL;
        done                    = 1'b0;
        nxt_state       = IDLE;

        // write
        m_axil_awaddr   = NULL;
        m_axil_awprot   = basic_awport;
        m_axil_awvalid  = INVALID;
        m_axil_wdata    = NULL;
        m_axil_wstrb    = be;
        m_axil_wvalid   = INVALID;
        m_axil_bready   = READY;

        // read
        m_axil_araddr   = NULL;
        m_axil_arprot   = basic_awport;
        m_axil_arvalid  = INVALID;
        m_axil_rready   = READY;

        /*
        input signals:
        m_axil_awready,     // Write address ready (from slave), ready to accept address / ctrl
        m_axil_wready,      // Write data ready, slave can accept the write data
        m_axil_bresp,       // Write response (from slave)
        m_axil_bvalid,      // Write response valid

        m_axil_arready,     // Read address ready (from slave), ready to accept an address and associated control signals
        m_axil_rdata,       // Read data
        m_axil_rresp,       // Read response (from slave)
        m_axil_rvalid,      // Read response valid, the channel is signaling the required read data
        */

        unique case (state)
                IDLE: begin
                        if (rden) begin
                                m_axil_rready   = READY;
                                m_axil_araddr   = addr & word_align_mask;
                                m_axil_arvalid  = VALID;
                                if (read_handshake == 2'b00) begin
                                        nxt_state       = R_ADDR;
                                        done                    = 1'b0;
```

```
                        data_out        = NULL;
                end else if (read_handshake == 2'b10) begin
                        nxt_state       = R_DATA;
                        done                    = 1'b0;
                        data_out        = NULL;
                end else if (read_handshake == 2'b11) begin
                        nxt_state       = IDLE;
                        done                    = DONE;
                        data_out        = m_axil_rdata;
                end else begin
                        nxt_state       = BAD;
                        done                    = 1'b0;
                        data_out        = NULL;
                end
        end else if (wren) begin
                m_axil_awaddr   = addr & word_align_mask;
                m_axil_awvalid  = VALID;
                m_axil_wdata    = data_in;
                m_axil_wvalid   = VALID;
                if (write_handshake == 3'b000) begin
                        nxt_state   = W_ADDR;
                        done            = 1'b0;
                end else if (write_handshake == 3'b100) begin
                        nxt_state   = W_DATA;
                        done            = 1'b0;
                end else if (write_handshake == 3'b110) begin
                        nxt_state   = W_RESP;
                        done            = 1'b0;
                end else if (write_handshake == 3'b111) begin
                        nxt_state   = IDLE;
                        done            = DONE;
                end else begin
                        nxt_state   = BAD;
                        done            = 1'b0;
                end
        end else begin
                nxt_state = IDLE;
        end
end

R_ADDR: begin
        m_axil_rready   = READY;
        m_axil_araddr   = addr & word_align_mask;
        m_axil_arvalid  = VALID;
        if (read_handshake == 2'b00) begin
                nxt_state   = R_ADDR;
                done            = 1'b0;
                data_out    = NULL;
        end else if (read_handshake == 2'b10) begin
                nxt_state   = R_DATA;
                done            = 1'b0;
                data_out    = NULL;
        end else if (read_handshake == 2'b11) begin
                nxt_state   = IDLE;
                done            = DONE;
                data_out    = m_axil_rdata;
        end else begin
                nxt_state   = BAD;
                done            = 1'b0;
                data_out    = NULL;
        end
end

R_DATA: begin
        m_axil_rready = READY;
        if (read_handshake == 2'b00) begin
                nxt_state   = R_DATA;
                done            = 1'b0;
                data_out    = NULL;
        end else if (read_handshake == 2'b01) begin
```

```
                nxt_state   = IDLE;
                done                = DONE;
                data_out    = m_axil_rdata;
        end else begin
                nxt_state   = BAD;
                done                = 1'b0;
                data_out    = NULL;
        end
end

W_ADDR: begin
        m_axil_awaddr       = addr & word_align_mask;
        m_axil_awvalid      = VALID;
        m_axil_wdata        = data_in;
        m_axil_wvalid       = VALID;
        if (write_handshake == 3'b000) begin
                nxt_state   = W_ADDR;
                done                = 1'b0;
        end else if (write_handshake == 3'b100) begin
                nxt_state   = W_DATA;
                done                = 1'b0;
        end else if (write_handshake == 3'b110) begin
                nxt_state   = W_RESP;
                done                = 1'b0;
        end else if (write_handshake == 3'b111) begin
                nxt_state   = IDLE;
                done                = DONE;
        end else begin
                nxt_state   = BAD;
                done                = 1'b0;
        end
end

W_DATA: begin
        m_axil_awaddr       = NULL;
        m_axil_awvalid      = INVALID;
        m_axil_wdata        = data_in;
        m_axil_wvalid       = VALID;
        if (write_handshake == 3'b000) begin
                nxt_state   = W_DATA;
                done                = 1'b0;
        end else if (write_handshake == 3'b010) begin
                nxt_state   = W_RESP;
                done                = 1'b0;
        end else if (write_handshake == 3'b011) begin
                nxt_state   = IDLE;
                done                = DONE;
        end else begin
                nxt_state   = BAD;
                done                = 1'b0;
        end
end

W_RESP: begin
        m_axil_awaddr       = NULL;
        m_axil_awvalid      = INVALID;
        m_axil_wdata        = NULL;
        m_axil_wvalid       = INVALID;
        if (write_handshake == 3'b000) begin
                nxt_state   = W_RESP;
                done                = 1'b0;
        end else if (write_handshake == 3'b001) begin
                nxt_state   = IDLE;
                done                = DONE;
        end else begin
                nxt_state   = BAD;
                done                = 1'b0;
        end
end
```

```
                        BAD: begin
                                nxt_state = BAD;
                        end

                        default: begin
                                data_out        = NULL;
                                done                    = 1'b0;
                                nxt_state       = IDLE;
                                m_axil_awaddr   = NULL;
                                m_axil_awprot   = basic_awport;
                                m_axil_awvalid  = INVALID;
                                m_axil_wdata    = NULL;
                                m_axil_wstrb    = be;
                                m_axil_wvalid   = INVALID;
                                m_axil_bready   = VALID;
                                m_axil_araddr   = NULL;
                                m_axil_arprot   = basic_awport;
                                m_axil_arvalid  = INVALID;
                                m_axil_rready   = VALID;
                        end
                endcase
        end

//////////////////////////// Formal Verifivation ////////////////////////////////

        ////////// read handshake //////////
        property read_handshake_success;
                disable iff (rst)
                @(posedge clk) read_addr_handshake |=> ##[0 : MEM_ACCESS_TIMEOUT] read_data_handshake;
        endproperty

        assert property(read_handshake_success)
                else $error("AXI-L read timeout");
        ////////////////////////////////////


        ////////// write handshake //////////
        property write_addr_handshake_success;
                disable iff (rst)
                @(posedge clk) write_addr_handshake |=> ##[0 : MEM_ACCESS_TIMEOUT] write_resp_handshake;
        endproperty

        assert property(write_addr_handshake_success)
                else $error("AXI-L write addr -> resp timeout");


        property write_data_handshake_success;
                disable iff (rst)
                @(posedge clk) write_data_handshake |=> ##[0 : MEM_ACCESS_TIMEOUT] write_resp_handshake;
        endproperty

        assert property(write_data_handshake_success)
                else $error("AXIL write data -> resp timeout");
        ////////////////////////////////////

//////////////////////////// End Formal Verifivation ////////////////////////////////

endmodule : mem_sys_axil
```

```
`timescale 1 ps / 1 ps

import defines::*;
import mem_defines::*;

module mem_sys_sdram (
        input   logic                   clk_50m,
        input   logic                   clk_100m,
        input   logic                   clk_100m_shift,
        input   logic                   rst_n,

        input   cache_addr_t    addr,                   // still 32 bits
        input   data_t                  data_in,
        input   logic                   wr,
        input   logic                   rd,
        input   logic                   valid,
        input   logic                   [BYTES-1:0] be,     // for write only

        output  data_t                  data_out,
        output  logic                   done,
        output  logic                   sdram_init_done,

        // SDRAM hardware pins
        output  logic                   sdram_clk,
        output  logic                   sdram_cke,
        output  logic                   sdram_cs_n,
        output  logic                   sdram_ras_n,
        output  logic                   sdram_cas_n,
        output  logic   sdram_we_n,
        output  logic   [ 1:0]  sdram_ba,
        output  logic   [12:0]  sdram_addr,
        inout   wire    [15:0]  sdram_data,
        output  logic   [ 1:0]  sdram_dqm
);

// sdram nets
logic                   sdram_wr;
logic                   sdram_rd;
logic                   sdram_valid;
logic                   sdram_done;
sdram_8_wd_t    sdram_line_in;
sdram_8_wd_t    sdram_line_out;
sdram_8_wd_t    sdram_line_buffer;    // updates on sdram done
logic                   [sdram_addr_len-1 : 0] sdram_user_addr;

// dcache nets
logic                   rd_dcache, wr_data_dcache, wr_flag_dcache, en_dcache;
//logic                 hit0_dcache, hit1_dcache;
logic                   hit0_check_dcache, hit1_check_dcache;     // from dacache wire, not data_line_dcache
//logic                 valid0_dcache, valid1_dcache;
//logic                 dirty0_dcache, dirty1_dcache;
//tag_t                 tag0_dcache, tag1_dcache;
logic                   lru_dcache;
flag_line_t             flag_line_in_dcache, flag_line_out_dcache;
data_line_t             data_line_in_dcache, data_line_out_dcache;
data_line_t             data_line_dcache;
flag_line_t             flag_line_dcache;
logic                   load_dcache_line;
logic                   clr_dcache_line;
index_t                 index;
tag_t                   tag;
logic[1:0]              word_off;
data_t                  data_out_dcache;
logic                   done_dcache;
data_line_en_t          data_line_en_dcache;


always_comb begin : data_out_mux
        data_out = data_out_dcache;
        done =          done_dcache;
```

```
end


always_comb begin : dcache_flag_assign
	//valid0_dcache		=		flag_line_dcache.valid0;
	//valid1_dcache		=		flag_line_dcache.valid1;
	//dirty0_dcache		=		flag_line_dcache.dirty0;
	//dirty1_dcache		=		flag_line_dcache.dirty1;
	//tag0_dcache				=		flag_line_dcache.tag0;
	//tag1_dcache				=		flag_line_dcache.tag1;
	lru_dcache		=		flag_line_dcache.lru;
end


always_comb begin : dcache_hit_assign
	if (valid && rd) begin
		//hit0_dcache = ((tag == tag0_dcache) && valid0_dcache) ? 1'b1 : 1'b0;
		//hit1_dcache = ((tag == tag1_dcache) && valid1_dcache) ? 1'b1 : 1'b0;
		hit0_check_dcache = ((tag == flag_line_out_dcache.tag0) && flag_line_out_dcache.valid0) ? 1'b1 : 1'b0;
		hit1_check_dcache = ((tag == flag_line_out_dcache.tag1) && flag_line_out_dcache.valid1) ? 1'b1 : 1'b0;
	end else if (valid && wr) begin
		//hit0_dcache = ((tag == tag0_dcache) || ~valid0_dcache) ? 1'b1 : 1'b0;
		//hit1_dcache = ((tag == tag1_dcache) || ~valid1_dcache) ? 1'b1 : 1'b0;
		hit0_check_dcache = ((tag == flag_line_out_dcache.tag0) || ~flag_line_out_dcache.valid0) ? 1'b1 : 1'b0;
		hit1_check_dcache = ((tag == flag_line_out_dcache.tag1) || ~flag_line_out_dcache.valid1) ? 1'b1 : 1'b0;
	end else begin
		//hit0_dcache = 1'b0;
		//hit1_dcache = 1'b0;
		hit0_check_dcache = 1'b0;
		hit1_check_dcache =1'b0;
	end
end


always_comb begin : cache_addr_assign
	index				= addr.index;
	tag					= addr.tag;
	word_off		= addr.word_off;
end


always_ff @(posedge clk_50m, negedge rst_n) begin
	if (~rst_n) begin
		data_line_dcache <= empty_data_line;
		flag_line_dcache <= empty_flag_line;
	end else if (clr_dcache_line) begin
		data_line_dcache <= empty_data_line;
		flag_line_dcache <= empty_flag_line;
	end else if (load_dcache_line) begin
		data_line_dcache <= data_line_out_dcache;
		flag_line_dcache <= flag_line_out_dcache;
	end else begin
		data_line_dcache <= data_line_dcache;
		flag_line_dcache <= flag_line_dcache;
	end
end


// loads sdram output whenever done,
// no reset signal cuz afraids timing issue
always_ff @( posedge clk_100m) begin : sdram_buffer_load
	if (sdram_done)
		sdram_line_buffer <= sdram_line_out;
	else
		sdram_line_buffer <= sdram_line_buffer;
end


cache dcache(
	// input nets
```

```
		.clk				(clk_50m),
		.en					(en_dcache),
		.index				(index),
		.rd					(rd_dcache),
		.wr_data		(wr_data_dcache),
		.wr_flag		(wr_flag_dcache),
		.flag_line_in		(flag_line_in_dcache),
		.data_line_in		(data_line_in_dcache),
		.data_line_en		(data_line_en_dcache),

		// output nets
		.flag_line_out		(flag_line_out_dcache),
		.data_line_out		(data_line_out_dcache)
);


typedef enum logic[2:0] {
		IDLE,
		CHECK,
		DONE,
		EVICT,
		LOAD,
		LOAD2
} cache_ctrl_state_t;


cache_ctrl_state_t dcache_state, next_dcache_state;
always_ff @( posedge clk_50m, negedge rst_n ) begin
		if (~rst_n)
			dcache_state <= IDLE;
		else
			dcache_state <= next_dcache_state;
end


always_comb begin : dcache_ctrl_fsm
		next_dcache_state	= IDLE;
		clr_dcache_line			= DISABLE;
		load_dcache_line	= DISABLE;
		en_dcache				= DISABLE;
		rd_dcache				= DISABLE;
		wr_data_dcache			= DISABLE;
		wr_flag_dcache			= DISABLE;
		done_dcache					= 1'b0;
		data_out_dcache			= NULL;
		flag_line_in_dcache	= empty_flag_line;
		data_line_in_dcache	= empty_data_line;
		data_line_en_dcache = 32'hffff_ffff;		// defalut enable all

		sdram_valid					= DISABLE;
		sdram_wr				= DISABLE;
		sdram_rd				= DISABLE;
		sdram_line_in			= 127'b0;
		sdram_user_addr			= 24'b0;

		unique case (dcache_state)
			IDLE : begin
				load_dcache_line = ENABLE;
				if ((rd || wr) && valid) begin
					next_dcache_state	= CHECK;
					en_dcache				= ENABLE;
					rd_dcache				= ENABLE;
					clr_dcache_line			= DISABLE;
				end else begin
					next_dcache_state	= IDLE;
					en_dcache				= DISABLE;
					rd_dcache				= DISABLE;
					clr_dcache_line			= ENABLE;
				end
			end
```

```
                CHECK : begin
                        load_dcache_line = ENABLE;
                        if (hit0_check_dcache || hit1_check_dcache) begin
                                next_dcache_state = DONE;
                        end else if (
                                (~hit0_check_dcache && ~hit1_check_dcache) &&
                                (flag_line_out_dcache.valid0 && flag_line_out_dcache.valid1) &&
                                (
                                        (flag_line_out_dcache.lru && flag_line_out_dcache.dirty1) ||
                                        (~flag_line_out_dcache.lru && flag_line_out_dcache.dirty0)
                                )
                        ) begin
                                next_dcache_state = EVICT;
                        end else begin
                                next_dcache_state = LOAD;
                        end
                end

                DONE : begin
                        next_dcache_state       = IDLE;
                        done_dcache                             = 1'b1;
                        en_dcache                       = ENABLE;
                        wr_data_dcache                  = wr;
                        wr_flag_dcache                  = wr || rd;
                        if (wr) begin           // store as if hit
                                data_out_dcache                 = NULL;
                                if (hit0_check_dcache) begin
                                        if (DEBUG_MEM_SYS) begin
                                                $strobe("wrote value:%d to index:%d way:%b word:%d", data_in, index, 1'b0,
word_off);
                                        end
                                        unique case (word_off)
                                                2'b00: begin
                                                        data_line_en_dcache                     =
{{be},{4'b1111},{4'b1111},{4'b1111},

        {4'b1111},{4'b1111},{4'b1111},{4'b1111}};
                                                        data_line_in_dcache.data0w0     = data_in;
                                                        data_line_in_dcache.data0w1     = data_line_out_dcache.data0w1;
                                                        data_line_in_dcache.data0w2     = data_line_out_dcache.data0w2;
                                                        data_line_in_dcache.data0w3     = data_line_out_dcache.data0w3;
                                                        data_line_in_dcache.data1w0     = data_line_out_dcache.data1w0;
                                                        data_line_in_dcache.data1w1     = data_line_out_dcache.data1w1;
                                                        data_line_in_dcache.data1w2     = data_line_out_dcache.data1w2;
                                                        data_line_in_dcache.data1w3     = data_line_out_dcache.data1w3;
                                                        flag_line_in_dcache.valid0      = VALID;
                                                        flag_line_in_dcache.dirty0      = DIRTY;
                                                        flag_line_in_dcache.tag0        = tag;
                                                        flag_line_in_dcache.valid1      = flag_line_out_dcache.valid1;
                                                        flag_line_in_dcache.dirty1      = flag_line_out_dcache.dirty1;
                                                        flag_line_in_dcache.tag1        = flag_line_out_dcache.tag1;
                                                        flag_line_in_dcache.lru                 = 1'b1;
                                                        flag_line_in_dcache.x5                  = 5'b0;
                                                end
                                                2'b01: begin
                                                        data_line_en_dcache                     =
{{4'b1111},{be},{4'b1111},{4'b1111},

        {4'b1111},{4'b1111},{4'b1111},{4'b1111}};
                                                        data_line_in_dcache.data0w0     = data_line_out_dcache.data0w0;
                                                        data_line_in_dcache.data0w1     = data_in;
                                                        data_line_in_dcache.data0w2     = data_line_out_dcache.data0w2;
                                                        data_line_in_dcache.data0w3     = data_line_out_dcache.data0w3;
                                                        data_line_in_dcache.data1w0     = data_line_out_dcache.data1w0;
                                                        data_line_in_dcache.data1w1     = data_line_out_dcache.data1w1;
                                                        data_line_in_dcache.data1w2     = data_line_out_dcache.data1w2;
                                                        data_line_in_dcache.data1w3     = data_line_out_dcache.data1w3;
                                                        flag_line_in_dcache.valid0      = VALID;
                                                        flag_line_in_dcache.dirty0      = DIRTY;
```

```
                                        flag_line_in_dcache.tag0        = tag;
                                        flag_line_in_dcache.valid1      = flag_line_out_dcache.valid1;
                                        flag_line_in_dcache.dirty1      = flag_line_out_dcache.dirty1;
                                        flag_line_in_dcache.tag1        = flag_line_out_dcache.tag1;
                                        flag_line_in_dcache.lru                 = 1'b1;
                                        flag_line_in_dcache.x5                  = 5'b0;
                                end
                                2'b10: begin
                                        data_line_en_dcache                     =
{{4'b1111},{4'b1111},{be},{4'b1111},

        {4'b1111},{4'b1111},{4'b1111},{4'b1111}};
                                        data_line_in_dcache.data0w0     = data_line_out_dcache.data0w0;
                                        data_line_in_dcache.data0w1     = data_line_out_dcache.data0w1;
                                        data_line_in_dcache.data0w2     = data_in;
                                        data_line_in_dcache.data0w3     = data_line_out_dcache.data0w3;
                                        data_line_in_dcache.data1w0     = data_line_out_dcache.data1w0;
                                        data_line_in_dcache.data1w1     = data_line_out_dcache.data1w1;
                                        data_line_in_dcache.data1w2     = data_line_out_dcache.data1w2;
                                        data_line_in_dcache.data1w3     = data_line_out_dcache.data1w3;
                                        flag_line_in_dcache.valid0      = VALID;
                                        flag_line_in_dcache.dirty0      = DIRTY;
                                        flag_line_in_dcache.tag0        = tag;
                                        flag_line_in_dcache.valid1      = flag_line_out_dcache.valid1;
                                        flag_line_in_dcache.dirty1      = flag_line_out_dcache.dirty1;
                                        flag_line_in_dcache.tag1        = flag_line_out_dcache.tag1;
                                        flag_line_in_dcache.lru                 = 1'b1;
                                        flag_line_in_dcache.x5                  = 5'b0;
                                end
                                2'b11: begin
                                        data_line_en_dcache                     =
{{4'b1111},{4'b1111},{4'b1111},{be},

        {4'b1111},{4'b1111},{4'b1111},{4'b1111}};
                                        data_line_in_dcache.data0w0     = data_line_out_dcache.data0w0;
                                        data_line_in_dcache.data0w1     = data_line_out_dcache.data0w1;
                                        data_line_in_dcache.data0w2     = data_line_out_dcache.data0w2;
                                        data_line_in_dcache.data0w3     = data_in;
                                        data_line_in_dcache.data1w0     = data_line_out_dcache.data1w0;
                                        data_line_in_dcache.data1w1     = data_line_out_dcache.data1w1;
                                        data_line_in_dcache.data1w2     = data_line_out_dcache.data1w2;
                                        data_line_in_dcache.data1w3     = data_line_out_dcache.data1w3;
                                        flag_line_in_dcache.valid0      = VALID;
                                        flag_line_in_dcache.dirty0      = DIRTY;
                                        flag_line_in_dcache.tag0        = tag;
                                        flag_line_in_dcache.valid1      = flag_line_out_dcache.valid1;
                                        flag_line_in_dcache.dirty1      = flag_line_out_dcache.dirty1;
                                        flag_line_in_dcache.tag1        = flag_line_out_dcache.tag1;
                                        flag_line_in_dcache.lru                 = 1'b1;
                                        flag_line_in_dcache.x5                  = 5'b0;
                                end
                        endcase
                end else if (hit1_check_dcache && ~hit0_check_dcache) begin
                        if (DEBUG_MEM_SYS) begin
                                $strobe("wrote value:%d to index:%d way:%b word:%d", data_in, index, 1'b1,
word_off);
                        end
                        unique case (word_off)
                                2'b00: begin
                                        data_line_en_dcache                     =
{{4'b1111},{4'b1111},{4'b1111},{4'b1111},

        {be},{4'b1111},{4'b1111},{4'b1111}};
                                        data_line_in_dcache.data0w0     = data_line_out_dcache.data0w0;
                                        data_line_in_dcache.data0w1     = data_line_out_dcache.data0w1;
                                        data_line_in_dcache.data0w2     = data_line_out_dcache.data0w2;
                                        data_line_in_dcache.data0w3     = data_line_out_dcache.data0w3;
                                        data_line_in_dcache.data1w0     = data_in;
                                        data_line_in_dcache.data1w1     = data_line_out_dcache.data1w1;
                                        data_line_in_dcache.data1w2     = data_line_out_dcache.data1w2;
```

```
                data_line_in_dcache.data1w3     = data_line_out_dcache.data1w3;
                flag_line_in_dcache.valid0      = flag_line_out_dcache.valid0;
                flag_line_in_dcache.dirty0      = flag_line_out_dcache.dirty0;
                flag_line_in_dcache.tag0        = flag_line_out_dcache.tag0;
                flag_line_in_dcache.valid1      = VALID;
                flag_line_in_dcache.dirty1      = DIRTY;
                flag_line_in_dcache.tag1        = tag;
                flag_line_in_dcache.lru                 = 1'b0;
                flag_line_in_dcache.x5                  = 5'b0;
        end
        2'b01: begin
                data_line_en_dcache                     =
{{4'b1111},{4'b1111},{4'b1111},{4'b1111},
        {4'b1111},{be},{4'b1111},{4'b1111}};
                data_line_in_dcache.data0w0     = data_line_out_dcache.data0w0;
                data_line_in_dcache.data0w1     = data_line_out_dcache.data0w1;
                data_line_in_dcache.data0w2     = data_line_out_dcache.data0w2;
                data_line_in_dcache.data0w3     = data_line_out_dcache.data0w3;
                data_line_in_dcache.data1w0     = data_line_out_dcache.data1w0;
                data_line_in_dcache.data1w1     = data_in;
                data_line_in_dcache.data1w2     = data_line_out_dcache.data1w2;
                data_line_in_dcache.data1w3     = data_line_out_dcache.data1w3;
                flag_line_in_dcache.valid0      = flag_line_out_dcache.valid0;
                flag_line_in_dcache.dirty0      = flag_line_out_dcache.dirty0;;
                flag_line_in_dcache.tag0        = flag_line_out_dcache.tag0;
                flag_line_in_dcache.valid1      = VALID;
                flag_line_in_dcache.dirty1      = DIRTY;
                flag_line_in_dcache.tag1        = tag;
                flag_line_in_dcache.lru                 = 1'b0;
                flag_line_in_dcache.x5                  = 5'b0;
        end
        2'b10: begin
                data_line_en_dcache                     =
{{4'b1111},{4'b1111},{4'b1111},{4'b1111},
        {4'b1111},{4'b1111},{be},{4'b1111}};
                data_line_in_dcache.data0w0     = data_line_out_dcache.data0w0;
                data_line_in_dcache.data0w1     = data_line_out_dcache.data0w1;
                data_line_in_dcache.data0w2     = data_line_out_dcache.data0w2;
                data_line_in_dcache.data0w3     = data_line_out_dcache.data0w3;
                data_line_in_dcache.data1w0     = data_line_out_dcache.data1w0;
                data_line_in_dcache.data1w1     = data_line_out_dcache.data1w1;
                data_line_in_dcache.data1w2     = data_in;
                data_line_in_dcache.data1w3     = data_line_out_dcache.data1w3;
                flag_line_in_dcache.valid0      = flag_line_out_dcache.valid0;
                flag_line_in_dcache.dirty0      = flag_line_out_dcache.dirty0;
                flag_line_in_dcache.tag0        = flag_line_out_dcache.tag0;
                flag_line_in_dcache.valid1      = VALID;
                flag_line_in_dcache.dirty1      = DIRTY;
                flag_line_in_dcache.tag1        = tag;
                flag_line_in_dcache.lru                 = 1'b0;
                flag_line_in_dcache.x5                  = 5'b0;
        end
        2'b11: begin
                data_line_en_dcache                     =
{{4'b1111},{4'b1111},{4'b1111},{4'b1111},
        {4'b1111},{4'b1111},{4'b1111},{be}};
                data_line_in_dcache.data0w0     = data_line_out_dcache.data0w0;
                data_line_in_dcache.data0w1     = data_line_out_dcache.data0w1;
                data_line_in_dcache.data0w2     = data_line_out_dcache.data0w2;
                data_line_in_dcache.data0w3     = data_line_out_dcache.data0w3;
                data_line_in_dcache.data1w0     = data_line_out_dcache.data1w0;
                data_line_in_dcache.data1w1     = data_line_out_dcache.data1w1;
                data_line_in_dcache.data1w2     = data_line_out_dcache.data1w2;
                data_line_in_dcache.data1w3     = data_in;
                flag_line_in_dcache.valid0      = flag_line_out_dcache.valid0;
                flag_line_in_dcache.dirty0      = flag_line_out_dcache.dirty0;
                flag_line_in_dcache.tag0        = flag_line_out_dcache.tag0;
```

```
                                                flag_line_in_dcache.valid1      = VALID;
                                                flag_line_in_dcache.dirty1      = DIRTY;
                                                flag_line_in_dcache.tag1        = tag;
                                                flag_line_in_dcache.lru                 = 1'b0;
                                                flag_line_in_dcache.x5                  = 5'b0;
                                        end
                                endcase
                        end else begin
                                data_line_in_dcache = empty_data_line;
                                flag_line_in_dcache = empty_flag_line;
                        end
                end else begin          // load as if hit
                        data_line_in_dcache = empty_data_line;
                        if (hit0_check_dcache) begin
                                unique case (word_off)
                                        2'b00: begin
                                                data_out_dcache                         =
data_line_out_dcache.data0w0;
                                                flag_line_in_dcache.valid0      = flag_line_out_dcache.valid0;
                                                flag_line_in_dcache.dirty0      = flag_line_out_dcache.dirty0;
                                                flag_line_in_dcache.tag0        = flag_line_out_dcache.tag0;
                                                flag_line_in_dcache.valid1      = flag_line_out_dcache.valid1;
                                                flag_line_in_dcache.dirty1      = flag_line_out_dcache.dirty1;
                                                flag_line_in_dcache.tag1        = flag_line_out_dcache.tag1;
                                                flag_line_in_dcache.lru                 = 1'b1;
                                                flag_line_in_dcache.x5                  = 5'b0;
                                        end

                                        2'b01: begin
                                                data_out_dcache                         =
data_line_out_dcache.data0w1;
                                                flag_line_in_dcache.valid0      = flag_line_out_dcache.valid0;
                                                flag_line_in_dcache.dirty0      = flag_line_out_dcache.dirty0;
                                                flag_line_in_dcache.tag0        = flag_line_out_dcache.tag0;
                                                flag_line_in_dcache.valid1      = flag_line_out_dcache.valid1;
                                                flag_line_in_dcache.dirty1      = flag_line_out_dcache.dirty1;
                                                flag_line_in_dcache.tag1        = flag_line_out_dcache.tag1;
                                                flag_line_in_dcache.lru                 = 1'b1;
                                                flag_line_in_dcache.x5                  = 5'b0;
                                        end

                                        2'b10: begin
                                                data_out_dcache                         =
data_line_out_dcache.data0w2;
                                                flag_line_in_dcache.valid0      = flag_line_out_dcache.valid0;
                                                flag_line_in_dcache.dirty0      = flag_line_out_dcache.dirty0;
                                                flag_line_in_dcache.tag0        = flag_line_out_dcache.tag0;
                                                flag_line_in_dcache.valid1      = flag_line_out_dcache.valid1;
                                                flag_line_in_dcache.dirty1      = flag_line_out_dcache.dirty1;
                                                flag_line_in_dcache.tag1        = flag_line_out_dcache.tag1;
                                                flag_line_in_dcache.lru                 = 1'b1;
                                                flag_line_in_dcache.x5                  = 5'b0;
                                        end

                                        2'b11: begin
                                                data_out_dcache                         =
data_line_out_dcache.data0w3;
                                                flag_line_in_dcache.valid0      = flag_line_out_dcache.valid0;
                                                flag_line_in_dcache.dirty0      = flag_line_out_dcache.dirty0;
                                                flag_line_in_dcache.tag0        = flag_line_out_dcache.tag0;
                                                flag_line_in_dcache.valid1      = flag_line_out_dcache.valid1;
                                                flag_line_in_dcache.dirty1      = flag_line_out_dcache.dirty1;
                                                flag_line_in_dcache.tag1        = flag_line_out_dcache.tag1;
                                                flag_line_in_dcache.lru                 = 1'b1;
                                                flag_line_in_dcache.x5                  = 5'b0;
                                        end
                                endcase
                        end else if (hit1_check_dcache && ~hit0_check_dcache) begin
                                unique case (word_off)
                                        2'b00: begin
```

```
                                                data_out_dcache                 =
data_line_out_dcache.data1w0;
                                                flag_line_in_dcache.valid0      = flag_line_out_dcache.valid0;
                                                flag_line_in_dcache.dirty0      = flag_line_out_dcache.dirty0;
                                                flag_line_in_dcache.tag0        = flag_line_out_dcache.tag0;
                                                flag_line_in_dcache.valid1      = flag_line_out_dcache.valid1;
                                                flag_line_in_dcache.dirty1      = flag_line_out_dcache.dirty1;
                                                flag_line_in_dcache.tag1        = flag_line_out_dcache.tag1;
                                                flag_line_in_dcache.lru                 = 1'b0;
                                                flag_line_in_dcache.x5                  = 5'b0;
                                        end

                                        2'b01: begin
                                                data_out_dcache                 =
data_line_out_dcache.data1w1;
                                                flag_line_in_dcache.valid0      = flag_line_out_dcache.valid0;
                                                flag_line_in_dcache.dirty0      = flag_line_out_dcache.dirty0;
                                                flag_line_in_dcache.tag0        = flag_line_out_dcache.tag0;
                                                flag_line_in_dcache.valid1      = flag_line_out_dcache.valid1;
                                                flag_line_in_dcache.dirty1      = flag_line_out_dcache.dirty1;
                                                flag_line_in_dcache.tag1        = flag_line_out_dcache.tag1;
                                                flag_line_in_dcache.lru                 = 1'b0;
                                                flag_line_in_dcache.x5                  = 5'b0;
                                        end

                                        2'b10: begin
                                                data_out_dcache                 =
data_line_out_dcache.data1w2;
                                                flag_line_in_dcache.valid0      = flag_line_out_dcache.valid0;
                                                flag_line_in_dcache.dirty0      = flag_line_out_dcache.dirty0;
                                                flag_line_in_dcache.tag0        = flag_line_out_dcache.tag0;
                                                flag_line_in_dcache.valid1      = flag_line_out_dcache.valid1;
                                                flag_line_in_dcache.dirty1      = flag_line_out_dcache.dirty1;
                                                flag_line_in_dcache.tag1        = flag_line_out_dcache.tag1;
                                                flag_line_in_dcache.lru                 = 1'b0;
                                                flag_line_in_dcache.x5                  = 5'b0;
                                        end

                                        2'b11: begin
                                                data_out_dcache                 =
data_line_out_dcache.data1w3;
                                                flag_line_in_dcache.valid0      = flag_line_out_dcache.valid0;
                                                flag_line_in_dcache.dirty0      = flag_line_out_dcache.dirty0;
                                                flag_line_in_dcache.tag0        = flag_line_out_dcache.tag0;
                                                flag_line_in_dcache.valid1      = flag_line_out_dcache.valid1;
                                                flag_line_in_dcache.dirty1      = flag_line_out_dcache.dirty1;
                                                flag_line_in_dcache.tag1        = flag_line_out_dcache.tag1;
                                                flag_line_in_dcache.lru                 = 1'b0;
                                                flag_line_in_dcache.x5                  = 5'b0;
                                        end
                                endcase
                        end else begin
                                data_out_dcache = NULL;
                                flag_line_in_dcache = empty_flag_line;
                        end
                end
        end

        EVICT : begin
                sdram_valid                     = ENABLE;
                sdram_wr                = ENABLE;
                sdram_user_addr         = (~lru_dcache) ?       {{flag_line_dcache.tag0[11:0]},index,{3'b0}}:

        {{flag_line_dcache.tag1[11:0]},index,{3'b0}};
                if (sdram_done) begin
                        // no need to update flags
                        // cuz flags will be updated in load stage
                        // right?...
                        next_dcache_state       = LOAD;
                        if (DEBUG_MEM_SYS) begin
```

```
                                $display("evicted way %b", lru_dcache);
                        end
                end else begin
                        next_dcache_state       = EVICT;
                        sdram_line_in           = (~lru_dcache) ? {
                                                  {data_line_dcache.data0w0},
                                                  {data_line_dcache.data0w1},
                                                  {data_line_dcache.data0w2},
                                                  {data_line_dcache.data0w3}
                                                } : {
                                                  {data_line_dcache.data1w0},
                                                  {data_line_dcache.data1w1},
                                                  {data_line_dcache.data1w2},
                                                  {data_line_dcache.data1w3}
                                                };
                        end
        end

        LOAD : begin
                sdram_valid                                                             = ENABLE;
                sdram_rd                                                        = ENABLE;
                // not {{addr[24:4]},{4'b0}} because this addr is in 16bit word, not 8 bit word
                sdram_user_addr                                                 = {{addr[24:4]},{3'b0}};
                if (sdram_done) begin
                        next_dcache_state                               = LOAD2;
                        en_dcache                                               = ENABLE;
                        wr_data_dcache                                  = ENABLE;
                        wr_flag_dcache                                  = ENABLE;
                        if (~lru_dcache) begin// overwrite way 0
                                data_line_in_dcache.data0w0     = {{sdram_line_buffer.w0},{sdram_line_buffer.w1}};
                                data_line_in_dcache.data0w1     = {{sdram_line_buffer.w2},{sdram_line_buffer.w3}};
                                data_line_in_dcache.data0w2     = {{sdram_line_buffer.w4},{sdram_line_buffer.w5}};
                                data_line_in_dcache.data0w3     = {{sdram_line_buffer.w6},{sdram_line_buffer.w7}};
                                data_line_in_dcache.data1w0     = data_line_dcache.data1w0;
                                data_line_in_dcache.data1w1     = data_line_dcache.data1w1;
                                data_line_in_dcache.data1w2     = data_line_dcache.data1w2;
                                data_line_in_dcache.data1w3     = data_line_dcache.data1w3;
                                flag_line_in_dcache.valid0      = VALID;
                                flag_line_in_dcache.dirty0      = CLEAN;
                                flag_line_in_dcache.tag0        = tag;
                                flag_line_in_dcache.valid1      = flag_line_dcache.valid1;
                                flag_line_in_dcache.dirty1      = flag_line_dcache.dirty1;
                                flag_line_in_dcache.tag1        = flag_line_dcache.tag1;
                                flag_line_in_dcache.lru                 = flag_line_dcache.lru;         // only filp
in DONE state
                                flag_line_in_dcache.x5                  = 5'b0;
                        end else begin                          // overwrite way 1
                                data_line_in_dcache.data0w0     = data_line_dcache.data0w0;
                                data_line_in_dcache.data0w1     = data_line_dcache.data0w1;
                                data_line_in_dcache.data0w2     = data_line_dcache.data0w2;
                                data_line_in_dcache.data0w3     = data_line_dcache.data0w3;
                                data_line_in_dcache.data1w0     = {{sdram_line_buffer.w0},{sdram_line_buffer.w1}};
                                data_line_in_dcache.data1w1     = {{sdram_line_buffer.w2},{sdram_line_buffer.w3}};
                                data_line_in_dcache.data1w2     = {{sdram_line_buffer.w4},{sdram_line_buffer.w5}};
                                data_line_in_dcache.data1w3     = {{sdram_line_buffer.w6},{sdram_line_buffer.w7}};
                                flag_line_in_dcache.valid0      = flag_line_dcache.valid0;
                                flag_line_in_dcache.dirty0      = flag_line_dcache.dirty0;
                                flag_line_in_dcache.tag0        = flag_line_dcache.tag0;
                                flag_line_in_dcache.valid1      = VALID;
                                flag_line_in_dcache.dirty1      = CLEAN;
                                flag_line_in_dcache.tag1        = tag;
                                flag_line_in_dcache.lru                 = flag_line_dcache.lru;         // only filp
in DONE state
                                flag_line_in_dcache.x5                  = 5'b0;
                        end
                end else begin
                        next_dcache_state                       = LOAD;
                        en_dcache                                       = DISABLE;
                        wr_data_dcache                          = DISABLE;
                        wr_flag_dcache                          = DISABLE;
```

```
                        end
                end

                LOAD2: begin
                        next_dcache_state               = DONE;
                        en_dcache                               = ENABLE;
                        rd_dcache                               = ENABLE;
                end

                default: begin
                        next_dcache_state               = IDLE;
                end
        endcase
end


// top level of a sdram controller
sdram sdram_ctrl_inst(
   .clk_50m                             (clk_50m),
        .clk_100m                       (clk_100m),
        .clk_100m_shift                 (clk_100m_shift),
   .rst_n                               (rst_n),

   .sdram_clk                           (sdram_clk),
   .sdram_cke                           (sdram_cke),
   .sdram_cs_n                          (sdram_cs_n),
   .sdram_ras_n                 (sdram_ras_n),
   .sdram_cas_n                 (sdram_cas_n),
   .sdram_we_n                          (sdram_we_n),
   .sdram_ba                            (sdram_ba),
   .sdram_addr                          (sdram_addr),
   .sdram_data                          (sdram_data),
   .sdram_dqm                           (sdram_dqm),

        // user control interface
        // a transaction is complete when valid && done
        .addr                                   (sdram_user_addr),
        .wr                                             (sdram_wr),
        .rd                                             (sdram_rd),
        .valid                                  (sdram_valid),
        .data_line_in                   (sdram_line_in),
        .data_line_out                  (sdram_line_out),
        .done                                   (sdram_done),
        .sdram_init_done        (sdram_init_done)
);


// synthesis translate_off
// TODO: change this to debug module
always_ff @(negedge clk_50m) begin : checks
        assert ( ~( (dcache_state == DONE) && ~(hit0_check_dcache || hit1_check_dcache) && sdram_init_done ) )
        else begin
                $error("Assertion hit_check failed! at time=%t", $time);
                $strobe("hit0 = %b, hit1=%b", hit0_check_dcache, hit1_check_dcache);
                //$stop();
        end
end
// synthesis translate_on

endmodule: mem_sys_sdram
```

```
import defines::*;
import mem_defines::*;

// synopsys translate_off
`timescale 1 ns / 1 ps
// synopsys translate_on


// TODO: add - save how to forward
// addi x1, x1, 4; sb x1, 0(x1)

module memory (
        input   logic                           clk,
        input   logic                           rst_n,
        input   data_t                          addr,
        input   data_t                          data_in_raw,
        input   data_t                          mem_mem_fwd_data,
        input   mem_fwd_sel_t           fwd_m2m,                // mem to mem forwarding
        input   instr_t                         instr,

        output  data_t                          data_out,
        output  data_t                          data_in_final,  // for debug
        output  logic                           sdram_init_done,
        output  logic                           done,

        axil_interface.axil_master      axil_bus
);

        // control signals
        opcode_t        opcode;
        funct3_t        funct3;
        funct5_t        funct5;
        logic                   wren, rden;
        logic                   is_st, is_ld;
        logic                   mem_access_done;
        logic                   valid;

        // debug signals
        logic                   addr_misalign;
        logic                   misalign_trap;

        // atomic contril signals
        instr_a_t       instr_a;
        logic                   is_atomic, is_lr, is_sc;
        logic                   update;         // enable exclusive monitor for a single cycle
        logic                   sc_success;

        always_comb begin : signal_assign
                instr_a         =       instr_a_t'(instr);
                opcode          =       instr_a.opcode;
                funct3          =       instr_a.funct3;
                funct5          =       instr_a.funct5;
                is_atomic  =    opcode == ATOMIC;
                is_lr           =       is_atomic && funct5 == LR;
                is_sc           =       is_atomic && funct5 == SC;
                is_ld           =       opcode == LOAD;
                is_st           =       opcode == STORE;
                valid           =       wren || rden;
                sdram_init_done = DONE;
        end


        typedef enum logic[2:0] {
                IDLE,           // no memory access
                REGULAR,        // actual memory access
                AMO1, AMO2      // other automic memory access
        } memory_ctrl_state_t;


        memory_ctrl_state_t state, nxt_state;
```

```
always_ff @( posedge clk, negedge rst_n ) begin
        if (~rst_n)
                state <= IDLE;
        else
                state <= nxt_state;
end

always_comb begin : memory_fsm_ctrl
        nxt_state   = IDLE;
        rden                = DISABLE;
        wren                = DISABLE;
        update              = DISABLE;
        done                = 1'b0;
        unique case (state)
                IDLE        : begin
                        if (is_ld) begin
                                nxt_state   = REGULAR;
                                rden                = ENABLE;
                                wren                = DISABLE;
                                done                = 1'b0;
                                update              = DISABLE;
                        end else if (is_st) begin
                                nxt_state   = REGULAR;
                                rden                = DISABLE;
                                wren                = ENABLE;
                                done                = 1'b0;
                                update              = DISABLE;
                        end else if (is_lr) begin
                                nxt_state   = REGULAR;
                                rden                = ENABLE;
                                wren                = DISABLE;
                                done                = 1'b0;
                                update              = DISABLE;
                        end else if (is_sc) begin
                                rden                = DISABLE;
                                if (sc_success) begin // success, write 0 to rd
                                        nxt_state   = REGULAR;
                                        wren                = ENABLE;
                                        done                = 1'b0;
                                        update              = DISABLE;
                                end else begin          // fail, write non-zero value to rd
                                        nxt_state   = IDLE;
                                        wren                = DISABLE;
                                        done                = 1'b1;
                                        update              = ENABLE;
                                end
                        end else begin
                                nxt_state   = IDLE;
                                rden                = DISABLE;
                                wren                = DISABLE;
                                done                = 1'b0;
                                update              = DISABLE;
                        end
                end

                REGULAR             : begin
                        if (mem_access_done) begin
                                nxt_state   = IDLE;
                                done                = 1'b1;
                                wren                = is_st || is_sc;
                                rden                = is_ld || is_lr;
                                update              = is_sc || is_lr || is_st;
                        end else begin
                                nxt_state   = REGULAR;
                                done                = 1'b0;
                                wren                = is_st || is_sc;
                                rden                = is_ld || is_lr;
                                update              = DISABLE;
                        end
                end
```

```
                        default     : begin
                                nxt_state   = IDLE;
                                rden                = DISABLE;
                                wren                = DISABLE;
                                update              = DISABLE;
                                done                = 1'b0;
                        end

                endcase
        end


        logic[BYTES-1:0] be;
        always_comb begin : byte_enable_pharse
                if (wren) begin
                        unique case (funct3)
                                SB: begin
                                        unique case (addr[1:0])
                                                2'b00: be = 4'b1000;
                                                2'b01: be = 4'b0100;
                                                2'b10: be = 4'b0010;
                                                2'b11: be = 4'b0001;
                                        endcase
                                end
                                SH: begin
                                        unique case (addr[1])
                                                2'b0: be = 4'b1100;
                                                2'b1: be = 4'b0011;
                                        endcase
                                end
                                SW: begin
                                        be = 4'b1111;
                                end
                                default: be = 4'b0;
                        endcase
                end else begin
                        be = 4'b1111;
                end
        end

        // may need to switch endianess for storing in of memory depending on endianess
        data_t data_in;                                 // after fwd
        //data_t data_in_final;         // decleared as output, after possible endian switch
        data_t data_out_mem;                    // data just out of mem, blue raw

        always_comb begin : sel_fwd_data
                data_in = (fwd_m2m == WB_MEM_SEL) ? mem_mem_fwd_data : data_in_raw;
        end

        // switch data endianess to little when storing if necessary
        always_comb begin : switch_endian_in
                if (wren) begin
                        case (funct3)
                                SB: begin
                                        case (addr[1:0])
                                                2'b00: data_in_final = {{data_in[7:0]},{8'b0},{8'b0},{8'b0}};
                                                2'b01: data_in_final = {{8'b0},{data_in[7:0]},{8'b0},{8'b0}};
                                                2'b10: data_in_final = {{8'b0},{8'b0},{data_in[7:0]},{8'b0}};
                                                2'b11: data_in_final = {{8'b0},{8'b0},{8'b0},{data_in[7:0]}};
                                        endcase
                                end

                                SH: begin
                                        case (addr[1])
                                                1'b0: begin
                                                        if (ENDIANESS == BIG_ENDIAN)
                                                                data_in_final                                   =
{{data_in[15:8]},{data_in[7:0]},{16'b0}};
                                                        else
```

```
                                                                        data_in_final                                   =
{{data_in[7:0]},{data_in[15:8]},{16'b0}};
                                                        end
                                                        1'b1: begin
                                                                if (ENDIANESS == BIG_ENDIAN)
                                                                        data_in_final                                   =
{{16'b0},{data_in[15:8]},{data_in[7:0]}};
                                                                else
                                                                        data_in_final                                   =
{{16'b0},{data_in[7:0]},{data_in[15:8]}};
                                                        end
                                                endcase
                                        end

                                        SW: begin
                                                if (ENDIANESS == BIG_ENDIAN)
                                                        data_in_final = data_in;
                                                else
                                                        data_in_final = swap_endian(data_in);
                                        end

                                        default: begin
                                                data_in_final = NULL;
                                        end
                                endcase
                        end else begin
                                data_in_final = NULL;
                        end
        end

        word_t d;
        always_comb begin // abbr for shorter code
                d = word_t'(data_out_mem);
        end

        mem_out_sel_t mem_out_sel;
        always_comb begin : mem_out_sel_assign
                mem_out_sel =           (is_ld || is_st) ? REGULAR_LD_OUT :
                                                        (is_lr) ? LOAD_CONDITIONAL_OUT :
                                                        (is_sc && sc_success) ? STORE_CONDITIONAL_SUC_OUT :
                                                        (is_sc && ~sc_success) ? STORE_CONDITIONAL_FAIL_OUT:
                                                        (is_atomic) ? AMO_INSTR_OUT:
                                                        NULL_OUT;
        end

        always_comb begin : output_mask_pharse
                unique case (mem_out_sel)
                        REGULAR_LD_OUT: begin
                                unique case (funct3)
                                        LB:             begin
                                                case (addr[1:0])
                                                        2'b00:  data_out = sign_extend_b(d.b0);
                                                        2'b01:  data_out = sign_extend_b(d.b1);
                                                        2'b10:  data_out = sign_extend_b(d.b2);
                                                        2'b11:  data_out = sign_extend_b(d.b3);
                                                endcase
                                        end

                                        LH:             begin
                                                case (addr[1])
                                                        1'b0: begin
                                                                if (ENDIANESS == BIG_ENDIAN)
                                                                        data_out = sign_extend_h({{d.b0},{d.b1}});
                                                                else
                                                                        data_out = sign_extend_h({{d.b1},{d.b0}});
                                                        end
                                                        1'b1: begin
                                                                if (ENDIANESS == BIG_ENDIAN)
                                                                        data_out = sign_extend_h({{d.b2},{d.b3}});
                                                                else
```

```
                                                data_out = sign_extend_h({{d.b3},{d.b2}});
                                        end
                                endcase
                        end

                        LW:             begin
                                if (ENDIANESS == BIG_ENDIAN)
                                        data_out = {{d.b0},{d.b1},{d.b2},{d.b3}};
                                else
                                        data_out = {{d.b3},{d.b2},{d.b1},{d.b0}};
                        end

                        LBU:    begin
                                case (addr[1:0])
                                        2'b00:  data_out = zero_extend_b(d.b0);
                                        2'b01:  data_out = zero_extend_b(d.b1);
                                        2'b10:  data_out = zero_extend_b(d.b2);
                                        2'b11:  data_out = zero_extend_b(d.b3);
                                endcase
                        end

                        LHU:    begin
                                case (addr[1])
                                        1'b0: begin
                                                if (ENDIANESS == BIG_ENDIAN)
                                                        data_out = zero_extend_h({{d.b0},{d.b1}});
                                                else
                                                        data_out = zero_extend_h({{d.b1},{d.b0}});
                                        end
                                        1'b1: begin
                                                if (ENDIANESS == BIG_ENDIAN)
                                                        data_out = zero_extend_h({{d.b2},{d.b3}});
                                                else
                                                        data_out = zero_extend_h({{d.b3},{d.b2}});
                                        end
                                endcase
                        end

                        default:begin
                                data_out = NULL;
                        end
        endcase
end

LOAD_CONDITIONAL_OUT: begin
        if (ENDIANESS == BIG_ENDIAN)
                data_out = {{d.b0},{d.b1},{d.b2},{d.b3}};
        else
                data_out = {{d.b3},{d.b2},{d.b1},{d.b0}};
end

STORE_CONDITIONAL_SUC_OUT: begin
        data_out = SC_SUCCESS_CODE;
end

STORE_CONDITIONAL_FAIL_OUT: begin
        data_out = SC_FAIL_ECODE;
end

AMO_INSTR_OUT: begin
        data_out = NULL;
end

NULL_OUT: begin
        data_out = NULL;
end

default: begin
        data_out = NULL;
end
```

```
        endcase
end


// exclusive monitor, used to see if a conditional store will success
exclusive_monitor #(
        .RES_ENTRY_CNT  (MAX_NEST_LOCK)
) exclusive_monitor_inst (
        .clk                    (clk),
        .rst_n                  (rst_n),
        .instr                  (instr_a_t'(instr)),
        .addr                   (addr),
        .mem_wr                 (wren),
        .update                 (update),
        .success        (sc_success)
);

mem_sys_axil memory_system (
        .clk                    (clk),
        .rst_n                  (rst_n),

        .addr                   (addr),
        .data_in        (data_in_final),
        .wr                             (wren),
        .rd                             (rden),
        .valid                  (valid),
        .be                             (be),

        .data_out       (data_out_mem),
        .done                   (mem_access_done),

        .m_axil_awaddr  (axil_bus.axil_awaddr),
        .m_axil_awprot  (axil_bus.axil_awprot),
        .m_axil_awvalid (axil_bus.axil_awvalid),
        .m_axil_awready (axil_bus.axil_awready),
        .m_axil_wdata   (axil_bus.axil_wdata),
        .m_axil_wstrb   (axil_bus.axil_wstrb),
        .m_axil_wvalid  (axil_bus.axil_wvalid),
        .m_axil_wready  (axil_bus.axil_wready),
        .m_axil_bresp   (axil_bus.axil_bresp),
        .m_axil_bvalid  (axil_bus.axil_bvalid),
        .m_axil_bready  (axil_bus.axil_bready),
        .m_axil_araddr  (axil_bus.axil_araddr),
        .m_axil_arprot  (axil_bus.axil_arprot),
        .m_axil_arvalid (axil_bus.axil_arvalid),
        .m_axil_arready (axil_bus.axil_arready),
        .m_axil_rdata   (axil_bus.axil_rdata),
        .m_axil_rresp   (axil_bus.axil_rresp),
        .m_axil_rvalid  (axil_bus.axil_rvalid),
        .m_axil_rready  (axil_bus.axil_rready)
);


////////////////////////// formal verification //////////////////////////

//// misalign assersion ////
always_comb begin
        unique case (funct3)
                LB:             addr_misalign = 1'b0;
                LH:             addr_misalign = addr[0];
                LW:             addr_misalign = |addr[1:0];
                LBU:    addr_misalign = 1'b0;
                LHU:    addr_misalign = addr[0];
                default:addr_misalign = 1'b0;
        endcase
        misalign_trap = addr_misalign && (rden || wren) && valid;
end

property memory_address_misalign;
```

```
                disable iff (~rst_n)
                @(posedge clk) ~misalign_trap;
        endproperty

        assert property (memory_address_misalign)
                else $error("misaligned memory access");
        ////////////////////////


        //// write and rd in the same cycle ////
        property memory_conflicting_access;
                disable iff (~rst_n)
                @(posedge clk) ~(rden && wren);
        endproperty

        assert property (memory_conflicting_access)
                else $error("read and write mem at same access");
        ////////////////////////////////////////

/*
        //// assume all access are below max physical memory ////
        logic                           max_phy_access;
        logic                           max_phy_access_trap;
        always_comb begin
                max_phy_access = $signed(addr) >= $signed(MAX_PHY_ADDR);
                max_phy_access_trap = max_phy_access && (rden || wren);
        end

        property memory_max_phy_access;
                disable iff (~rst_n)
                @(posedge clk) ~max_phy_access_trap;
        endproperty

        assert property (memory_max_phy_access)
                else $error("access exceed physical memory boundary");
        ////////////////////////////////////////////////////////
*/
endmodule : memory
```

```
`timescale 1ns / 1ps

import defines::*;

module uart_axil_tb ();

	logic clk, rst, rst_n;
	assign rst = ~rst_n;

	logic TX, RX;

	axi_lite_interface axil_bus (
		.clk	(clk),
		.rst	(rst)
	);

	clkrst #(.FREQ(FREQ)) clkrst_inst (
		.clk	(clk),
		.rst_n	(rst_n),
		.go			()
	);

	data_t addr, data_in, data_out;
	logic wren, rden, valid, done;

	mem_sys_axil memory_system (
		.clk				(clk),
		.rst_n				(rst_n),

		.addr				(addr),
		.data_in		(data_in),
		.wr					(wren),
		.rd					(rden),
		.valid				(valid),
		.be					(4'b1111),

		.data_out		(data_out),
		.done				(done),

		.m_axil_clk		(axil_bus.m_axil_clk),
		.m_axil_rst			(axil_bus.m_axil_rst),
		.m_axil_awaddr	(axil_bus.m_axil_awaddr),
		.m_axil_awprot	(axil_bus.m_axil_awprot),
		.m_axil_awvalid	(axil_bus.m_axil_awvalid),
		.m_axil_awready	(axil_bus.m_axil_awready),
		.m_axil_wdata	(axil_bus.m_axil_wdata),
		.m_axil_wstrb	(axil_bus.m_axil_wstrb),
		.m_axil_wvalid	(axil_bus.m_axil_wvalid),
		.m_axil_wready	(axil_bus.m_axil_wready),
		.m_axil_bresp	(axil_bus.m_axil_bresp),
		.m_axil_bvalid	(axil_bus.m_axil_bvalid),
		.m_axil_bready	(axil_bus.m_axil_bready),
		.m_axil_araddr	(axil_bus.m_axil_araddr),
		.m_axil_arprot	(axil_bus.m_axil_arprot),
		.m_axil_arvalid	(axil_bus.m_axil_arvalid),
		.m_axil_arready	(axil_bus.m_axil_arready),
		.m_axil_rdata	(axil_bus.m_axil_rdata),
		.m_axil_rresp	(axil_bus.m_axil_rresp),
		.m_axil_rvalid	(axil_bus.m_axil_rvalid),
		.m_axil_rready	(axil_bus.m_axil_rready)
	);

	uart_axil my_uart (
		.clk			(clk),
		.rst			(rst),
		.awvalid_i	(axil_bus.s_axil_awvalid),
		.awaddr_i	(axil_bus.s_axil_awaddr),
		.wvalid_i	(axil_bus.s_axil_wvalid),
		.wdata_i	(axil_bus.s_axil_wdata),
		.wstrb_i	(axil_bus.s_axil_wstrb),
```

```
                .bready_i   (axil_bus.s_axil_bready),
                .arvalid_i  (axil_bus.s_axil_arvalid),
                .araddr_i   (axil_bus.s_axil_araddr),
                .rready_i   (axil_bus.s_axil_rready),

                .awready_o          (axil_bus.s_axil_awready),
                .wready_o (axil_bus.s_axil_wready),
                .bvalid_o   (axil_bus.s_axil_bvalid),
                .bresp_o    (axil_bus.s_axil_bresp),
                .arready_o (axil_bus.s_axil_arready),
                .rvalid_o   (axil_bus.s_axil_rvalid),
                .rdata_o    (axil_bus.s_axil_rdata),
                .rresp_o    (axil_bus.s_axil_rresp),

                .awprot_i   (axil_bus.s_axil_awprot),
                .arprot_i   (axil_bus.s_axil_arprot),

                .uart_rx    (RX),
                .uart_tx    (TX)
        );

initial begin
        init ();
        write_simp(32'h0, 32'h7);
        repeat(10000) @(posedge clk);
end

task init ();
        addr = 0;
        data_in = 0;
        wren = 0;
        rden = 0;
        valid = 0;
        repeat(100) @(posedge clk);
endtask

task  write_simp(input data_t a, input data_t d);
        fork
                begin
                        repeat(100) @(posedge clk);
                        $display("write timeout!");
                        $stop();
                end

                begin
                wren        = 1;
                valid       = 1;
                addr        = a;
                data_in     = d;
                @(posedge done);
                @(posedge clk);
                valid = 0;
                wren = 0;
                end
        join_any
        disable fork;
endtask

endmodule : uart_axil_tb

module clkrst #(
        FREQ = FREQ
) (
        output logic clk,
        output logic rst_n,
        output logic go
);

        localparam period = 1e12/FREQ; // in ps
        localparam half_period = period/2;
```

```
        initial begin
                clk                     = 1'b0;
                rst_n           = 1'b0;
                go                      = 1'b0;
                repeat(5) @(negedge clk);
                #100;
                rst_n           = 1'b1;
                repeat(233) @(negedge clk);
                go                      = 1'b1;
                @(negedge clk);
                go                      = 1'b0;
        end

        always #half_period begin
    clk = ~clk;
  end

endmodule : clkrst
```

```
module uart_axil_wrapper # (
	parameter  CLK_FREQ		= 5e7,
	parameter  UART_BPS		= 9600,
	parameter  FIFO_WIDTH_TX	= 10,	// FIFO depth = 2^WIDTH
	parameter  FIFO_WIDTH_RX	= 5,
	parameter  ADDR_WIDTH		= 4
) (
	input	logic				clk,
	input	logic				rst,

	// UART TX/RX
	input	logic				uart_rx,
	output	logic				uart_tx,
	input	logic				uart_cts,	// high: master cannot take input
	output	logic				uart_rts,	// high: we cannot take input

	axil_interface.axil_slave	s00
);

	uart_axil # (
		.CLK_FREQ			(CLK_FREQ),
		.UART_BPS			(UART_BPS),
		.FIFO_WIDTH_TX		(FIFO_WIDTH_TX),
		.FIFO_WIDTH_RX		(FIFO_WIDTH_RX),
		.ADDR_WIDTH			(ADDR_WIDTH)
	) uarts (
		.clk				(clk),
		.rst				(rst),

		// UARTS
		.uart_rx			(uart_rx),
		.uart_tx			(uart_tx),
		.uart_cts			(uart_cts),
		.uart_rts			(uart_rts),

		// AXI inputs
		.awvalid_i			(s00.axil_awvalid),
		.awaddr_i			(s00.axil_awaddr),
		.wvalid_i			(s00.axil_wvalid),
		.wdata_i			(s00.axil_wdata),
		.wstrb_i			(s00.axil_wstrb),
		.bready_i			(s00.axil_bready),
		.arvalid_i			(s00.axil_arvalid),
		.araddr_i			(s00.axil_araddr),
		.rready_i			(s00.axil_rready),

		// Outputs
		.awready_o			(s00.axil_awready),
		.wready_o			(s00.axil_wready),
		.bvalid_o			(s00.axil_bvalid),
		.bresp_o			(s00.axil_bresp),
		.arready_o			(s00.axil_arready),
		.rvalid_o			(s00.axil_rvalid),
		.rdata_o			(s00.axil_rdata),
		.rresp_o			(s00.axil_rresp),

		// unused AXI signal
		.awprot_i			(s00.axil_awprot),
		.arprot_i			(s00.axil_arprot)
	);

endmodule : uart_axil_wrapper
```

```
import defines::*;
import pref_defines::*;
import axi_defines::*;


module uart_axil #(
        parameter  CLK_FREQ                                     = 5e7,
        parameter  UART_BPS                                     = 9600,
        parameter  FIFO_WIDTH_TX                = 10,   // FIFO depth = 2^WIDTH
        parameter  FIFO_WIDTH_RX                = 5,
        parameter  ADDR_WIDTH                                   = 4
) (
        // Inputs
        input   logic                                                   clk,
        input   logic                                                   rst,

        // axi-lite
        input   logic                                                   awvalid_i,
        input   logic   [ADDR_WIDTH - 1:0]      awaddr_i,
        input   logic                                                   wvalid_i,
        input   logic   [31:0]                                  wdata_i,
        input   logic   [ 3:0]                                  wstrb_i,
        input   logic                                                   bready_i,
        input   logic                                                   arvalid_i,
        input   logic   [ADDR_WIDTH - 1:0]      araddr_i,
        input   logic                                                   rready_i,

        // Outputs
        output  logic                                                   awready_o,
        output  logic                                                   wready_o,
        output  logic                                                   bvalid_o,
        output  logic   [1:0]                                   bresp_o,
        output  logic                                                   arready_o,
        output  logic                                                   rvalid_o,
        output  logic   [31:0]                                  rdata_o,
        output  logic   [ 1:0]                                  rresp_o,

        // unused AXI signal
        input   logic   [ 2:0]                                  awprot_i,
        input   logic   [ 2:0]                                  arprot_i,

        // UART TX/RX
        input   logic                                                   uart_rx,
        output  logic                                                   uart_tx,
        input   logic                                                   uart_cts,   // high: master cannot take input
        output  logic                                                   uart_rts    // high: we cannot take input
);

        assign uart_rts = 1'b0; // disable flow control for now

        // UART module wire
        logic                           uart_send_data, rx_done_raw, rx_done, tx_done;
        logic   [7:0]   tx_data, rx_data;

        // SIMP bus wire
        logic   [31:0]  simp_addr;
        logic   [31:0]  simp_data_in;
        logic                   simp_wr;
        logic                   simp_rd;
        logic                   simp_valid;
        logic   [ 3:0]  simp_be;

        logic   [31:0]  simp_data_out;
        logic                   simp_done;

        // operation define
        logic                   op_write_tx_fifo, op_write_tx_fifo_done;
        logic                   op_read_rx_fifo, op_read_rx_fifo_done;
        logic                   op_read_rx_num, op_read_rx_num_done;
```

```
always_comb begin : op_assign
	op_write_tx_fifo	= simp_valid && simp_wr && (simp_addr == UART_DATA_ADDR);
	op_read_rx_fifo		= simp_valid && simp_rd && (simp_addr == UART_DATA_ADDR);
	op_read_rx_num		= simp_valid && simp_rd && (simp_addr == UART_RX_DATA_NUM_ADDR);
end

assign	simp_done =	op_write_tx_fifo_done ||
					op_read_rx_fifo_done ||
					op_read_rx_num_done;

// TX side logic
logic	[7:0]	fifo_in_tx, fifo_out_tx;
logic		fifo_wr_en_tx, fifo_rd_en_tx;
logic		fifo_empty_tx;


fifo # (
	.BUF_WIDTH		(FIFO_WIDTH_TX),
	.DATA_WIDTH		(8)
) uart_tx_fifo (
	.clk			(clk),
	.rst			(rst),
	.buf_in			(fifo_in_tx),
	.buf_out		(fifo_out_tx),
	.wr_en			(fifo_wr_en_tx),
	.rd_en			(fifo_rd_en_tx),
	.buf_empty		(fifo_empty_tx),
	.buf_full		(),
	.buf_almost_full	(),
	.fifo_counter		()
);


typedef enum logic [1:0] {
	IDLE_FIFO_TX,
	WRITE_FIFO_TX
} uart_tx_fifo_state_t;
uart_tx_fifo_state_t state_fifo_tx, nxt_state_fifo_tx;

always_ff @( posedge clk, posedge rst ) begin
	if (rst)
		state_fifo_tx <= IDLE_FIFO_TX;
	else
		state_fifo_tx <= nxt_state_fifo_tx;
end


always_comb begin : uart_tx_fifo_state_fsm
	nxt_state_fifo_tx	= IDLE_FIFO_TX;
	fifo_in_tx		= 8'b0;
	fifo_wr_en_tx		= 1'b0;
	op_write_tx_fifo_done	= 1'b0;

	unique case (state_fifo_tx)

		IDLE_FIFO_TX:	begin
			if (op_write_tx_fifo) begin
				nxt_state_fifo_tx	= WRITE_FIFO_TX;
			end else begin
				nxt_state_fifo_tx	= IDLE_FIFO_TX;
			end
		end

		WRITE_FIFO_TX:	begin
			fifo_in_tx		= (ENDIANESS == BIG_ENDIAN) ?
simp_data_in[7:0] : simp_data_in[31:24];
			fifo_wr_en_tx		= ENABLE;
			nxt_state_fifo_tx	= IDLE_FIFO_TX;
			op_write_tx_fifo_done	= DONE;
		end
```

```
                default:    begin
                                nxt_state_fifo_tx           = IDLE_FIFO_TX;
                                fifo_in_tx                  = 8'b0;
                                fifo_wr_en_tx               = 1'b0;
                                op_write_tx_fifo_done       = 1'b0;
                            end
        endcase

end

typedef enum logic [1:0] {
        IDLE_TX,
        WAIT_LOAD_TX,
        TRANSMIT_TX
} uart_tx_state_t;
uart_tx_state_t state_uart_tx, nxt_state_uart_tx;

always_ff @( posedge clk or posedge rst) begin
        if (rst)
                state_uart_tx <= IDLE_TX;
        else
                state_uart_tx <= nxt_state_uart_tx;
end

always_comb begin : uart_tx_state_fsm

        nxt_state_uart_tx       = IDLE_TX;
        tx_data                         = fifo_out_tx;
        fifo_rd_en_tx           = DISABLE;
        uart_send_data          = DISABLE;

        unique case (state_uart_tx)
                IDLE_TX:        begin
                        if (~fifo_empty_tx) begin
                                nxt_state_uart_tx       = WAIT_LOAD_TX;
                                fifo_rd_en_tx           = ENABLE;
                        end else begin
                                nxt_state_uart_tx       = IDLE_TX;
                        end
                end

                WAIT_LOAD_TX:   begin
                        nxt_state_uart_tx       = TRANSMIT_TX;
                        uart_send_data          = ENABLE;
                end

                TRANSMIT_TX:    begin
                        if (tx_done) begin
                                nxt_state_uart_tx       = IDLE_TX;
                        end else begin
                                nxt_state_uart_tx       = TRANSMIT_TX;
                        end
                end

                default:        begin
                        nxt_state_uart_tx       = IDLE_TX;
                        tx_data                 = fifo_out_tx;
                        fifo_rd_en_tx           = DISABLE;
                        uart_send_data          = DISABLE;
                end
        endcase
end


// RX side logic
logic   [7:0]   fifo_in_rx, fifo_out_rx;
logic           fifo_wr_en_rx, fifo_rd_en_rx;
logic   [FIFO_WIDTH_RX-1:0]     fifo_counter_rx;
logic           rx_done_ff0, rx_done_ff1;
```

```
always_ff @(posedge clk, posedge rst) begin : rx_done_pos_edge_detect
	if (rst) begin
		rx_done_ff0 <= 1'b0;
		rx_done_ff1 <= 1'b1;
	end else begin
		rx_done_ff0 <= rx_done_raw;
		rx_done_ff1 <= rx_done_ff0;
	end
end

assign rx_done = ~rx_done_ff1 && rx_done_ff0;		// posedge

fifo # (
	.BUF_WIDTH				(FIFO_WIDTH_RX),
	.DATA_WIDTH				(8)
) uart_rx_fifo (
	.clk					(clk),
	.rst					(rst),
	.buf_in					(fifo_in_rx),
	.buf_out		(fifo_out_rx),
	.wr_en					(fifo_wr_en_rx),
	.rd_en					(fifo_rd_en_rx),
	.buf_empty				(),
	.buf_full		(),
	.buf_almost_full	(),
	.fifo_counter		(fifo_counter_rx)
);


typedef enum logic [2:0] {
	IDLE_RX,
	LOAD_CHAR_RX,
	READ_NUM_RX,
	READ_NUM_RX_2  // delay one cycle
} uart_rx_state_t;
uart_rx_state_t state_uart_rx, nxt_state_uart_rx;

always_ff @( posedge clk or posedge rst) begin
	if (rst)
		state_uart_rx <= IDLE_RX;
	else
		state_uart_rx <= nxt_state_uart_rx;
end

always_comb begin : fifo_rx_ctrl
	fifo_in_rx		= rx_done ? rx_data : 8'b0;
	fifo_wr_en_rx		= rx_done;
end

data_t	rx_num_big, rx_num_small;
assign rx_num_big = fifo_counter_rx; // should pad upper bits to 0
assign rx_num_small = swap_endian(rx_num_big);

data_t	rx_data_big, rx_data_small;
assign rx_data_big = fifo_out_rx;// should pad upper bits to 0
assign rx_data_small = swap_endian(fifo_out_rx);


always_comb begin : uart_rx_fifo_state_fsm

	nxt_state_uart_rx		= IDLE_RX;
	fifo_rd_en_rx				= DISABLE;
	op_read_rx_fifo_done= DISABLE;
	op_read_rx_num_done			= DISABLE;
	simp_data_out				= NULL;

	unique case (state_uart_rx)
		IDLE_RX: begin
			if (op_read_rx_fifo) begin
```

```
                                        nxt_state_uart_rx       = LOAD_CHAR_RX;
                                        fifo_rd_en_rx                   = ENABLE;
                                end else if (op_read_rx_num) begin
                                        nxt_state_uart_rx       = READ_NUM_RX;
                                end else begin
                                        nxt_state_uart_rx       = IDLE_RX;
                                end
                        end

                        LOAD_CHAR_RX: begin
                                nxt_state_uart_rx               = IDLE_RX;
                                op_read_rx_fifo_done= DONE;
                                simp_data_out                           = (ENDIANESS == BIG_ENDIAN) ? rx_data_big :
rx_data_small;
                        end

                        READ_NUM_RX: begin
                                nxt_state_uart_rx       = READ_NUM_RX_2;
                        end

                        READ_NUM_RX_2: begin
                                nxt_state_uart_rx       = IDLE_RX;
                                op_read_rx_num_done             = DONE;
                                simp_data_out                   = (ENDIANESS == BIG_ENDIAN) ? rx_num_big : rx_num_small;
                        end

                        default: begin
                                nxt_state_uart_rx               = IDLE_RX;
                                fifo_rd_en_rx                           = DISABLE;
                                op_read_rx_fifo_done= DISABLE;
                                simp_data_out                           = NULL;
                        end

                endcase
        end


        uart # (
                .CLK_FREQ       (CLK_FREQ),
                .UART_BPS       (UART_BPS)
        ) my_uart (
                // input
                .clk                    (clk),
                .rst_n                  (~rst),
                .RX                             (uart_rx),
                .send_data (uart_send_data),
                .tx_data    (tx_data),

                // output
                .TX                             (uart_tx),
                .rx_done    (rx_done_raw),
                .tx_done    (tx_done),
                .rx_data    (rx_data)
        );


        // connect everything with same name
        axil2simp #(
                .ADDR_WIDTH     (ADDR_WIDTH)
        ) axil_to_simp_bridge
        (
                .*
        );


endmodule : uart_axil
```

```
module uart_monitor # (
	parameter  CLK_FREQ = 5e7,
	parameter  UART_BPS = 9600
) (
	input	logic				clk,
	input	logic				rst,
	input	logic				RX,
	output	logic [7:0]		char
);

	logic done;
	logic [7:0] data, data_latch;

	always_ff @(posedge clk or posedge rst) begin
		if (rst)
			data_latch <= 8'b0;
		else if (done)
			data_latch <= data;
		else
			data_latch <= data_latch;
	end

	assign char = data_latch;

	uart_rx # (
		.CLK_FREQ		(CLK_FREQ),
		.UART_BPS		(UART_BPS)
	) myRX (
		.clk			(clk),
		.rst_n			(~rst),
		.RX				(RX),
		.uart_done (done),
		.uart_data  (data)
	);

endmodule : uart_monitor
```

```
module uart_rx # (
	parameter  CLK_FREQ = 5e7,    // clk freq
	parameter  UART_BPS = 9600    // port rate
) (
	input	logic                           clk,
	input	logic                           rst_n,

	input	logic                           RX,
	output	logic                           uart_done,
	output	logic	[7:0]	uart_data
);


	logic[19:0] BPS_CNT;
	assign BPS_CNT = CLK_FREQ / UART_BPS;


	//reg define
	reg                             RX_d0;
	reg                             RX_d1;
	reg	[15:0]	clk_cnt;
	reg	[ 3:0]	rx_cnt;
	reg                             rx_flag;
	reg	[ 7:0]	rxdata;


	//wire define
	wire	start_flag;


	// double flop RX data
	always_ff @(posedge clk or negedge rst_n) begin
		if (!rst_n) begin
			RX_d0 <= 1'b0;
			RX_d1 <= 1'b0;
		end
		else begin
			RX_d0 <= RX;
			RX_d1 <= RX_d0;
		end
	end


	// RX falling edge as start
	assign  start_flag = RX_d1 & (~RX_d0);


	// start capture when RX falling edge
	always_ff @(posedge clk or negedge rst_n) begin
		if (!rst_n)
			rx_flag <= 1'b0;
		else begin
			if(start_flag)
				rx_flag <= 1'b1;
			// else if((rx_cnt == 4'd9)&&(clk_cnt == BPS_CNT/2))
			else if((rx_cnt == 4'd9)&&(clk_cnt == BPS_CNT - 9))
				rx_flag <= 1'b0; // stop rx when the counter reached its half
			else
				rx_flag <= rx_flag;
		end
	end


	always_ff @(posedge clk or negedge rst_n) begin
		if (!rst_n) begin
			clk_cnt	<= 16'd0;
			rx_cnt	<= 4'd0;
		end
		else if ( rx_flag ) begin
			if (clk_cnt < BPS_CNT - 1) begin
```

```
                                clk_cnt     <= clk_cnt + 1'b1;
                                rx_cnt      <= rx_cnt;
                        end
                        else begin
                                clk_cnt     <= 16'd0;
                                rx_cnt      <= rx_cnt + 1'b1;
                        end
                end else begin
                                clk_cnt     <= 16'd0;
                                rx_cnt      <= 4'd0;
                end
        end


        always_ff @(posedge clk or negedge rst_n) begin
                if (!rst_n) begin
                        rxdata <= 8'd0;
                end else if (rx_flag) begin
                        if (clk_cnt == BPS_CNT / 2) begin               // counter reached middle point
                                unique case ( rx_cnt )
                                        4'd1 : rxdata[0] <= RX_d1;          // LSB
                                        4'd2 : rxdata[1] <= RX_d1;
                                        4'd3 : rxdata[2] <= RX_d1;
                                        4'd4 : rxdata[3] <= RX_d1;
                                        4'd5 : rxdata[4] <= RX_d1;
                                        4'd6 : rxdata[5] <= RX_d1;
                                        4'd7 : rxdata[6] <= RX_d1;
                                        4'd8 : rxdata[7] <= RX_d1;          // MSB
                                        default:;
                                endcase
                        end
                        else begin
                                rxdata <= rxdata;
                        end
                end else begin
                        rxdata <= 8'd0;
                end
        end


        always_ff @(posedge clk or negedge rst_n) begin
                if (!rst_n) begin
                        uart_data <= 8'd0;
                        uart_done <= 1'b0;
                end
                else if(rx_cnt == 4'd9) begin
                        uart_data <= rxdata;
                        uart_done <= 1'b1;
                end
                else begin
                        uart_data <= 8'd0;
                        uart_done <= 1'b0;
                end
        end

endmodule : uart_rx
```

```
module uart_tx # (
	parameter  CLK_FREQ = 5e7,    // clk freq
	parameter  UART_BPS = 9600    // port rate
) (
	input		logic                           clk,
	input		logic                           rst_n,

	input		logic                           uart_en,
	input		logic		[7:0]		uart_din,
	output		logic                           TX,
	output		logic                           tx_done
);


	logic[19:0] BPS_CNT;
	assign BPS_CNT = CLK_FREQ / UART_BPS;


	//reg define
	reg                             uart_en_d0;
	reg                             uart_en_d1;
	reg		[15:0]		clk_cnt;
	reg		[ 3:0]		tx_cnt;
	reg                             tx_flag;
	reg		[ 7:0]		tx_data;


	//wire define
	wire                    en_flag;


	// rising edge of enable
	assign en_flag = (~uart_en_d1) & uart_en_d0;


	//negedge for tx_flag means tx done
	reg dff1, dff2;
	always_ff @ (posedge clk, negedge rst_n) begin
		if (!rst_n) begin
			dff1		<= 1'b0;
			dff2		<= 1'b0;
		end else begin
			dff1		<= tx_flag;
			dff2		<= dff1;
		end
	end
	assign tx_done = (!dff1) & (dff2);


	// double flop uart_en signal
	always_ff @(posedge clk or negedge rst_n) begin
		if (!rst_n) begin
			uart_en_d0<= 1'b0;
			uart_en_d1<= 1'b0;
		end
		else begin
			uart_en_d0<= uart_en;
			uart_en_d1<= uart_en_d0;
		end
	end


	// start tx
	always_ff @(posedge clk or negedge rst_n) begin
		if (!rst_n) begin
			tx_flag		<= 1'b0;
			tx_data		<= 8'd0;
		end else if (en_flag) begin
					tx_flag		<= 1'b1;
					tx_data		<= uart_din;
```

```
                // bug:     this stop flag should work, but original code use BPS_CNT / 2
                end else if ((tx_cnt == 4'd10)&&(clk_cnt == BPS_CNT - 9)) begin
                                tx_flag     <= 1'b0;
                                tx_data     <= 8'd0;
                end else begin
                                tx_flag     <= tx_flag;
                                tx_data     <= tx_data;
                end
        end


        always_ff @(posedge clk or negedge rst_n) begin
                if (!rst_n) begin
                        clk_cnt     <= 16'd0;
                        tx_cnt      <= 4'd0;
                end
                else if (tx_flag) begin
                        if (clk_cnt < BPS_CNT - 1) begin
                                clk_cnt     <= clk_cnt + 1'b1;
                                tx_cnt      <= tx_cnt;
                        end
                        else begin
                                clk_cnt     <= 16'd0;
                                tx_cnt      <= tx_cnt + 1'b1;
                        end
                end
                else begin
                        clk_cnt     <= 16'd0;
                        tx_cnt      <= 4'd0;
                end
        end


        always_ff @(posedge clk or negedge rst_n) begin
                if (!rst_n) begin
                        TX <= 1'b1;
                end else if (tx_flag) begin
                        case(tx_cnt)
                                4'd0:       TX <= 1'b0;                     // start bit
                                4'd1:       TX <= tx_data[0];       // LSB
                                4'd2:       TX <= tx_data[1];
                                4'd3:       TX <= tx_data[2];
                                4'd4:       TX <= tx_data[3];
                                4'd5:       TX <= tx_data[4];
                                4'd6:       TX <= tx_data[5];
                                4'd7:       TX <= tx_data[6];
                                4'd8:       TX <= tx_data[7];       // MSB
                                4'd9:       TX <= 1'b1;                     // stop bit
                                4'd10:      TX <= 1'b1;                     // extra stop bit for resting
                                default:;
                        endcase
                end else begin
                        TX <= 1'b1;             // HIGH when IDLE
                end
        end

endmodule :     uart_tx
```

```
module uart # (
    parameter  CLK_FREQ = 5e7,
    parameter  UART_BPS = 9600
) (
    input      logic                clk,
    input      logic                rst_n,
    input      logic                RX,
    input      logic                send_data,
    input      logic      [7:0]     tx_data,

    output     logic                TX,
    output     logic                rx_done,
    output     logic                tx_done,
    output     logic      [7:0]     rx_data
);

    uart_rx # (
        .CLK_FREQ       (CLK_FREQ),
        .UART_BPS       (UART_BPS)
    ) myRX (
        .clk            (clk),
        .rst_n          (rst_n),
        .RX                         (RX),
        .uart_done (rx_done),
        .uart_data  (rx_data)
    );

    uart_tx # (
        .CLK_FREQ       (CLK_FREQ),
        .UART_BPS       (UART_BPS)
    ) myTX (
        .clk            (clk),
        .rst_n          (rst_n),
        .uart_en    (send_data),
        .uart_din   (tx_data),
        .TX                         (TX),
        .tx_done    (tx_done)
    );

endmodule : uart
```

```
module hex7seg (
	input logic  [3:0] a,
	output logic [6:0] y
);

	logic [6:0] yh;		// active high version of y
	always_comb begin
		unique case (a)
			4'h0:		yh = 7'b0111111;
			4'h1:		yh = 7'b0000110;
			4'h2:		yh = 7'b1011011;
			4'h3:		yh = 7'b1001111;
			4'h4:		yh = 7'b1100110;
			4'h5:		yh = 7'b1101101;
			4'h6:		yh = 7'b1111101;
			4'h7:		yh = 7'b0000111;
			4'h8:		yh = 7'b1111111;
			4'h9:		yh = 7'b1101111;
			4'ha:		yh = 7'b1110111;
			4'hb:		yh = 7'b1111100;
			4'hc:		yh = 7'b0111001;
			4'hd:		yh = 7'b1011110;
			4'he:		yh = 7'b1111001;
			4'hf:		yh = 7'b1110001;
			default:	yh = 7'b0000000;
		endcase
		y = ~yh;
	end

endmodule : hex7seg
```

```
package pref_defines;
import defines::*;
import mem_defines::*;

`ifndef _pref_defines_
`define _pref_defines_


localparamH2F_BASE                          = 32'hFC00_0000;
localparamH2F_LW_BASE                       = 32'hFF20_0000;      // not used for now


////////////////////// 7 seg defines //////////////////////

localparamSEG_BASE                                  = 32'h0400_0000;

localparamSEG_ADDR_MASK                     = 32'h0000_001C;
localparamSEG_DATA_MASK                     = 32'h0000_000F;

localparamSEG_H0_OFF                                = 32'h0;
localparamSEG_H1_OFF                                = 32'h4;
localparamSEG_H2_OFF                                = 32'h8;
localparamSEG_H3_OFF                                = 32'hC;
localparamSEG_H4_OFF                                = 32'h10;
localparamSEG_H5_OFF                                = 32'h14;

localparamSEG_H0_ADDR                               = SEG_H0_OFF;
localparamSEG_H1_ADDR                               = SEG_H1_OFF;
localparamSEG_H2_ADDR                               = SEG_H2_OFF;
localparamSEG_H3_ADDR                               = SEG_H3_OFF;
localparamSEG_H4_ADDR                               = SEG_H4_OFF;
localparamSEG_H5_ADDR                               = SEG_H5_OFF;

//////////////////////////////////////////////////////


////////////////////// UART defines //////////////////////
localparamUART_BPS                                  = 115200;
localparamUART_BASE                                 = 32'h0401_0000;

// write to here will write one byte of data to transmit fifo
// read from here will load one byte of data from receive fifo
localparamUART_DATA_OFF                     = 32'h0;

// read from here will tell number of bytes available in receive fifo
localparamUART_RX_DATA_NUM_OFF  = 32'h4;

localparamUART_DATA_ADDR                            = UART_DATA_OFF;
localparamUART_RX_DATA_NUM_ADDR             = UART_RX_DATA_NUM_OFF;

//////////////////////////////////////////////////////


`endif

endpackage : pref_defines
```

```
module seg_axil_wrapper (
        input   logic                           clk,
        input   logic                           rst,

        axil_interface.axil_slave       s00
);

        seg_axil segs (
                .clk        (clk),
                .rst        (rst),

                // AXI inputs
                .cfg_awvalid_i      (s00.axil_awvalid),
                .cfg_awaddr_i       (s00.axil_awaddr),
                .cfg_wvalid_i       (s00.axil_wvalid),
                .cfg_wdata_i        (s00.axil_wdata),
                .cfg_wstrb_i        (s00.axil_wstrb),
                .cfg_bready_i       (s00.axil_bready),
                .cfg_arvalid_i      (s00.axil_arvalid),
                .cfg_araddr_i       (s00.axil_araddr),
                .cfg_rready_i       (s00.axil_rready),

                // Outputs
                .cfg_awready_o      (s00.axil_awready),
                .cfg_wready_o       (s00.axil_wready),
                .cfg_bvalid_o       (s00.axil_bvalid),
                .cfg_bresp_o        (s00.axil_bresp),
                .cfg_arready_o      (s00.axil_arready),
                .cfg_rvalid_o       (s00.axil_rvalid),
                .cfg_rdata_o        (s00.axil_rdata),
                .cfg_rresp_o        (s00.axil_rresp),

                // unused AXI signal
                .cfg_awprot_i       (s00.axil_awprot),
                .cfg_arprot_i       (s00.axil_arprot),

                .hex0                       (),
                .hex1                       (),
                .hex2                       (),
                .hex3                       (),
                .hex4                       (),
                .hex5                       ()
        );

endmodule : seg_axil_wrapper
```

```
import defines::*;
import pref_defines::*;
import axi_defines::*;

module seg_axil (
	// Inputs
	input	logic				clk,
	input	logic				rst,

	// AXI inputs
	input	logic				cfg_awvalid_i,
	input	logic	[4:0]		cfg_awaddr_i,
	input	logic				cfg_wvalid_i,
	input	logic	[31:0]		cfg_wdata_i,
	input	logic	[ 3:0]		cfg_wstrb_i,
	input	logic				cfg_bready_i,
	input	logic				cfg_arvalid_i,
	input	logic	[4:0]		cfg_araddr_i,
	input	logic				cfg_rready_i,

	// Outputs
	output	logic				cfg_awready_o,
	output	logic				cfg_wready_o,
	output	logic				cfg_bvalid_o,
	output	logic	[1:0]		cfg_bresp_o,
	output	logic				cfg_arready_o,
	output	logic				cfg_rvalid_o,
	output	logic	[31:0]		cfg_rdata_o,
	output	logic	[ 1:0]		cfg_rresp_o,

	// unused AXI signal
	input	logic	[ 2:0]		cfg_awprot_i,
	input	logic	[ 2:0]		cfg_arprot_i,

	// 7-Seg output
	output	logic	[6:0]		hex0,
	output	logic	[6:0]		hex1,
	output	logic	[6:0]		hex2,
	output	logic	[6:0]		hex3,
	output	logic	[6:0]		hex4,
	output	logic	[6:0]		hex5
);

	typedef enum logic[2:0] {
		IDLE,
		W_ADDR,
		W_DATA,
		W_RESP,
		R_ADDR,
		R_RESP
	} state_t;

	state_t state, nxt_state;

	always_ff @( posedge clk, posedge rst ) begin
		if (rst)
			state <= IDLE;
		else
			state <= nxt_state;
	end

	// (* RAM_STYLE="logic" *)
	logic	[3:0]		seg_mem [0:5];

	logic	[4:0]		seg_address, wr_addr_latch, rd_addr_latch;
	assign	seg_address = cfg_awaddr_i[4:0] & SEG_ADDR_MASK;

	logic	[3:0]		seg_data, data_latch;
	assign	seg_data = (ENDIANESS == BIG_ENDIAN) ? cfg_wdata_i[3:0] : cfg_wdata_i[27:24];
```

```
// five channels' handshake
logic read_addr_handshake, read_data_handshake;
logic write_addr_handshake, write_data_handshake, write_resp_handshake;

always_comb begin
        read_addr_handshake = cfg_aready_o && cfg_arvalid_i;
        read_data_handshake = cfg_rready_i && cfg_rvalid_o;
        write_addr_handshake = cfg_awready_o && cfg_awvalid_i;
        write_data_handshake = cfg_wready_o && cfg_wvalid_i;
        write_resp_handshake = cfg_bready_i && cfg_bvalid_o;
end

always_comb begin : fsm
        nxt_state       = IDLE;
        cfg_awready_o   = 1'b0;
        cfg_wready_o    = 1'b0;
        cfg_bvalid_o    = INVALID;
        cfg_bresp_o             = RESP_OKAY;
        cfg_arready_o   = 1'b0;
        cfg_rvalid_o    = INVALID;
        cfg_rdata_o             = NULL;
        cfg_rresp_o             = RESP_OKAY;

        unique case (state)

                IDLE:   begin
                        if (cfg_arvalid_i) begin
                                cfg_arready_o = 1'b1;
                                nxt_state = R_RESP;
                        end else if (cfg_awvalid_i) begin
                                cfg_awready_o = 1'b1;
                                if (cfg_wvalid_i) begin
                                        cfg_wready_o = 1'b1;
                                        nxt_state = W_RESP;
                                end else begin
                                        nxt_state = W_DATA;
                                end
                        end else begin
                                nxt_state   = IDLE;
                        end
                end

                W_DATA:begin
                        if (cfg_wvalid_i) begin
                                cfg_wready_o = 1'b1;
                                nxt_state = W_RESP;
                        end begin
                                nxt_state = W_DATA;
                        end
                end

                W_RESP: begin
                        cfg_bvalid_o = VALID;
                        if (write_resp_handshake) begin
                                nxt_state = IDLE;
                        end else begin
                                nxt_state = W_RESP;
                        end
                end

                R_RESP:  begin
                        cfg_rvalid_o = VALID;
                        unique case (rd_addr_latch & SEG_ADDR_MASK)
                                SEG_H0_OFF:         begin
                                        cfg_rdata_o = {{28'b0}, {seg_mem[0]}};
                                end

                                SEG_H1_OFF:         begin
                                        cfg_rdata_o = {{28'b0}, {seg_mem[1]}};
                                end
```

```
                                        SEG_H2_OFF:     begin
                                                cfg_rdata_o = {{28'b0}, {seg_mem[2]}};
                                        end

                                        SEG_H3_OFF:     begin
                                                cfg_rdata_o = {{28'b0}, {seg_mem[3]}};
                                        end

                                        SEG_H4_OFF:     begin
                                                cfg_rdata_o = {{28'b0}, {seg_mem[4]}};
                                        end

                                        SEG_H5_OFF:     begin
                                                cfg_rdata_o = {{28'b0}, {seg_mem[5]}};
                                        end

                                        default:    begin
                                                cfg_rdata_o = NULL;
                                        end

                                endcase

                                if (read_data_handshake) begin
                                        nxt_state = IDLE;
                                end else begin
                                        nxt_state = R_RESP;
                                end
                        end

                        default:begin
                                nxt_state               = IDLE;
                                cfg_awready_o           = 1'b0;
                                cfg_wready_o            = 1'b0;
                                cfg_bvalid_o            = INVALID;
                                cfg_bresp_o                     = RESP_OKAY;
                                cfg_arready_o           = 1'b0;
                                cfg_rvalid_o            = INVALID;
                                cfg_rdata_o                     = NULL;
                                cfg_rresp_o                     = RESP_OKAY;
                        end

        endcase
end


always_ff @( posedge clk ) begin
        if (write_addr_handshake)
                wr_addr_latch <= seg_address;
        else
                wr_addr_latch <= wr_addr_latch;

        if (write_data_handshake)
                data_latch <= seg_data;
        else
                data_latch <= data_latch;

        if (read_addr_handshake)
                rd_addr_latch <= cfg_araddr_i[4:0];
        else
                rd_addr_latch <= rd_addr_latch;
end

integer i;

always_ff @( posedge clk, posedge rst) begin
        if (rst) begin
                for (i = 0; i < 6; i++) begin
                        seg_mem[i] <= 4'b0;
                end
```

```
                end else begin
                        if (write_resp_handshake) begin
                                for (i = 0; i < 6*4; i+=4) begin
                                        if (i == wr_addr_latch) begin
                                                seg_mem[i>>2] <= data_latch;
                                        end else begin
                                                seg_mem[i>>2] <= seg_mem[i>>2];
                                        end
                                end
                        end else begin
                                for (i = 0; i < 6; i++) begin
                                        seg_mem[i] <= seg_mem[i];
                                end
                        end
                end
        end

        hex7seg h0 (
                .a      (seg_mem[0]),
                .y      (hex0)
        );

        hex7seg h1 (
                .a      (seg_mem[1]),
                .y      (hex1)
        );

        hex7seg h2 (
                .a      (seg_mem[2]),
                .y      (hex2)
        );

        hex7seg h3 (
                .a      (seg_mem[3]),
                .y      (hex3)
        );

        hex7seg h4 (
                .a      (seg_mem[4]),
                .y      (hex4)
        );

        hex7seg h5 (
                .a      (seg_mem[5]),
                .y      (hex5)
        );

endmodule : seg_axil
```

```
import defines::*;

module ex_mem_reg (
        // common
        input clk,
        input rst_n,
        input flush,
        input en,

        // input
        input instr_t       instr_in,
        input data_t        alu_result_in,
        input data_t        rs2_in,
        input data_t        pc_in,
        input logic         rd_wren_in,
        input logic         instr_valid_in,

        // output
        output instr_t      instr_out,
        output data_t       alu_result_out,
        output data_t       rs2_out,
        output data_t       pc_out,
        output logic        rd_wren_out,
        output logic        instr_valid_out
);

        dffe_wrap #(.WIDTH(XLEN), .GEN_TARGET(TARGET)) instr_reg (
                .clk    (clk),
                .en             (en),
                .rst_n  (rst_n),
                .d              (instr_in),
                .q              (instr_out)
        );

        dffe_wrap #(.WIDTH(XLEN), .GEN_TARGET(TARGET)) alu_result_reg (
                .clk    (clk),
                .en             (en),
                .rst_n  (rst_n),
                .d              (alu_result_in),
                .q              (alu_result_out)
        );

        dffe_wrap #(.WIDTH(XLEN), .GEN_TARGET(TARGET)) rs2_reg (
                .clk    (clk),
                .en             (en),
                .rst_n  (rst_n),
                .d              (rs2_in),
                .q              (rs2_out)
        );

        dffe_wrap #(.WIDTH(XLEN), .GEN_TARGET(TARGET)) pc_reg (
                .clk    (clk),
                .en             (en),
                .rst_n  (rst_n),
                .d              (pc_in),
                .q              (pc_out)
        );

        dffe_wrap #(.WIDTH(1), .GEN_TARGET(TARGET)) rd_wren_reg (
                .clk    (clk),
                .en             (en),
                .rst_n  (rst_n),
                .d              (rd_wren_in),
                .q              (rd_wren_out)
        );

        dffe_wrap #(.WIDTH(1), .GEN_TARGET(TARGET)) instr_valid_reg (
                .clk    (clk),
                .en             (en),
                .rst_n  (rst_n),
```

```
                .d              (instr_valid_in),
                .q              (instr_valid_out)
        );

endmodule
```

```
import defines::*;

module id_ex_reg (
        // common
        input clk,
        input rst_n,
        input flush,
        input en,

        // input
        input instr_t       instr_in,
        input data_t        rs1_in,
        input data_t        rs2_in,
        input data_t        imm_in,
        input data_t        pc_in,
        input logic         branch_taken_in,
        input logic         instr_valid_in,

        // output
        output instr_t      instr_out,
        output data_t       rs1_out,
        output data_t       rs2_out,
        output data_t       imm_out,
        output data_t       pc_out,
        output logic        branch_taken_out,
        output logic        instr_valid_out
);

        dffe_wrap #(.WIDTH(XLEN), .GEN_TARGET(TARGET)) instr_reg (
                .clk    (clk),
                .en             (en),
                .rst_n  (rst_n),
                .d              (flush ? NOP : instr_in),
                .q              (instr_out)
        );

        dffe_wrap #(.WIDTH(XLEN), .GEN_TARGET(TARGET)) rs1_reg (
                .clk    (clk),
                .en             (en),
                .rst_n  (rst_n),
                .d              (flush ? 0 : rs1_in),
                .q              (rs1_out)
        );

        dffe_wrap #(.WIDTH(XLEN), .GEN_TARGET(TARGET)) pc_reg (
                .clk    (clk),
                .en             (en),
                .rst_n  (rst_n),
                .d              (flush ? 0 : pc_in),
                .q              (pc_out)
        );

        dffe_wrap #(.WIDTH(XLEN), .GEN_TARGET(TARGET)) rs2_reg (
                .clk    (clk),
                .en             (en),
                .rst_n  (rst_n),
                .d              (flush ? 0 : rs2_in),
                .q              (rs2_out)
        );

        dffe_wrap #(.WIDTH(XLEN), .GEN_TARGET(TARGET)) imm_reg (
                .clk    (clk),
                .en             (en),
                .rst_n  (rst_n),
                .d              (flush ? 0 : imm_in),
                .q              (imm_out)
        );

        dffe_wrap #(.WIDTH(1), .GEN_TARGET(TARGET)) branch_taken_reg (
                .clk    (clk),
```

```
                .en                     (en),
                .rst_n      (rst_n),
                .d                      (flush ? 1'b0 : branch_taken_in),
                .q                      (branch_taken_out)
        );

        dffe_wrap #(.WIDTH(1), .GEN_TARGET(TARGET)) instr_valid_reg (
                .clk        (clk),
                .en                     (en),
                .rst_n      (rst_n),
                .d                      (instr_valid_in),
                .q                      (instr_valid_out)
        );

endmodule
```

```
import defines::*;

module if_id_reg (
        // common
        input clk,
        input rst_n,
        input flush,
        input en,

        // input
        input data_t        pc_in,
        input instr_t       instr_in,
        input logic         instr_valid_in,

        // output
        output data_t       pc_out,
        output instr_t      instr_out,
        output logic        instr_valid_out
);

        dffe_wrap #(.WIDTH(XLEN), .GEN_TARGET(TARGET)) pc_reg (
                .clk    (clk),
                .en             (en),
                .rst_n  (rst_n),
                .d              (flush ? 0 : pc_in),
                .q              (pc_out)
        );

        dffe_wrap #(.WIDTH(XLEN), .GEN_TARGET(TARGET)) instr_reg (
                .clk    (clk),
                .en             (en),
                .rst_n  (rst_n),
                .d              (flush ? NOP : instr_in),
                .q              (instr_out)
        );

        dffe_wrap #(.WIDTH(1), .GEN_TARGET(TARGET)) instr_valid_reg (
                .clk    (clk),
                .en             (en),
                .rst_n  (rst_n),
                .d              (flush ? INVALID : instr_valid_in),
                .q              (instr_valid_out)
        );

endmodule
```

```
import defines::*;

module mem_wb_reg (
        // common
        input clk,
        input rst_n,
        input flush,
        input en,

        // input
        input instr_t       instr_in,
        input data_t        alu_result_in,
        input data_t        mem_data_in_in,
        input data_t        mem_data_out_in,
        input data_t        pc_in,
        input logic         rd_wren_in,
        input logic         instr_valid_in,
        input data_t        mem_addr_in,

        // output
        output instr_t      instr_out,
        output data_t       alu_result_out,
        output data_t       mem_data_in_out,
        output data_t       mem_data_out_out,
        output data_t       pc_out,
        output logic        rd_wren_out,
        output logic        instr_valid_out,
        output data_t       mem_addr_out
);

        dffe_wrap #(.WIDTH(XLEN), .GEN_TARGET(TARGET)) instr_reg (
                .clk    (clk),
                .en             (en),
                .rst_n  (rst_n),
                .d              (instr_in),
                .q              (instr_out)
        );

        dffe_wrap #(.WIDTH(XLEN), .GEN_TARGET(TARGET)) alu_result_reg (
                .clk    (clk),
                .en             (en),
                .rst_n  (rst_n),
                .d              (alu_result_in),
                .q              (alu_result_out)
        );

        dffe_wrap #(.WIDTH(XLEN), .GEN_TARGET(TARGET)) mem_data_out_reg (
                .clk    (clk),
                .en             (en),
                .rst_n  (rst_n),
                .d              (mem_data_out_in),
                .q              (mem_data_out_out)
        );

        dffe_wrap #(.WIDTH(XLEN), .GEN_TARGET(TARGET)) mem_data_in_reg (
                .clk    (clk),
                .en             (en),
                .rst_n  (rst_n),
                .d              (mem_data_in_in),
                .q              (mem_data_in_out)
        );

        dffe_wrap #(.WIDTH(XLEN), .GEN_TARGET(TARGET)) pc_reg (
                .clk    (clk),
                .en             (en),
                .rst_n  (rst_n),
                .d              (pc_in),
                .q              (pc_out)
        );
```

```
    dffe_wrap #(.WIDTH(1), .GEN_TARGET(TARGET)) rd_wren_reg (
        .clk    (clk),
        .en             (en),
        .rst_n  (rst_n),
        .d              (rd_wren_in),
        .q              (rd_wren_out)
    );

    dffe_wrap #(.WIDTH(1), .GEN_TARGET(TARGET)) instr_valid_reg (
        .clk    (clk),
        .en             (en),
        .rst_n  (rst_n),
        .d              (instr_valid_in),
        .q              (instr_valid_out)
    );

    dffe_wrap #(.WIDTH(XLEN), .GEN_TARGET(TARGET)) mem_addr_reg (
        .clk    (clk),
        .en             (en),
        .rst_n  (rst_n),
        .d              (mem_addr_in),
        .q              (mem_addr_out)
    );

endmodule
```

```
`timescale 1 ns / 1 ps

import defines::*;
import mem_defines::*;
import axi_defines::*;
import pref_defines::*;


module reference_test_axi ();

	localparam REG_DEBUG = DISABLE;
	localparam MEM_DEBUG = DISABLE;

	integer error;
	int fd;

	logic clk, rst_n, ebreak_start, go, rst;
	assign rst = ~rst_n;

	logic ref_halt, ref_halt_wait;
	logic kill_ref;
	logic [XLEN-1:0] boot_pc [0:0];

	clkrst #(.FREQ(FREQ)) clkrst_inst (
		.clk		(clk),
		.rst_n		(rst_n),
		.go			(go)
	);

	initial begin
		$readmemh("boot.cfg", boot_pc);
		$display("DUT: boot mode: binary");
		$display("DUT: booting from pc = %h", boot_pc[0] * 4 + 32'h10000);
	end


	axil_interface instr_bus();			// S00
	axil_interface data_bus();			// S01

	axil_interface ram_bus();			// M00
	axil_interface seg_bus();			// M01
	axil_interface uart_bus();			// M02

	// unused placeholder dummy
	axil_interface s02_bus();
	axil_interface s03_bus();

	axil_interface m03_bus();
	axil_interface m04_bus();
	axil_interface m05_bus();


	proc processor (
		.clk				(clk),
		.rst_n				(rst_n),
		.go					(go),
		.boot_pc		(boot_pc[0][9:0]),

		.data_bus		(data_bus),
		.instr_bus		(instr_bus)
	);

	axil_ram_sv_wrapper # (
		.ADDR_WIDTH		(19),
		.bootload		(ENABLE)
	) text (
		.clk				(clk),
		.rst				(rst),
		.axil_bus		(ram_bus)
	);
```

```
logic [3:0] s0, s1, s2, s3, s4, s5;
seg_axil_wrapper dummy_seg_display (
        .clk        (clk),
        .rst        (rst),
        .s00        (seg_bus)
);
always_comb begin : seg_peek
        s0 = dummy_seg_display.segs.seg_mem[0];
        s1 = dummy_seg_display.segs.seg_mem[1];
        s2 = dummy_seg_display.segs.seg_mem[2];
        s3 = dummy_seg_display.segs.seg_mem[3];
        s4 = dummy_seg_display.segs.seg_mem[4];
        s5 = dummy_seg_display.segs.seg_mem[5];
end

logic riscy_uart_rx;
logic riscy_uart_tx;
logic riscy_uart_cts;
logic riscy_uart_rts;
logic [7:0] uart_char;

uart_axil_wrapper # (
        .UART_BPS       (UART_BPS)
) dummy_uart_slave (
        .clk            (clk),
        .rst            (rst),
        .s00            (uart_bus),
        .uart_rx    (riscy_uart_rx),
        .uart_tx    (riscy_uart_tx),
        .uart_cts   (riscy_uart_cts),
        .uart_rts   (riscy_uart_rts)
);


uart_monitor # (
        .CLK_FREQ       (FREQ),
        .UART_BPS       (UART_BPS)
) my_uart_monitor (
        .clk            (clk),
        .rst            (rst),
        .RX                     (riscy_uart_tx),
        .char           (uart_char)
);


logic uart_driver_en;
logic uart_driver_done;
logic[7:0] uart_driver_data;
uart_tx # (
        .CLK_FREQ       (FREQ),
        .UART_BPS       (UART_BPS)
) my_uart_driver (
        .clk            (clk),
        .rst_n          (rst_n),

        .TX                     (riscy_uart_rx),
        .uart_en    (uart_driver_en),
        .uart_din   (uart_driver_data),
        .tx_done    (uart_driver_done)
);

assign riscy_uart_cts = 1'b0;

axil_dummy_master dummy_master_02 (s02_bus);
axil_dummy_master dummy_master_03 (s03_bus);


axil_dummy_slave dummy_slave_03 (m03_bus);
```

```
axil_dummy_slave dummy_slave_04 (m04_bus);
axil_dummy_slave dummy_slave_05 (m05_bus);

axil_interconnect_4x6_wrapper # (
        .ADDR_WIDTH                     (XLEN),

        .M00_BASE_ADDR          (32'h0),
        .M00_ADDR_WIDTH                 (32'd26),

        .M01_BASE_ADDR          (SEG_BASE),
        .M01_ADDR_WIDTH                 (32'd5),

        .M02_BASE_ADDR          (UART_BASE),
        .M02_ADDR_WIDTH                 (32'd5),

        .M03_BASE_ADDR          (32'h0800_0000),
        .M03_ADDR_WIDTH                 (32'd0),

        .M04_BASE_ADDR          (32'h8100_0000),
        .M04_ADDR_WIDTH                 (32'd0),

        .M05_BASE_ADDR          (32'h8200_0000),
        .M05_ADDR_WIDTH                 (32'd0)
) interconnect_4x6 (
        .clk                            (clk),
        .rst                            (rst),

        .s00                            (instr_bus),
        .s01                            (data_bus),
        .s02                            (s02_bus),
        .s03                            (s03_bus),

        .m00                            (ram_bus),
        .m01                            (seg_bus),
        .m02                            (uart_bus),
        .m03                            (m03_bus),
        .m04                            (m04_bus),
        .m05                            (m05_bus)
);


ref_hier proc_ref (
        .clk                            (clk),
        .rst                            (rst),
        .kill                           (kill_ref)
);

// UART initial input

task uart_write_char(input logic[7:0] c);
        @(negedge clk);
        uart_driver_en                  = ENABLE;
        uart_driver_data        = c;
        repeat(100) @(negedge clk);
        uart_driver_en                  = DISABLE;
        @(posedge uart_driver_done);
        uart_driver_data        = 8'b0;
        @(negedge uart_driver_done);
        @(negedge clk);
endtask

initial begin
        uart_driver_en                  = DISABLE;
        uart_driver_data        = 8'b0;
        @(posedge go);
        repeat(1000) @(negedge clk);
        uart_write_char(8'h46);         // char 'F'
        uart_write_char(8'h47);         // char 'F'
end
```

```
	initial begin
		ref_halt = 1'b0;
		wait(ref_halt_wait);
		ref_halt = 1'b1;
	end

	always_comb begin
		ref_halt_wait = (data_t'(proc_ref.mem_i_inst_w) == EBREAK);
		kill_ref = ref_halt;
	end

	// reg dut wire
	logic	reg_wr_en_dut;
	r_t	reg_wr_addr_dut;
	data_t	reg_wr_data_dut;
	always_comb begin : reg_dut_wire_assign
		reg_wr_en_dut	= processor.rd_wren_w;
		reg_wr_addr_dut	= processor.rd_addr;
		reg_wr_data_dut	= processor.wb_data;
	end

	// mem dut wire
	logic	mem_wr_en_dut, mem_rd_en_dut, mem_access_done_dut;
	data_t	mem_wr_data_in_dut, mem_rd_data_out_dut;
	data_t	mem_access_addr_dut;
	always_comb begin : mem_dut_wire_assign
		mem_wr_en_dut		= processor.memory_inst.wren;
		mem_rd_en_dut		= processor.memory_inst.rden;
		mem_access_done_dut	= processor.mem_access_done;
		mem_wr_data_in_dut = (ENDIANESS == BIG_ENDIAN) ? processor.memory_inst.data_in_final :
                                                                 swap_endian(processor.memory_inst.data_in_final);
		mem_rd_data_out_dut	= (ENDIANESS == BIG_ENDIAN) ?
processor.memory_inst.memory_system.m_axil_rdata :

	swap_endian(processor.memory_inst.memory_system.m_axil_rdata);
		mem_access_addr_dut	= processor.memory_inst.addr & word_align_mask;
	end


	// reg ref wire
	logic	reg_wr_en_ref;
	r_t	reg_wr_addr_ref;
	data_t	reg_wr_data_ref;
	always_comb begin : reg_reg_wire_assign
		reg_wr_en_ref		= proc_ref.core_ref.rd_writeen_w;
		reg_wr_addr_ref		= r_t'(proc_ref.core_ref.rd_q);
		reg_wr_data_ref		= data_t'(proc_ref.core_ref.rd_val_w);
	end

	assign ebreak_start		= processor.instr_w == EBREAK;

	// mem ref wire
	logic	mem_wr_en_ref, mem_rd_en_ref, mem_access_ack_ref;
	data_t	mem_wr_data_in_ref, mem_rd_data_out_ref;
	data_t	mem_access_addr_ref;
	always_comb begin : mem_ref_wire_assign
		mem_wr_en_ref		= (proc_ref.mem_d_wr_w != 4'b0);	// byte enable all 0s
		mem_rd_en_ref		= proc_ref.mem_d_rd_w;
		mem_access_ack_ref = proc_ref.mem_d_ack_w;
		mem_wr_data_in_ref = proc_ref.mem_d_data_wr_w;
		mem_rd_data_out_ref= proc_ref.mem_d_data_rd_w;
		mem_access_addr_ref= proc_ref.mem_d_addr_w;
	end

	typedef enum logic {
		READ, WRITE
	} rw_t;

	typedef struct packed {
		rw_t	rw;
```

```
		r_t			rw_addr;
		data_t		rw_data;
		integer		sim_time;
		data_t		pc;
		//instr_t	instr;
} reg_access_t;

typedef struct packed {
		rw_t		rw;
		data_t		rw_addr;
		data_t		rw_data;
		integer		sim_time;
		data_t		pc;
		//instr_t	instr;
} mem_access_t;

typedef struct packed {
		integer		sim_time;
		data_t		pc;
		//instr_t	instr;
} pc_log_t;

reg_access_t reg_access_log_ref[$] = {};
reg_access_t reg_access_log_dut[$] = {};
mem_access_t mem_access_log_ref[$] = {};
mem_access_t mem_access_log_dut[$] = {};
pc_log_t pc_log_ref[$] = {};
pc_log_t pc_log_dut[$] = {};

function void push_pc_ref();
		if (pc_log_ref.size() == 0) begin
			pc_log_ref.push_back(
				pc_log_t'(
					{
						sim_time:  $time,
						pc:				proc_ref.core_ref.pc_q
					}
				)
			);
		end else if (pc_log_ref[pc_log_ref.size()-1].pc == (proc_ref.core_ref.pc_q)) begin
			// do nothing, duplicative entry
		end else begin
			pc_log_ref.push_back(
				pc_log_t'(
					{
						sim_time:  $time,
						pc:				proc_ref.core_ref.pc_q
					}
				)
			);
		end
endfunction

function void push_pc_dut();
		if (pc_log_dut.size() == 0) begin
			pc_log_dut.push_back(
				pc_log_t'(
					{
						sim_time:  $time,
						pc:				processor.pc_w
					}
				)
			);
		end else if (pc_log_dut[pc_log_dut.size()-1].pc == (processor.pc_w)) begin
			// do nothing, duplicative entry
		end else begin
			pc_log_dut.push_back(
				pc_log_t'(
					{
						sim_time:  $time,
```

```
                                                pc:                                     processor.pc_w
                                        }
                                )
                        );
        end
endfunction

function void compare_pc_log();
        for (integer i = 0; i < ( (pc_log_ref.size() > pc_log_dut.size()) ? pc_log_dut.size() : pc_log_ref.size() ); i++ ) begin
                if (pc_log_ref[i].pc != pc_log_dut[i].pc) begin
                        // $display("PC mismatch found at the #%d instr", i + 1);
                        return;
                end
        end
        $display("Good: PC flow match");
endfunction

always_ff @(posedge clk) begin
        if (~$isunknown(proc_ref.core_ref.pc_q))
                push_pc_ref();
        if (processor.instr_valid_w)
                push_pc_dut();
end

function void push_reg_ref();
        if (reg_access_log_ref.size() == 0) begin
                reg_access_log_ref.push_back(
                        reg_access_t'({
                                rw:                                     WRITE,
                                rw_addr:   reg_wr_addr_ref,
                                rw_data:   reg_wr_data_ref,
                                sim_time:  $time,
                                pc:                                     (proc_ref.core_ref.pc_q - 32'd4)
                                //instr:                instr_t'(proc_ref.core_ref.mem_i_inst_i)
                        })

                );
                if (REG_DEBUG) begin
                        $display(
                                "debug: REF REG WRITE %h, to X%d at time=%t with pc=%h",
                                reg_wr_data_ref, $unsigned(reg_wr_addr_ref), $time, proc_ref.core_ref.pc_q - 32'd4
                        );
                end
        end else if ( reg_access_log_ref[reg_access_log_ref.size()-1].pc == (proc_ref.core_ref.pc_q - 32'd4)) begin
                // do nothing, duplicative entry
        end else begin
                reg_access_log_ref.push_back(
                        reg_access_t'({
                                rw:                                     WRITE,
                                rw_addr:   reg_wr_addr_ref,
                                rw_data:   reg_wr_data_ref,
                                sim_time:  $time,
                                pc:                                     (proc_ref.core_ref.pc_q - 4)
                                //instr:                instr_t'(proc_ref.core_ref.mem_i_inst_i)
                        })
                );
                if (REG_DEBUG) begin
                        $display(
                                "debug: REF REG WRITE %h, to X%d at time=%t with pc=%h",
                                reg_wr_data_ref, $unsigned(reg_wr_addr_ref), $time, proc_ref.core_ref.pc_q - 32'd4
                        );
                end
        end
endfunction


function void push_reg_dut();
        if (reg_access_log_dut.size() == 0) begin
                reg_access_log_dut.push_back(
                        reg_access_t'({
```

```
                                        rw:                             WRITE,
                                        rw_addr:    reg_wr_addr_dut,
                                        rw_data:    reg_wr_data_dut,
                                        sim_time:   $time,
                                        pc:                             (processor.pc_w)
                                }
                        );
                        if (REG_DEBUG) begin
                                $display(
                                        "debug: DUT REG WRITE %h, to X%d at time=%t with pc=%h",
                                        reg_wr_data_dut, $unsigned(reg_wr_addr_dut), $time, (processor.pc_w)
                                );
                        end
                end else if (reg_access_log_dut[reg_access_log_dut.size()-1].pc == (processor.pc_w)) begin
                        // do nothing, duplicative entry
                end else begin
                        reg_access_log_dut.push_back(
                                reg_access_t'({
                                        rw:                             WRITE,
                                        rw_addr:    reg_wr_addr_dut,
                                        rw_data:    reg_wr_data_dut,
                                        sim_time:   $time,
                                        pc:                             processor.pc_w
                                        //instr:                processor.instr_w
                                })
                        );
                        if (REG_DEBUG) begin
                                $display(
                                        "debug: DUT REG WRITE %h, to X%d at time=%t with pc=%h",
                                        reg_wr_data_dut, $unsigned(reg_wr_addr_dut), $time, (processor.pc_w)
                                );
                        end
                end
        endfunction


        task push_mem_ref_helper();
                mem_access_log_ref.push_back(
                        mem_access_t'({
                                rw:                             mem_wr_en_ref ? WRITE : READ,
                                rw_addr:    mem_access_addr_ref,
                                rw_data:    mem_wr_en_ref ? mem_wr_data_in_ref : mem_rd_data_out_ref,
                                sim_time:   $time,
                                pc:                             (proc_ref.core_ref.pc_q - 32'd4)
                                //instr:                instr_t'(proc_ref.mem_i_inst_w)
                        })
                );
                if (MEM_DEBUG) begin
                        if (mem_wr_en_ref) begin
                                $display(
                                        "debug: REF MEM WRITE %h, to   %h at time=%t with pc=%h",
                                        mem_wr_data_in_ref, mem_access_addr_ref, $time, proc_ref.core_ref.pc_q -
32'd4
                                );
                        end else begin // don't add " else if (mem_rd_en_ref)" here cuz rden is not aserted when ack is high
                                $display(
                                        "debug: REF MEM READ  %h, from %h at time=%t with pc=%h",
                                        mem_rd_data_out_ref, mem_access_addr_ref, $time, proc_ref.core_ref.pc_q -
32'd4
                                );
                        end
                end
        endtask


        task push_mem_ref();
                if (mem_access_log_ref.size() == 0) begin
                        if (mem_rd_en_ref) begin
                                @(posedge mem_access_ack_ref);
                                push_mem_ref_helper();
```

```
                end else begin
                        push_mem_ref_helper();
                end

        end else if ( mem_access_log_ref[mem_access_log_ref.size()-1].pc == proc_ref.core_ref.pc_q - 32'd4) begin
                // do nothing, duplicative entry
        end else begin
                if (mem_rd_en_ref) begin
                        @(posedge mem_access_ack_ref);
                        push_mem_ref_helper();
                end else begin
                        push_mem_ref_helper();
                end
        end
endtask


function void push_mem_dut();
        if (mem_access_log_dut.size() == 0) begin
                mem_access_log_dut.push_back(
                        mem_access_t'({
                                rw:                             mem_wr_en_dut ? WRITE : READ,
                                rw_addr:    mem_access_addr_dut,
                                rw_data:    mem_wr_en_dut ? mem_wr_data_in_dut : mem_rd_data_out_dut,
                                sim_time:   $time,
                                pc:                             processor.pc_m
                                //instr:                instr_t'(processor.instr_m)
                        })
                );
                if (MEM_DEBUG) begin
                        if (mem_wr_en_dut) begin
                                $display(
                                        "debug: DUT MEM WRITE %h, to   %h at time=%t with pc=%h",
                                        mem_wr_data_in_dut, mem_access_addr_dut, $time, (processor.pc_m)
                                );
                        end else begin
                                $display(
                                        "debug: DUT MEM READ  %h, from %h at time=%t with pc=%h",
                                        mem_rd_data_out_dut, mem_access_addr_dut, $time, (processor.pc_m)
                                );
                        end
                end
        end else if ( mem_access_log_dut[mem_access_log_dut.size()-1].pc == (processor.pc_m)) begin
                // do nothing, duplicative entry
        end else begin
                mem_access_log_dut.push_back(
                        mem_access_t'({
                                rw:                             mem_wr_en_dut ? WRITE : READ,
                                rw_addr:    mem_access_addr_dut,
                                rw_data:    mem_wr_en_dut ? mem_wr_data_in_dut : mem_rd_data_out_dut,
                                sim_time:   $time,
                                pc:                             processor.pc_m
                                //instr:                instr_t'(processor.instr_m)
                        })
                );
                if (MEM_DEBUG) begin
                        if (mem_wr_en_dut) begin
                                $display(
                                        "debug: DUT MEM WRITE %h, to   %h at time=%t with pc=%h",
                                        mem_wr_data_in_dut, mem_access_addr_dut, $time, (processor.pc_m)
                                );
                        end else begin
                                $display(
                                        "debug: DUT MEM READ  %h, from %h at time=%t with pc=%h",
                                        mem_rd_data_out_dut, mem_access_addr_dut, $time, (processor.pc_m)
                                );
                        end
                end
        end
endfunction
```

```
always @(negedge clk) begin : ref_debug_log
	// reg log
	if (reg_wr_en_ref && reg_wr_addr_ref != X0) begin
		push_reg_ref();
	end

	// mem log
	// cant use done for ref cuz it use ack handshake instead of done
	if (mem_wr_en_ref || mem_rd_en_ref) begin
		push_mem_ref();
	end
end


always @(negedge clk) begin : dut_debug_log
	// reg log
	if (reg_wr_en_dut && reg_wr_addr_dut != X0) begin
		push_reg_dut();
	end

	// mem log
	if ((mem_wr_en_dut || mem_rd_en_dut) && mem_access_done_dut) begin
		push_mem_dut();
	end
end


task compare_reg_log();
	assert	(reg_access_log_ref[0].rw == reg_access_log_dut[0].rw)
	else begin
		error = 1;
	end

	assert	(reg_access_log_ref[0].rw_addr == reg_access_log_dut[0].rw_addr)
	else begin
		error = 1;
	end

	assert	(reg_access_log_ref[0].rw_data == reg_access_log_dut[0].rw_data)
	else begin
		error = 1;
	end

	reg_access_log_ref.pop_front();
	reg_access_log_dut.pop_front();
endtask


task compare_mem_log();
	assert	(mem_access_log_ref[0].rw == mem_access_log_dut[0].rw)
	else begin
		error = 1;
	end

	assert	(mem_access_log_ref[0].rw_addr == mem_access_log_dut[0].rw_addr)
	else begin
		error = 1;
	end

	assert	(mem_access_log_ref[0].rw_data == mem_access_log_dut[0].rw_data)
	else begin
		error = 1;
	end

	//$display("poped mem log at pc=%d", mem_access_log_ref[0].pc);
	mem_access_log_ref.pop_front();
	mem_access_log_dut.pop_front();
endtask
```

```
task write_csv ();
        integer wli, f;             // write log index, file number
        $display("sim finished, writing csv log file...");

        f = $fopen("reg_ref.csv","w");
        $fwrite(f, "\"field\", \"time\", \"pc\", \"rw\", \"data\", \"addr\" \n");
        for (wli = 0; wli < reg_access_log_ref.size(); wli++) begin
                $fwrite(f, "\"%s\", \"%t\", \"0x%h\", \"%s\", \"0x%h\", \"%d\" \n",
                "reg",
                reg_access_log_ref[wli].sim_time / 1000,
                reg_access_log_ref[wli].pc,
                reg_access_log_ref[wli].rw == READ ? "read" : "write",
                reg_access_log_ref[wli].rw_data,
                reg_access_log_ref[wli].rw_addr);
        end
        $fclose(f);

        f = $fopen("reg_dut.csv","w");
        $fwrite(f, "\"field\", \"time\", \"pc\", \"rw\", \"data\", \"addr\" \n");
        for (wli = 0; wli < reg_access_log_dut.size(); wli++) begin
                $fwrite(f, "\"%s\", \"%t\", \"0x%h\", \"%s\", \"0x%h\", \"%d\" \n",
                "reg",
                reg_access_log_dut[wli].sim_time / 1000,
                reg_access_log_dut[wli].pc,
                reg_access_log_dut[wli].rw == READ ? "read" : "write",
                reg_access_log_dut[wli].rw_data,
                reg_access_log_dut[wli].rw_addr);
        end
        $fclose(f);

        f = $fopen("mem_ref.csv","w");
        $fwrite(f, "\"field\", \"time\", \"pc\", \"rw\", \"data\", \"addr\" \n");
        for (wli = 0; wli < mem_access_log_ref.size(); wli++) begin
                $fwrite(f, "\"%s\", \"%t\", \"0x%h\", \"%s\", \"0x%h\", \"0x%h\" \n",
                "mem",
                mem_access_log_ref[wli].sim_time / 1000,
                mem_access_log_ref[wli].pc,
                mem_access_log_ref[wli].rw == READ ? "read" : "write",
                mem_access_log_ref[wli].rw_data,
                mem_access_log_ref[wli].rw_addr);
        end
        $fclose(f);

        f = $fopen("mem_dut.csv","w");
        $fwrite(f, "\"field\", \"time\", \"pc\", \"rw\", \"data\", \"addr\" \n");
        for (wli = 0; wli < mem_access_log_dut.size(); wli++) begin
                $fwrite(f, "\"%s\", \"%t\", \"0x%h\", \"%s\", \"0x%h\", \"0x%h\" \n",
                "mem",
                mem_access_log_dut[wli].sim_time / 1000,
                mem_access_log_dut[wli].pc,
                mem_access_log_dut[wli].rw == READ ? "read" : "write",
                mem_access_log_dut[wli].rw_data,
                mem_access_log_dut[wli].rw_addr);
        end
        $fclose(f);

        f = $fopen("pc_ref.csv","w");
        $fwrite(f, "\"field\", \"time\", \"pc\" \n");
        for (wli = 0; wli < pc_log_ref.size(); wli++) begin
                $fwrite(f, "\"%s\", \"%t\", \"%h\" \n",
                "pc",
                pc_log_ref[wli].sim_time / 1000,
                pc_log_ref[wli].pc);
        end
        $fclose(f);

        f = $fopen("pc_dut.csv","w");
        $fwrite(f, "\"field\", \"time\", \"pc\" \n");
        for (wli = 0; wli < pc_log_dut.size(); wli++) begin
```

```
                $fwrite(f, "\"%s\", \"%t\", \"%h\" \n",
                "pc",
                pc_log_dut[wli].sim_time / 1000,
                pc_log_dut[wli].pc);
        end
        $fclose(f);
endtask

task write_log();
        integer wli, f;         // write log index, file number
        $display("sim finished, writing log file...");
        f = $fopen("reg_ref.log","w");
        for (wli = 0; wli < reg_access_log_ref.size(); wli++) begin
                if (reg_access_log_ref[wli].rw == READ) begin
                        $fwrite(f, "sim time = %t, pc = %h, READ %h from X%d \n",
                        reg_access_log_ref[wli].sim_time,
                        reg_access_log_ref[wli].pc,
                        reg_access_log_ref[wli].rw_data,
                        reg_access_log_ref[wli].rw_addr);
                end else begin
                        $fwrite(f, "sim time = %t, pc = %h, WRITE %h to X%d \n",
                        reg_access_log_ref[wli].sim_time,
                        reg_access_log_ref[wli].pc,
                        reg_access_log_ref[wli].rw_data,
                        reg_access_log_ref[wli].rw_addr);
                end
        end
        $fclose(f);

        f = $fopen("reg_dut.log","w");
        for (wli = 0; wli < reg_access_log_dut.size(); wli++) begin
                if (reg_access_log_dut[wli].rw == READ) begin
                        $fwrite(f, "sim time = %t, pc = %h, READ %h from X%d \n",
                        reg_access_log_dut[wli].sim_time,
                        reg_access_log_dut[wli].pc,
                        reg_access_log_dut[wli].rw_data,
                        reg_access_log_dut[wli].rw_addr);
                end else begin
                        $fwrite(f, "sim time = %t, pc = %h, WRITE %h to X%d \n",
                        reg_access_log_dut[wli].sim_time,
                        reg_access_log_dut[wli].pc,
                        reg_access_log_dut[wli].rw_data,
                        reg_access_log_dut[wli].rw_addr);
                end
        end
        $fclose(f);

        f = $fopen("mem_ref.log","w");
        for (wli = 0; wli < mem_access_log_ref.size(); wli++) begin
                if (mem_access_log_ref[wli].rw == READ) begin
                        $fwrite(f, "sim time = %t, pc = %h, READ %h from mem[%h] \n",
                        mem_access_log_ref[wli].sim_time,
                        mem_access_log_ref[wli].pc,
                        mem_access_log_ref[wli].rw_data,
                        mem_access_log_ref[wli].rw_addr);
                end else begin
                        $fwrite(f, "sim time = %t, pc = %h, WRITE %h to mem[%h] \n",
                        mem_access_log_ref[wli].sim_time,
                        mem_access_log_ref[wli].pc,
                        mem_access_log_ref[wli].rw_data,
                        mem_access_log_ref[wli].rw_addr);
                end
        end
        $fclose(f);

        f = $fopen("mem_dut.log","w");
        for (wli = 0; wli < mem_access_log_dut.size(); wli++) begin
                if (mem_access_log_dut[wli].rw == READ) begin
                        $fwrite(f, "sim time = %t, pc = %h, READ %h from mem[%h] \n",
                        mem_access_log_dut[wli].sim_time,
```

```
                                        mem_access_log_dut[wli].pc,
                                        mem_access_log_dut[wli].rw_data,
                                        mem_access_log_dut[wli].rw_addr);
                        end else begin
                                        $fwrite(f, "sim time = %t, pc = %h, WRITE %h to mem[%h] \n",
                                        mem_access_log_dut[wli].sim_time,
                                        mem_access_log_dut[wli].pc,
                                        mem_access_log_dut[wli].rw_data,
                                        mem_access_log_dut[wli].rw_addr);
                        end
                end
                $fclose(f);

                f = $fopen("pc_ref.log","w");
                for (wli = 0; wli < pc_log_ref.size(); wli++) begin
                        $fwrite(f, "%h - %d @ t = %t\n", pc_log_ref[wli].pc, pc_log_ref[wli].pc, pc_log_ref[wli].sim_time);
                end
                $fclose(f);

                f = $fopen("pc_dut.log","w");
                for (wli = 0; wli < pc_log_dut.size(); wli++) begin
                        $fwrite(f, "%h - %d @ t = %t\n", pc_log_dut[wli].pc, pc_log_dut[wli].pc, pc_log_dut[wli].sim_time);
                end
                $fclose(f);
endtask


initial begin
                fork
                        begin
                                        wait(processor.sdram_init_done);
                                        $display("sdram init done");
                        end

                        begin
                                        @(posedge ref_halt);
                                        $display("ref module finished program");
                        end

                        begin
                                        wait(ebreak_start);
                                        $display("dut module finished program");
                        end
                join
end


initial begin
                integer time_reg, time_mem;
                integer iter;
                iter = 0;
                error = 0;

                fork
                        // wait both REF and DUT finish
                        begin
                                        wait(ref_halt && ebreak_start);
                                        repeat(10) @(posedge clk);
                        end

                        // wait for timeout
                        begin
                                        repeat(TB_TIMEOUT) @(posedge clk);
                                        $display("TB timeout, stop logging");
                                        // $stop();
                        end
                join_any
                disable fork;           // disable the fork wither both core finish running or timeout

                #100;
```

```
        $display("reg access count: ref: %d, dut: %d", reg_access_log_ref.size(), reg_access_log_dut.size());
        $display("mem access count: ref: %d, dut: %d", mem_access_log_ref.size(), mem_access_log_dut.size());

        // write log into log file
        write_csv();
        write_log();

        while (
                (reg_access_log_ref.size() > 0) ||
                (mem_access_log_ref.size() > 0)
        ) begin

                if (reg_access_log_ref.size() == 0 && mem_access_log_ref.size() > 0) begin
                        compare_mem_log();
                end else if (mem_access_log_ref.size() == 0 && reg_access_log_ref.size() > 0) begin
                        compare_reg_log();
                end else if (reg_access_log_ref.size() == 0 && mem_access_log_ref.size() == 0) begin
                        $display("test should be stopping soon...");
                end else if (reg_access_log_ref[0].sim_time <= mem_access_log_ref[0].sim_time)begin
                        compare_reg_log();
                end else begin
                        compare_mem_log();
                end

                iter++;
                if (iter > 100000) begin
                        $display("log access overflow, stopping");
                        $stop();
                end
        end

        fd = $fopen("./result.txt", "w");
        if (!fd) begin
                $display("file open failed");
                $stop();
        end

        compare_pc_log();

        if (error)begin
                $display("log mismatch, test failed");
                $display("run analysis_log.py to see details");
                $fwrite(fd, "fail");
        end

        else begin
                $display("all log match, test passed");
                $fwrite(fd, "success");
        end

        if (proc_ref.core_ref.reg_file[10] == 42) begin
                $display("golden module passed the test");
                if (processor.decode_inst.registers_inst.reg_bypass_inst.registers[10] == 42) begin
                        $display("DUT dilivered correct answer, test passed?");
                end else begin
                        $display("DUT failed the test");
                end
        end else begin
                $display("golden module failed the test! bad test?");
                if (processor.decode_inst.registers_inst.reg_bypass_inst.registers[10] == 42) begin
                        $display("WTF this should not happed");
                end
        end

        $fclose(fd);
        $stop();
    end

endmodule : reference_test_axi
```

```
module clkrst #(
        FREQ = FREQ
) (
        output logic clk,
        output logic rst_n,
        output logic go
);

        localparam period = 1e9/FREQ;   // in ns
        localparam half_period = period/2;

        initial begin
                clk                     = 1'b0;
                rst_n           = 1'b0;
                go                      = 1'b0;
                repeat(5) @(negedge clk);
                #100;
                rst_n           = 1'b1;
                repeat(233) @(negedge clk);
                go                      = 1'b1;
                @(negedge clk);
                go                      = 1'b0;
        end

        always #half_period begin
    clk = ~clk;
  end

endmodule : clkrst
```

```
package defines;

`ifndef _defines_sv_
`define _defines_sv_

// hardware target
	typedef enum logic [1:0] {
		INDEPNDENT,
		ALTERA,
		XILINX
	} target_t;

	localparam [1:0]	TARGET  = ALTERA;

	// boot options
	typedef enum logic[1:0] {
		BINARY_BOOT,	// boot from a bin file generated from gcc
		RARS_BOOT,		// boot from a rars compiled mif file
		FPGA_BOOT		// boot on FPGA, no jokes here
	} boot_type_t;
	localparam[1:0] BOOT_TYPE = BINARY_BOOT;

	// byte-varient endianess
	localparam LITTLE_ENDIAN    = 1'b0;
	localparam BIG_ENDIAN		= 1'b1;
	localparam ENDIANESS		= LITTLE_ENDIAN;

//	ISA define
	localparam	XLEN			= 32;			// RV32
	localparamN				= XLEN;			// in case I forget should be XLEN
instead of N
	localparam	OSC_FREQ		= 5e7;			// 50Mhz crystal oscillator on
FPGA board
	localparam	FREQ			= OSC_FREQ;		// targeted core clock from PLL


// TB config
	localparamTB_TIMEOUT		= 23333;		// test timeout in clock cycles

	// testbench will end after this time is reached

//	constant define
	localparamBYTES			= XLEN / 8;		// number of bytes in a word
	localparamTRUE			= 1;
	localparamFALSE			= 0;
	localparam	NULL			= 32'b0;		// used to repersent blank data
	localparamENABLE		= 1'b1;
	localparamDISABLE		= 1'b0;
	localparamVALID			= 1'b1;
	localparamINVALID		= 1'b0;
	localparamREADY			= 1'b1;
	localparamDONE			= 1'b1;
	localparamSET			= 1'b1;
	localparamCLEAR			= 1'b0;
	localparamGND			= 1'b0;


//	debug log option
	localparamTOP_DEBUG		= ENABLE;


	// sopported extension
	// this part shoule be only accessed by generate
	localparamI_SUPPORT		= TRUE;		// Base (Integer) operations, must implement
	localparamM_SUPPORT		= FALSE; // Integer Mult / Dvi, should implement
	localparamZ_SUPPORT		= FALSE; // Instruction-Fetch fence, required for xv6
	localparamA_SUPPORT		= FALSE; // Atomic instructions, required for xv6
	localparamF_SUPPORT		= FALSE; // Single-Precision FP, implement if enough FPGA space
	localparamD_SUPPORT		= FALSE; // Double-Precision FP, should not implement
	localparamQ_SUPPORT		= FALSE; // Quad-Precision FP, should not implement
```

```
	localparam C_SUPPORT		= FALSE; // Compressed Instructions, should not implement
	localparam CSR_SUPPORT		= FALSE; // Control and status register, required for xv6


	// branch predictor
	typedef enum logic[1:0] {
		P_TAKEN,		// predict taken
		P_N_TAKEN,		// predict not taken
		BTFNT			// back taken forward not taken
	} branch_predictor_t;
	localparam[1:0] PREDICTOR = P_N_TAKEN;
	localparam TAKEN = 1'b1;
	localparam NOT_TAKEN = 1'b0;

	// Opcode define
	typedef enum logic[6:0] {
		R		=	7'b0110011,
		I		=	7'b0010011,
		B		=	7'b1100011,
		LUI		=	7'b0110111,
		AUIPC	=	7'b0010111,
		JAL		=	7'b1101111,
		JALR	=	7'b1100111,
		LOAD	=	7'b0000011,
		STORE	=	7'b0100011,
		MEM		=	7'b0001111,		// for fence instruction
		SYS		=	7'b1110011,		// ECALL, EBREAK, and CSR
		ATOMIC	=	7'b0101111, // atomic instr
		NULL_OP=	7'b0000000
	} opcode_t;


// basic data type define
	typedef logic [XLEN-1:0]		data_t;
	typedef logic [7:0]			byte_t;
	typedef logic [(XLEN/2-1):0]	half_word_t;
	typedef logic [2:0]			funct3_t;
	typedef logic [6:0]			funct7_t;
	typedef logic [11:0]			imm_t;		// only for I type instruction


	typedef struct packed {
		byte_t b0;
		byte_t b1;
		byte_t b2;
		byte_t b3;
	} word_t;

	typedef enum logic[4:0] {
		X0, X1, X2, X3, X4, X5, X6, X7,
		X8, X9, X10, X11, X12, X13, X14, X15,
		X16, X17, X18, X19, X20, X21, X22, X23,
		X24, X25, X26, X27, X28, X29, X30, X31
	} r_t;	// simulator should assign 5'd0 - 5'd31 in order


	localparam[XLEN-1:0]		NOP		= 32'h0000_0013;	// ADDI x0, x0, 0
	// localparam		[XLEN-1:0]		HALT	= 32'h0000_0063;// BEQ x0, x0, 0
	localparam[XLEN-1:0]		EBREAK	= 32'h0010_0073;
	localparam[XLEN-1:0]		ECALL	= 32'h0000_0073;


// Funt3 define
	// R type funct3
	localparam[2:0]	ADD			=	3'b000;		// rd <= rs1 + rs2, no overflow exception
	localparam[2:0]	SUB			=	3'b000;		// rd <= rs1 - rs2, no overflow exception
	localparam[2:0]	AND			=	3'b111;
	localparam[2:0]	OR			=	3'b110;
	localparam[2:0]	XOR			=	3'b100;
	localparam[2:0]	SLT			=	3'b010;		// set less than, rd <= 1 if rs1 < rs2
```

```
        localparam[2:0]     SLTU            =       3'b011;         // set less than unsigned, rd <= 1 if rs1 < rs2
        localparam[2:0]     SLL             =       3'b001;         // logical shift left, rd <= rs1 << rs2[4:0]
        localparam[2:0]     SRL             =       3'b101;         // logical shift right rd <= rs1 >> rs2[4:0]
        localparam[2:0]     SRA             =       3'b101;         // arithmetic shift right
        // MUL (same opcode as R) funct3
        localparam[2:0]     MUL             =       3'b000;         // (sign rs1*sign rs2)[XLEN-1:0] => rd
        localparam[2:0]     MULH            =       3'b001;         // (sign rs1*sign rs2)[2*XLEN-1:XLEN] => rd
        localparam[2:0]     MULHSU          =       3'b010;         // (sign rs1*unsign rs2)[2*XLEN-1:XLEN] => rd
        localparam[2:0]     MULHU           =       3'b011;         // (unsign rs1*unsign rs2)[2*XLEN-1:XLEN] => rd
        localparam[2:0]     DIV             =       3'b100;         // sign rs1 / sign rs2
        localparam[2:0]     DIVU            =       3'b101;         // unsign rs1 / unsign rs2
        localparam[2:0]     REM             =       3'b110;         // sign rs1 % sign rs2
        localparam[2:0]     REMU            =       3'b111;         // unsign rs1 % unsign rs2

        // I type funct3
        localparam[2:0]     ADDI            =       3'b000;
        localparam[2:0]     ANDI            =       3'b111;
        localparam[2:0]     ORI             =       3'b110;
        localparam[2:0]     XORI            =       3'b100;
        localparam[2:0]     SLTI            =       3'b010;         // Set less than immediate, rd <= 1 if rs1 < imm
        localparam[2:0]     SLTIU           =       3'b011;         // Set less than immediate unsigned, rd <= 1 if rs1 < imm
        localparam[2:0]     SLLI            =       3'b001;         // logical shift left imm
        localparam[2:0]     SRLI            =       3'b101;         // logical shift right imm
        localparam[2:0]     SRAI            =       3'b101;         // arithmetic shift right imm

        // B type funct3                                    branch imm have to shift left for 1
        localparam[2:0]     BEQ             =       3'b000;         // branch if rs1 == rs2
        localparam[2:0]     BNE             =       3'b001;         // branch if rs1 != rs2
        localparam[2:0]     BLT             =       3'b100;         // branch if rs1 < rs2 signed
        localparam[2:0]     BLTU            =       3'b110;         // branch if rs1 < rs2 unsigned
        localparam[2:0]     BGE             =       3'b101;         // branch if rs1 >= rs2 signed
        localparam[2:0]     BGEU            =       3'b111;         // branch if rs1 >= rs2 unsigned

        // U type have no funct3
        //localparam        [2:0]   LUI     =       3'b000;     // rd <= {imm, 12'b0}
        //localparam        [2:0]   AUIPC   =       3'b000;     // rd <= (pc_of_auipc + {imm, 12'b0})

        // J type have no funct3
        //localparam        [2:0]   JAL     =       3'b000;     // jump and link, rd <= pc_of_jal + 4, pc <= (pc_of_jal +
imm << 1)
        //localparam        [2:0]   JALR    =       3'b000;     // jump and link registor, rd <= (pc_of_jalr + 4),
                                                                                                        //  pc  <=
(rs1 + imm) && 0xfffe (set the last bit is always 0)

        // S type funct3 - Load
        localparam[2:0]     LB              =       3'b000;         // load 8 bits and sign extend to 32 bits
        localparam[2:0]     LH              =       3'b001;         // load 16 bits and sign extend to 32 bits
        localparam[2:0]     LW              =       3'b010;         // rd <= mem[rs1 + imm]
        localparam[2:0]     LBU             =       3'b100;         // load 8 bits and zero extend to 32 bits
        localparam[2:0]     LHU             =       3'b101;         // load 16 bits and zero extend to 32 bits

        // S type funct3 - Store
        localparam[2:0]     SB              =       3'b000;         // mem[rs1 + imm] <= rs2.byte0
        localparam[2:0]     SH              =       3'b001;         // mem[rs1 + imm] <= rs2.word0
        localparam[2:0]     SW              =       3'b010;         // mem[rs1 + imm] <= rs2
        //localparam        [2:0]   SBU     =       3'b100; not used
        //localparam        [2:0]   SHU     =       3'b101; not used

        // Fence (Memory ordering) funct3
        localparam[2:0]     FENCE           =       3'b000;
        localparam[2:0]     FENCEI          =       3'b001;

        // Atomic funct3
        localparam[2:0]     A_32            =       3'b010;
        localparam[2:0]     A_64            =       3'b011;     // not supported

        // SYS (ECALL, EBREAK, and CSR) funct3
        localparam[2:0]     CSRRW           =       3'b001;     // Atomic read/write CSR
        localparam[2:0]     CSRRS           =       3'b010;     // Atomic Read and Clear Bits
        localparam[2:0]     CSRRC           =       3'b011;
```

```
        localparam[2:0]     CSRRWI          =       3'b101;
        localparam[2:0]     CSRRSSI         =       3'b110;
        localparam[2:0]     CSRRCI          =       3'b111;

// funct5 define (Atomic only)
        typedef enum logic[4:0] {
                LR                          =       5'b00010,  // load-reserved
                SC                          =       5'b00011,  // store-conditional
                AMOSWAP                     =       5'b00001,  // rd <= mem[rs1], mem[rs1] <= rs2(old)
                AMOADD                      =       5'b00000,  // rd <= mem[rs1], mem[rs1] <= rs2(old) + rd
                AMOXOR          =       5'b00100,
                AMOAND                      =       5'b01100,
                AMOOR           =       5'b01000,
                AMOMIN          =       5'b10000,
                AMOMAX                      =       5'b10100,
                AMOMINU                     =       5'b11000,
                AMOMAXU                     =       5'b11100
        } funct5_t; // only for ATOMIC instruction


// funct7 define (R only)
        localparam[6:0]     M_INSTR         = 7'b000_0001;      // indicate this R type instruction is mult or div

        // little endian mask
        localparam[XLEN-1:0]        B_MASK_LITTLE = 32'hFF_00_00_00;
        localparam[XLEN-1:0]        H_MASK_LITTLE = 32'hFF_FF_00_00;
        localparam[XLEN-1:0]        W_MASK_LITTLE = 32'hFF_FF_FF_FF;
        localparam[BYTES-1:0]       B_EN_LITTLE = 4'b1000;
        localparam[BYTES-1:0] H_EN_LITTLE = 4'b1100;
        localparam[BYTES-1:0] W_EN_LITTLE = 4'b1111;

        // big endian mask
        localparam[XLEN-1:0]        B_MASK_BIG = 32'h00_00_00_FF;
        localparam[XLEN-1:0]        H_MASK_BIG = 32'h00_00_FF_FF;
        localparam[XLEN-1:0]        W_MASK_BIG = 32'hFF_FF_FF_FF;
        localparam[BYTES-1:0]       B_EN_BIG = 4'b0001;
        localparam[BYTES-1:0]       H_EN_BIG = 4'b0011;
        localparam[BYTES-1:0]       W_EN_BIG = 4'b1111;


        // instruction type define
        typedef struct packed{
                funct7_t    funct7;
                r_t                         rs2;
                r_t                         rs1;
                funct3_t    funct3;
                r_t                         rd;
                opcode_t    opcode;
        } instr_t;              // R (base) type

        typedef struct packed{
                imm_t               imm;
                r_t                         rs1;
                funct3_t    funct3;
                r_t                         rd;
                opcode_t    opcode;
        } instr_i_t;// I type

        typedef struct packed{
                logic[11:5]imm_h;
                r_t                         rs2;
                r_t                         rs1;
                funct3_t    funct3;
                logic[4:0]  imm_l;
                opcode_t    opcode;
        } instr_s_t;// S type

        typedef struct packed {
                funct5_t    funct5;
                logic               aq;
```

```
                logic           rl;
                r_t                     rs2;
                r_t                     rs1;
                funct3_t    funct3;
                r_t                     rd;
                opcode_t    opcode;
        } instr_a_t;            // Atomic type


function data_t sign_extend;        // sign extend 12bit imm
        input imm_t imm;
        return data_t'({ {20{imm[11]}}, {imm[11:0]} });
endfunction


function data_t sign_extend_h;    // sign extend 16-bit half word
        input half_word_t imm;
        return data_t'({ {16{imm[15]}}, {imm[15:0]} });
endfunction


function data_t sign_extend_b;    // sign extend 8 bit byte
        input byte_t imm;
        return data_t'({ {24{imm[7]}}, {imm[7:0]} });
endfunction


function data_t zero_extend;        // zero extend 12bit imm
        input imm_t imm;
        return data_t'({ {20'b0}, {imm[11:0]} });
endfunction


function data_t zero_extend_h;    // zero extend 16-bit half word
        input half_word_t imm;
        return data_t'({ {16'b0}, {imm[15:0]} });
endfunction


function data_t zero_extend_b;    // zero extend 8 bit byte
        input byte_t imm;
        return data_t'({ {24'b0}, {imm[7:0]} });
endfunction


function data_t get_imm;                // extract immidiate value from instruction
        input instr_t instr;
        unique case (instr.opcode)
                LUI:            return data_t'( { instr[31:12], 12'b0 } );
                AUIPC:          return data_t'( { instr[31:12], 12'b0 } );
                JAL:            return data_t'( { 32'd4 } );        // pc + 4 for ALU
                JALR:           return data_t'( { 32'd4 } );        // pc + 4 for ALU
                B:                      return data_t'( { { 20{ instr[31] } } , instr[7], instr[30:25], instr[11:8], 1'b0 } );
                LOAD:           return data_t'( { { 20{ instr[31] } } , instr[31:20] } );
                STORE:          return data_t'( { { 20{ instr[31] } } , instr[31:25], instr[11:7] } );
                I:                      return data_t'( { { 20{ instr[31] } } , instr[31:20] } );
                default:    return NULL;
        endcase
endfunction


function data_t swap_endian;
        input data_t data;
        return      data_t'     ({{data[7:0]},
                                                {data[15:8]},
                                                {data[23:16]},
                                                {data[31:24]}});
endfunction
```

```
// fwd mux ctrl signal types
typedef enum logic[1:0] {
        RS_ID_SEL       = 2'b00,
        EX_ID_SEL       = 2'b01,
        MEM_ID_SEL      = 2'b10,
        WB_ID_SEL       = 2'b11
} id_fwd_sel_t;


typedef enum logic[1:0] {
        RS_EX_SEL       = 2'b00,
        MEM_EX_SEL      = 2'b01,
        WB_EX_SEL       = 2'b10
} ex_fwd_sel_t;


typedef enum logic[1:0] {
        RS_MEM_SEL      = 2'b00,
        WB_MEM_SEL      = 2'b01
} mem_fwd_sel_t;


// sometimes i mistype "$stop()" as "stop()"...
function void stop;
        $stop();
endfunction


// register names
localparam[4:0]     ZERO    = X0;
localparam[4:0]     RA              = X1;
localparam[4:0]     SP              = X2;
localparam[4:0]     GP              = X3;
localparam[4:0]     TP              = X4;
localparam[4:0]     T0              = X5;
localparam[4:0]     T1              = X6;
localparam[4:0]     T2              = X7;
localparam[4:0]     S0              = X8;
localparam[4:0]     S1              = X9;
localparam[4:0]     A0              = X10;
localparam[4:0]     A1              = X11;
localparam[4:0]     A2              = X12;
localparam[4:0]     A3              = X13;
localparam[4:0]     A4              = X14;
localparam[4:0]     A5              = X15;
localparam[4:0]     A6              = X16;
localparam[4:0]     A7              = X17;
localparam[4:0]     S2              = X18;
localparam[4:0]     S3              = X19;
localparam[4:0]     S4              = X20;
localparam[4:0]     S5              = X21;
localparam[4:0]     S6              = X22;
localparam[4:0]     S7              = X23;
localparam[4:0]     S8              = X24;
localparam[4:0]     S9              = X25;
localparam[4:0]     S10             = X26;
localparam[4:0]     S11             = X27;
localparam[4:0]     T3              = X28;
localparam[4:0]     T4              = X29;
localparam[4:0]     T5              = X30;
localparam[4:0]     T6              = X31;

`endif

endpackage : defines
```

```
import defines::*;

module hazard_ctrl (
    // input signal
    input   instr_t         instr_f,
    input   instr_t         instr_d,
    input   instr_t         instr_x,
    input   instr_t         instr_m,
    input   instr_t         instr_w,

    input   logic           instr_valid_d,

    input   logic           ex_rd_write,
    input   logic           mem_rd_write,
    input   logic           wb_rd_write,

    input   logic           sdram_init_done,
    input   logic           execute_busy,
    input   logic           mem_access_done,
    input   logic           branch_taken_d,
    input   logic           branch_taken_x,

    // forwarding signal to id stage
    output  id_fwd_sel_t    fwd_id_rs1,
    output  id_fwd_sel_t    fwd_id_rs2,

    // forwarding signal to ex stage
    output  ex_fwd_sel_t    fwd_ex_rs1,
    output  ex_fwd_sel_t    fwd_ex_rs2,

    // forwarding signal to mem stage
    output  mem_fwd_sel_t   fwd_mem_rs1,
    output  mem_fwd_sel_t   fwd_mem_rs2,

    // stall signal
    output  logic           stall_pc,
    output  logic           stall_if_id,
    output  logic           stall_id_ex,
    output  logic           stall_ex_mem,
    output  logic           stall_mem_wb,

    // flush signal
    output  logic           flush_pc,
    output  logic           flush_if_id,
    output  logic           flush_id_ex,
    output  logic           flush_ex_mem,
    output  logic           flush_mem_wb
);

    r_t id_rs1, id_rs2, ex_rs1, ex_rs2, mem_rs1, mem_rs2;
    always_comb begin : rs_assign
        id_rs1  = instr_d.rs1;
        id_rs2  = instr_d.rs2;
        ex_rs1  = instr_x.rs1;
        ex_rs2  = instr_x.rs2;
        mem_rs1 = instr_m.rs1;
        mem_rs2 = instr_m.rs2;
    end

    r_t ex_rd, mem_rd, wb_rd;
    always_comb begin : rd_assign
        ex_rd   = instr_x.rd;
        mem_rd  = instr_m.rd;
        wb_rd   = instr_w.rd;
    end

    logic id_rs1_read, id_rs2_read, ex_rs1_read, ex_rs2_read, mem_rs1_read, mem_rs2_read;
    always_comb begin : rs_read_assign
        id_rs1_read     =   ((instr_d.opcode == B) ||
                            (instr_d.opcode == JALR));
```

```
    id_rs2_read     =   (instr_d.opcode == B) ||
                        (instr_d.opcode == STORE);

    ex_rs1_read     =   ((instr_x.opcode == R) ||
                        (instr_x.opcode == I) ||
                        (instr_x.opcode == STORE) ||
                        (instr_x.opcode == LOAD) ||
                        (instr_x.opcode == JALR));

    ex_rs2_read     =   ((instr_x.opcode == R));

    mem_rs1_read    =   ((instr_m.opcode == LOAD) ||
                        (instr_m.opcode == STORE));

    // mem_rs2_read =   DISABLE;
    mem_rs2_read    =   (instr_m.opcode == STORE);
end

logic hazard_ex2id_1, hazard_ex2id_2;
logic hazard_mem2id_1, hazard_mem2id_2;
logic hazard_wb2id_1, hazard_wb2id_2;
always_comb begin : id_hazard_detect
    hazard_ex2id_1 =    (id_rs1_read) &&
                        (id_rs1 != X0) &&
                        (ex_rd_write) &&
                        (ex_rd == id_rs1);

    hazard_ex2id_2 =    (id_rs2_read) &&
                        (id_rs2 != X0) &&
                        (ex_rd_write) &&
                        (ex_rd == id_rs2);

    hazard_mem2id_1 =   (id_rs1_read) &&
                        (id_rs1 != X0) &&
                        (mem_rd_write) &&
                        (mem_rd == id_rs1);

    hazard_mem2id_2 =   (id_rs2_read) &&
                        (id_rs2 != X0) &&
                        (mem_rd_write) &&
                        (mem_rd == id_rs2);

    hazard_wb2id_1 =    (id_rs1_read) &&
                        (id_rs1 != X0) &&
                        (wb_rd_write) &&
                        (wb_rd == id_rs1);

    hazard_wb2id_2 =    (id_rs2_read) &&
                        (id_rs2 != X0) &&
                        (wb_rd_write) &&
                        (wb_rd == id_rs2);
end

logic hazard_mem2ex_1, hazard_mem2ex_2;
logic hazard_wb2ex_1, hazard_wb2ex_2;
always_comb begin : ex_hazard_detect
    hazard_mem2ex_1 =   (ex_rs1_read) &&
                        (ex_rs1 != X0) &&
                        (mem_rd_write) &&
                        (mem_rd == ex_rs1);

    hazard_mem2ex_2 =   (ex_rs2_read) &&
                        (ex_rs2 != X0) &&
                        (mem_rd_write) &&
                        (mem_rd == ex_rs2);

    hazard_wb2ex_1 =    (ex_rs1_read) &&
                        (ex_rs1 != X0) &&
                        (wb_rd_write) &&
```

```
                (wb_rd == ex_rs1);

   hazard_wb2ex_2 =    (ex_rs2_read) &&
                (ex_rs2 != X0) &&
                (wb_rd_write) &&
                (wb_rd == ex_rs2);
end

logic hazard_wb2mem_1, hazard_wb2mem_2;
logic hazard_wb2mem;
always_comb begin : mem_hazard_detect
   hazard_wb2mem_1 =   (mem_rs1_read) &&
                (mem_rs1 != X0) &&
                (wb_rd_write) &&
                (wb_rd == mem_rs1);

   hazard_wb2mem_2 =   (mem_rs2_read) &&
                (mem_rs2 != X0) &&
                (wb_rd_write) &&
                (wb_rd == mem_rs2);
   hazard_wb2mem   =   hazard_wb2mem_1 || hazard_wb2mem_2;
end

always_comb begin : forward_sig_assign
   // forwarding signal to id stage
   fwd_id_rs1  =   hazard_ex2id_1  ? EX_ID_SEL :
               hazard_mem2id_1 ? MEM_ID_SEL :
               hazard_wb2id_1  ? WB_ID_SEL :
               RS_ID_SEL;
   fwd_id_rs2  =   hazard_ex2id_2  ? EX_ID_SEL :
               hazard_mem2id_2 ? MEM_ID_SEL :
               hazard_wb2id_2  ? WB_ID_SEL :
               RS_ID_SEL;

   // forwarding signal to ex stage
   fwd_ex_rs1  =   hazard_mem2ex_1 ? MEM_EX_SEL :
               hazard_wb2ex_1  ? WB_EX_SEL :
               RS_EX_SEL;
   fwd_ex_rs2  =   hazard_mem2ex_2 ? MEM_EX_SEL :
               hazard_wb2ex_2  ? WB_EX_SEL :
               RS_EX_SEL;

   // forwarding signal to mem stage
   fwd_mem_rs1 =   hazard_wb2mem_1 ? WB_MEM_SEL :
               RS_MEM_SEL;
   fwd_mem_rs2 =   hazard_wb2mem_2 ? WB_MEM_SEL :
               RS_MEM_SEL;
end

logic data_mem_stall;   // pipeline stall from data memory access
always_comb begin : data_mem_stall_assign
   data_mem_stall = ((instr_m.opcode == STORE) || (instr_m.opcode == LOAD)) && ~mem_access_done;
end

// hazard when load/jump follows a branch
// a true hazard and does not be resolven by forwarding
// both hazard 4 and harrard 5 requires to stall pipeline on F and D stages
logic load_hazard_1a, load_hazard_1b; // load - branch
logic load_hazard_2a, load_hazard_2b; // load - whatever - branch
logic load_hazard_1 , load_hazard_2;
logic decode_use_rs1, decode_use_rs2; // JALR and Branch

always_comb begin : load_branch_stall   // a true hazzard that must stall
   decode_use_rs1  =   ((instr_d.opcode == JALR) || (instr_d.opcode == B));

   decode_use_rs2  =   (instr_d.opcode == B);

   load_hazard_1a  =   (instr_x.opcode == LOAD) &&
                (decode_use_rs1) &&
                (instr_x.rd != X0) &&
```

```
                (instr_x.rd == instr_d.rs1);

    load_hazard_1b  =   (instr_x.opcode == LOAD) &&
                (decode_use_rs2) &&
                (instr_x.rd != X0) &&
                (instr_x.rd == instr_d.rs2);

    load_hazard_1   =   load_hazard_1a || load_hazard_1b;

    load_hazard_2a  =   (instr_m.opcode == LOAD) &&
                (decode_use_rs1) &&
                (instr_m.rd != X0) &&
                (instr_m.rd == instr_d.rs1);

    load_hazard_2b  =   (instr_m.opcode == LOAD) &&
                (decode_use_rs2) &&
                (instr_m.rd != X0) &&
                (instr_m.rd == instr_d.rs2);

    load_hazard_2   =   load_hazard_2a || load_hazard_2b;
  end

  // TODO: when seeing a FENSE instruction in decode stage, stall PC and IF untie see
  // TODO: add fense bit in the pipeline stages to indicate instruction after finse done exe
  always_comb begin : stall_assign
      stall_pc        = data_mem_stall || ~sdram_init_done || (load_hazard_1 && ~data_mem_stall) || (load_hazard_2 && ~data_mem_stall) ||
execute_busy;
      stall_if_id     = data_mem_stall || ~sdram_init_done || (load_hazard_1 && ~data_mem_stall) || (load_hazard_2 && ~data_mem_stall) ||
execute_busy;
    stall_id_ex     = data_mem_stall || ~sdram_init_done || execute_busy;
    stall_ex_mem    = data_mem_stall || ~sdram_init_done;
    stall_mem_wb    = data_mem_stall || ~sdram_init_done;   // stall for mem-mem fwd
  end

  logic jump_d, jump_x;   // jump instruction in decode/exe stage
  logic branch_d, branch_x;
  always_comb begin : flush_crtl_signal_assign
    jump_d = (instr_d.opcode == JAL) || (instr_d.opcode == JALR);
    jump_x = (instr_x.opcode == JAL) || (instr_x.opcode == JALR);
    branch_d = (instr_d.opcode == B) && branch_taken_d;
    branch_x = (instr_x.opcode == B) && branch_taken_x;
  end


  always_comb begin : flush_assign
    if (load_hazard_1) begin
      flush_pc = DISABLE;
    end else if (load_hazard_2) begin
      flush_pc = (~data_mem_stall) ? ((jump_d || branch_d) && instr_valid_d) : DISABLE;
    end else
      flush_pc = (jump_d || branch_d) && instr_valid_d;

    flush_if_id     = jump_x || branch_x;
    flush_id_ex     = DISABLE;
    flush_ex_mem    = DISABLE;
    flush_mem_wb    = DISABLE;
  end

endmodule : hazard_ctrl
```

```
#
TARGET = riscy

#
ALT_DEVICE_FAMILY = soc_cv_av
#SOCEDS_ROOT = $(SOCEDS_DEST_ROOT)
#HWLIBS_ROOT = $(SOCEDS_ROOT)/ip/altera/hps/altera_hps/hwlib
#CROSS_COMPILE = arm-linux-gnueabihf-
#CFLAGS = -g -Wall -D$(ALT_DEVICE_FAMILY) -I$(HWLIBS_ROOT)/include/$(ALT_DEVICE_FAMILY) -
I$(HWLIBS_ROOT)/include/
CFLAGS = -g -Wall -D $(ALT_DEVICE_FAMILY) -I include/
LDFLAGS =  -g -Wall

#CC = $(CROSS_COMPILE)gcc
CC = gcc
#ARCH= arm

build: $(TARGET)
$(TARGET): main.o
	$(CC) $(LDFLAGS)   $^ -o $@
%.o : %.c
	$(CC) $(CFLAGS) -c $< -o $@

.PHONY: clean
clean:
	rm -f $(TARGET) *.a *.o *~
```

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include "include/hps.h"

#define RISCY_LITTLE_ENDIAN

// global define
const uint32_t h2f_lw_base                  = (unsigned int) ALT_LWFPGASLVS_OFST;
const uint32_t h2f_base                             = (unsigned int) 0xC0000000;

// const uint32_t elf_load_offset  = 0x00004000;       // in words (4 byte)
const uint32_t elf_load_offset      = 0x00000000;       // in words (4 byte)

// sdram define
#define SDRAM
#ifdef SDRAM
const uint32_t sdram_range                  = 0x03FFFFFF;       // 0x0 - 0x3ffffff
const uint32_t sdram_addr_mask = 0x03FFFFFC;        // word-aligned access
const uint32_t sdram_offset                 = 0x00000000;       // offset from bridge
const uint32_t sdram_size_byte   = 0x04000000;      // 512Mb
const uint32_t sdram_size_word  = 0x01000000;       // 64MB
#else
const uint32_t sdram_range                  = 0x0007FFFF;
const uint32_t sdram_addr_mask = 0x0007FFFC;
const uint32_t sdram_offset                 = 0x00000000;
const uint32_t sdram_size_byte   = 0x00080000;
const uint32_t sdram_size_word  = 0x00020000;
#endif


// 7-seg display define
const uint32_t seg_range                    = 0x0000001F;       // 0x0 - 0x1f
const uint32_t seg_offset                   = 0x04000000;       // offset from bridge
const uint32_t seg_size_byte        = 0x00000020;       // 32 bytes
const uint32_t seg_size_word        = 0x00000008;       // 8 words (6 needed)
const uint32_t seg_addr_mask        = 0xFFFFFFFC;       // word-align
const uint32_t seg_data_mask        = 0x0000000F;       // only first byte valid


// uart serial define
const uint32_t uart_range                   = 0x0000001F;
const uint32_t uart_offset                  = 0x04010000;
const uint32_t uart_size_byte       = 0x00000020;
const uint32_t uart_size_word       = 0x00000008;
const uint32_t uart_data_mask       = 0x000000FF;


uint32_t swap_endian (uint32_t in) {
        return      ((in>>24)&0xff) |                       // move byte 3 to byte 0
                            ((in<<8)&0xff0000) |            // move byte 1 to byte 2
                            ((in>>8)&0xff00) |              // move byte 2 to byte 1
                            ((in<<24)&0xff000000);
}


void* map_addr (int pa_base, int size_byte) {
        uint32_t page_mask, page_size;
        int fd;
        void* return_va;
        if( ( fd = open( "/dev/mem", ( O_RDWR | O_SYNC ) ) ) == -1 ) {
                printf( "ERROR: could not open \"/dev/mem\"...\n" );
                return( (void*)(-1) );
        }
```

```
	// get page size in byte
	page_size = sysconf(_SC_PAGESIZE);

	// in number of allocated page
	uint32_t alloc_mem_size = (((size_byte / page_size) + 1) * page_size);
	page_mask = (page_size - 1);
	return_va = mmap(
		NULL,
		alloc_mem_size,
		( PROT_READ | PROT_WRITE ),
		MAP_SHARED,
		fd,
		(pa_base & ~page_mask)
	);

	if( return_va == MAP_FAILED ) {
		printf( "ERROR: mmap() failed...\n" );
		close( fd );
		return( (void*)(-1) );
	}
	close( fd );
	return (return_va);
}


int unmap_addr (void* vp_base, u_int32_t unmap_size_byte) {
	if( munmap( vp_base, unmap_size_byte ) != 0 ) {
		printf( "ERROR: munmap() failed...\n" );
		return( -1 );
	}
	return (0);
}


// SDRAM controlling functions
void* init_sdram() {
	uint32_t sdram_pa_base		= h2f_base + sdram_offset;	// PA of sdram from HPS's perspective
	return map_addr (sdram_pa_base, sdram_size_byte);
}

int clean_sdram (void* vp_base) {
	return unmap_addr (vp_base, 0x10000);
}

uint32_t read_sdram (uint32_t * addr) {
	return *((uint32_t *)addr);
}

void write_sdram (uint32_t* addr, uint32_t data) {
	*addr = data;
}

// off in word, not byte
int touch_sdram (void* base, uint32_t off) {
	uint32_t data = rand();
	write_sdram(((uint32_t *)base) + off, data);
	uint32_t x = read_sdram(((uint32_t *)base) + off);
	uint32_t sdram_pa_base		= h2f_base + sdram_offset;	// PA of sdram from HPS's perspective
	if (x == data) {
		printf("touche word off: %x, PA: %x success \n", off, (sdram_pa_base) + (off * 4));
		return 0;
	} else {
		printf("touche PA %x fail \n", (sdram_pa_base) + (off * 4));
		return -1;
	}
}

void touch_sdram_range (void* base, int start, int step) {
	int i;
	for (i = 0; i > -1; i += step) {
```

```
                touch_sdram(base, start + (step * i));
        }
}


// 7-Seg controlling functions

void* init_seg() {
        uint32_t seg_pa_base = h2f_base + seg_offset;        // PA of sdram from HPS's perspective
        return map_addr (seg_pa_base, seg_size_byte);
}

int clean_seg (void* vp_base) {
        return unmap_addr (vp_base, seg_size_byte);
}

void set_seg_single (void* vp, int index, uint32_t number) {
        uint32_t hex_seg_digit = number & seg_data_mask;
        #ifdef RISCY_LITTLE_ENDIAN
                hex_seg_digit = swap_endian(hex_seg_digit);
        #endif
        *((uint32_t*)vp+index) = hex_seg_digit;
}

void set_seg (void* vp, uint32_t number) {
        int i;
        for (i = 0; i < 6; i++) {
                set_seg_single (vp, i, number);
                number = number >> 4;
        }
        return;
}

void boot_load (char* filename, int swap) {
        // prep work, accocate memory
        int i;
        FILE* file_ptr;
        file_ptr = fopen(filename,"rb");
        if (!file_ptr) {
                perror("fopen");
                exit(EXIT_FAILURE);
        }

        struct stat st;
   if (stat(filename, &st) == -1) {
      perror("stat");
      exit(EXIT_FAILURE);
   }

        uint32_t instr_size_word;
        instr_size_word = (st.st_size) >> 2;
        printf("bootloader start, boot sector size: %d words\n", instr_size_word);

        usleep(100);

        char* instr_arr_byte = malloc(st.st_size);
        uint32_t* instr_arr = (uint32_t*)instr_arr_byte;

        fread(instr_arr_byte, st.st_size, 1, file_ptr);
        fclose(file_ptr);

        // swap the endianess of each instruction as we are using big endian for now

        if (swap) {
                printf("swapping endianess...\n");
                for (i = 0; i < instr_size_word; i++) {
                        instr_arr[i] = swap_endian(instr_arr[i]);
                }
        }
```

```
	// map sdram into our own memory space
	printf("mapping shared memory space...\n");
	usleep(100);

	uint32_t* sdram_vp = (uint32_t*)init_sdram();


	// write the data into sdram
	printf("bootloading in progress...\n");
	usleep(100);
	for (i = 0; i < instr_size_word; i++) {
		write_sdram(sdram_vp + i + elf_load_offset, instr_arr[i]);
	}

	// read sdram to check data corruption
	printf("performing sanity check...\n");
	usleep(100);
	uint32_t sanity_check;
	int err = 0;
	for (i = 0; i < instr_size_word; i++) {
		sanity_check = read_sdram(sdram_vp + i + elf_load_offset);
		if (sanity_check != instr_arr[i]) {
			printf("data mismatch at word %d\n",(i));
			printf("expecting: %x, get: %x\n\n", instr_arr[i], sanity_check);
			printf("bootloader sanity check failed, existing... \n");
			exit(EXIT_FAILURE);
		}
	}

	// unmap sdram from our memory space
	printf("bootloading complete, cheaning up the mess...\n");
	usleep(100);
	clean_sdram(sdram_vp);

	// free allocated pointers
	// free(instr_arr);
	free(instr_arr_byte);

	// display message;
	if (err == 0) {
		printf("bootload success\n");
	}
}


void sanity_test_seg() {
	printf("starting 7seg test, seg should show non-zero number\n");
	usleep(100);
	void* seg_vp = (void*)(init_seg());
	set_seg (seg_vp, 0x00123456);
	clean_seg(seg_vp);
	usleep(100);
}

void sdram_range_test() {
	void* sdram_vp = init_sdram();
	touch_sdram(sdram_vp, sdram_size_byte & sdram_addr_mask);
	clean_sdram(sdram_vp);
}


void sdram_random_rw_test (int iter) {
	int i;
	uint32_t data;
	uint32_t result;
	void* sdram_vp = init_sdram();
	for (i = 0; i < iter; i++) {
		data = rand();
		write_sdram((uint32_t*)(sdram_vp + i*4), data);
		result = read_sdram((uint32_t*)(sdram_vp + i*4));
```

```
                        if (data != result) {
                                printf("sdram random rw test failed at iter %d\n", i);
                                printf("expecting %x, get %x\n", data, result);
                                //break;
                        }
                }
                clean_sdram(sdram_vp);
}


void sanity_test_sdram() {
                printf("starting sdram sanity test: range test...\n");
                usleep(1000);
                sdram_range_test();
                printf("starting sdram sanity test: rw test...\n");
                usleep(1000);
                sdram_random_rw_test(1000);
}

void* init_uart() {
                uint32_t uart_pa_base = h2f_base + uart_offset;          // PA of sdram from HPS's perspective
                return map_addr (uart_pa_base, uart_size_byte);
}

int clean_uart(void* vp_base) {
                return unmap_addr (vp_base, uart_size_byte);
}

void uart_put_str (char* str, int len) {
                int i;
                char c;
                uint8_t char_byte;
                uint32_t char_word;
                uint32_t* uart_vp = (uint32_t*)init_uart();
                for (i = 0; i < len; i++) {
                        c = str[i];
                        char_byte = (uint8_t) c;
                        char_word = ((uint32_t) char_byte) & uart_data_mask;
                        #ifdef RISCY_LITTLE_ENDIAN
                        char_word = swap_endian(char_word);
                        #endif
                        *(uart_vp) = char_word;
                }
                clean_uart(uart_vp);
}


void sanity_test_uart() {
                char greeting[15] = "Hello RISCY\r\n";
                printf("performing uart serial test...\n");
                printf("check serial for valid output\n\n");
                uart_put_str(greeting, 15);
                usleep(100);
}


void boot () {
                printf("starting RISCY bootloading process...\n");
                sanity_test_sdram();
                sanity_test_seg();
                sanity_test_uart();
                boot_load("./riscy.elf", 1);
}


uint32_t peek (uint32_t addr) {
                void* sdram_vp = init_sdram();
                uint32_t value =  read_sdram(sdram_vp + addr);
                clean_sdram(sdram_vp);
                return value;
```

```
}


int main (int argc, char *argv[]) {
	if( argc == 2 ) {
		printf("The argument supplied is %s\n", argv[1]);
		char* cmd = argv[1];

		if (strcmp(cmd, "boot") == 0) {
			boot();
		} else {
			printf("Invalid argument\n");
		}

	} else if( argc == 3 ) {
		char* cmd = argv[1];
		char* addr_str = argv[2];

		if (strcmp(cmd, "peek") == 0) {
			uint32_t addr = (uint32_t)strtol(addr_str, NULL, 16);
			uint32_t value = peek(addr);
			printf("addr: %X, value_se: %X, value_be: %X\n", addr, value, swap_endian(value));
		} else {
			printf("Invalid argument\n");
		}
	} else {
		printf("Invalid argument.\n");
	}
	return 0;
}
```

```
# -g: enable gdb symbols
# -wall: enable all warnings
# -O0: only optimization for compilation time
# -mdiv: use hardware int division
# -mno-fdiv: do not use hardware fp div
# -no-pie: don't make position independent executable (PIE)
# -mstrict-align: Do not generate unaligned memory accesses
# -mlittle-endian: generate little endian code
# -march=rv32im: rv32im ISA
# -mabi=ilp32: 32 bit data type without fp
# -lstdc++: enable c++ compile
# -mbranch-cost=1: low branch cost, so hopefully disable instruction reorder
# -mexplicit-relocs: use assembler relocation operators when dealing with symbolic addresses, hopefully reduce pseudo instruction
# -mno-relax: no linker relaxations, make less use of GP which i have no idea how to handle
# -D: defines a macro to be used by the preprocessor
# -nostdlib: don't use standard libiary
# --entry main: use main function as entry point
# -fno-builtin-function: ???

GCC = riscv64-unknown-elf-g++
CFLAGS = \
        -Wall \
        -g \
        -O3 \
        -static \
        -mno-fdiv \
        -mstrict-align \
        -mlittle-endian \
        -march=rv32i \
        -mabi=ilp32 \

CFLAGS += -fno-builtin-function

OBJCOPY = riscv64-unknown-elf-objcopy
OBJDUMP = riscv64-unknown-elf-objdump
READELF = riscv64-unknown-elf-readelf
TARGET = ./riscy
PYTHON = python

all: clean assemble deassemble link asm header section offset

assemble: Seg.o Serial.o main.o printf.o ttt.o

Seg.o:Seg.cpp
        $(GCC) $^ -c $(CFLAGS) -o $@

Serial.o:Serial.cpp
        $(GCC) $^ -c $(CFLAGS) -o $@

main.o:riscy.cpp
        $(GCC) $^ -c $(CFLAGS) -o $@

printf.o:printf.c
        $(GCC) $^ -c $(CFLAGS) -o $@

ttt.o:ttt.cpp
        $(GCC) $^ -c $(CFLAGS) -o $@


deassemble: Seg.s Serial.s main.s printf.s ttt.s


Seg.s:Seg.o
        $(OBJDUMP) -D -g $^ > $@

Serial.s:Serial.o
        $(OBJDUMP) -D -g $^ > $@

main.s:main.o
        $(OBJDUMP) -D -g $^ > $@
```

```
printf.s:printf.o
	$(OBJDUMP) -D -g $^ > $@

ttt.s:ttt.o
	$(OBJDUMP) -D -g $^ > $@



link:assemble
	$(GCC) --specs=nano.specs -T elf32lriscv.ld -e _start $(CFLAGS) \
	Seg.o Serial.o main.o printf.o ttt.o \
	-o $(TARGET).elf


asm:link
	$(OBJDUMP) -D riscy.elf > $(TARGET).s


# extract elf header info from elf file
header:link
	$(READELF) -h $(TARGET).elf > $(TARGET).header

# extract section information from elf file
section:link
	$(READELF) -S $(TARGET).elf > $(TARGET).section

offset:asm header section
	$(PYTHON) ./get_off.py

# delete all but .c, .h file
clean:
	rm -f ./*.o ./*.s $(TARGET) \
	./*.bin ./*.mif ./*.mc ./*.header \
	./*.section ./*.cfg ./*.out ./*.elf
```

```
def get_entry_addr(filename="./test.header"):
	with open(filename) as file:
		lines = file.readlines()
		lines = [line.rstrip() for line in lines]
	del lines[0]
	for i in range(len(lines)):
		lines[i] = " ".join(lines[i].split())  # remove dupe white space
	header_dict = {}
	for line in lines:
		temp_line = line.split(":")
		header_dict[temp_line[0]] = temp_line[1]
	entry_addr = int(header_dict["Entry point address"], 16)
	elf_va_offset = 0x10000
	print("entry point is base : 0x10000, offset: ",(entry_addr - elf_va_offset))
	print("binary offset: {0:b}".format(entry_addr - elf_va_offset))
	fp = open("boot.cfg", 'w')
	fp.write(format(entry_addr - elf_va_offset, "x"))
	fp.close()
	return entry_addr


def get_text_addr(filename="./test.section"):
	with open(filename) as file:
		lines = file.readlines()
		lines = [line.rstrip() for line in lines]
	textline = ''
	for line in lines:
		if ".text" in line:
			textline = " ".join(line.split())  # remove dupe white space
	file.close()
	textline = textline.split()
	index = -1
	for i in range(textline.__len__()):
		if textline[i] == '.text':
			index = i
		if index >= 0 and i == index + 2:
			return int(textline[i], 16)


def get_main_addr(filename="./instr_full.s"):
	with open(filename) as file:
		lines = file.readlines()
		lines = [line.rstrip() for line in lines]
	for i in range(len(lines)):
		lines[i] = " ".join(lines[i].split())  # remove dupe white space
	for i in range(len(lines)):
		if '"<main>:"' in lines[i]:
			asm = lines[i].split()
			return int(asm[0], 16)
	print("get main addr failed")
	return -1


def get_enrty_point_offset():
	entry_addr = get_entry_addr()
	print(f"entry addr: {entry_addr}")
	text_addr = get_text_addr()
	print(f"text addr: {text_addr}")
	main_offset = entry_addr - text_addr
	print(f"boot offset: {main_offset} words")
	fp = open("boot.cfg", 'w')
	fp.write(format(main_offset, "x"))
	fp.close()


# offset fron text base
def get_main_offset_old():
	main_addr = get_main_addr()
	print(f"main_addr: {main_addr}")
	text_addr = get_text_addr(riscy.section)
```

```
        print(f"text addr: {text_addr}")
        main_offset = main_addr - text_addr
        print(f"boot offset: {int(main_offset/4)} words")
        print("main_offset: {0:b} byte".format(main_offset))
        fp = open("boot.cfg", 'w')
        fp.write(format(main_offset, "x"))
        fp.close()


# offset from start of ELF file in
def get_main_offset():
        base_addr = 0
        print(f"base addr: 0x{hex(base_addr)}")

        main_addr = get_main_addr("riscy.s")
        print(f"main_addr: 0x{hex(main_addr)}")

        main_offset = main_addr - base_addr

        print(f"main offset: {int(main_offset >> 2)} words")
        print("main offset: {0:b} words".format(main_offset >> 2))

        entry_addr = get_entry_addr("riscy.header")
        print(f"entry addr: {hex(entry_addr)}")

        entry_offset = entry_addr - base_addr
        print("entry_offset: {0:b}".format(entry_offset >> 2))

        fp = open("boot.cfg", 'w')
        fp.write(format(entry_offset >> 2, "x"))
        fp.close()



if __name__ == '__main__':
        get_main_offset()
```

```
MEMORY
{
	/*BRAM(rx):ORIGIN=0x08000000,LENGTH=512K*/
	SDRAM(rwx):ORIGIN =0x00000000,LENGTH =64M
}

/*
reference 1: https://youtu.be/B7oKdUvRhQQ
reference 2: https://github.com/pulp-platform/pulp-riscv-gnu-toolchain/blob/master/riscv.ld
*/

/*=====================================================================*/
/* Default maven linker script                                         */
/*=====================================================================*/
/* This is the default linker script for maven. It is based off of the
   mips idt32.ld linker script. I have added many more comments and
   tried to clean things up a bit. For more information about standard
   MIPS sections see Section 9.5 of "See MIPS Run Linux" by Dominic
   Sweetman. For more generic information about the init, fini, ctors,
   and dtors sections see the paper titled "ELF From the Programmers
   Perspective" by Hongiu Lu. */

/*---------------------------------------------------------------------*/
/* Setup                                                               */
/*---------------------------------------------------------------------*/

/* The OUTPUT_ARCH command specifies the machine architecture where the
   argument is one of the names used in the BFD library. More
   specifically one of the entires in bfd/cpu-mips.c */

OUTPUT_ARCH( "riscv" )

/* The ENTRY command specifies the entry point (ie. first instruction to
   execute). The symbol _start is defined in crt0.S */

ENTRY( _riscy_start )

/* The GROUP command is special since the listed archives will be
   searched repeatedly until there are no new undefined references. We
   need this since -lc depends on -lgloss and -lgloss depends on -lc. I
   thought gcc would automatically include -lgcc when needed, but
   idt32.ld includes it explicitly here and I was seeing link errors
   without it. */

GROUP( -lc -lgloss -lgcc )

/*---------------------------------------------------------------------*/
/* Sections                                                            */
/*---------------------------------------------------------------------*/
/* This is where we specify how the input sections map to output
	sections. The .= commands set the location counter, and the
	sections are inserted in increasing address order according to the
	location counter. The following statement will take all of the .bar
	input sections and reloate them into the .foo output section which
	starts at address 0x1000.

		. = 0.x1000;
		.foo : { *(.bar) }

	If we wrap an input specification with a KEEP command then it
	prevents it from being eliminted during "link-time garbage
	collection". I'm not sure what this is, so I just followed what was
	done in idt32.ld.

	We can also set a global external symbol to a specific address in the
	output binary with this syntax:

		_etext = .;
```

```
                PROVIDE( etext = . );

        This will set the global symbol _ftext to the current location. If we
        wrap this in a PROVIDE commad, the symbol will only be set if it is
        not defined. We do this with symbols which don't begin with an
        underscore since technically in ansi C someone might have a function
        with the same name (eg. etext).

        If we need to label the beginning of a section we need to make sure
        that the linker doesn't insert an orphan section inbetween where we
        set the symbol and the actual begining of the section. We can do that
        by assigning the location dot to itself.

                . = .
                _ftext = .;
                .text :
                { }

        */

SECTIONS
{
        /*-----------------------------------------------------------------------*/
        /* Code and read-only segment                                            */
        /*-----------------------------------------------------------------------*/

        /* Begining of code and text segment */
        . = 0x00000000;
        _ftext = .;
        PROVIDE( eprol = . );

        /* text: Program code section */
        .text :
        {
                *(.text)      /*wild card, merge all text section*/
                *(.text.*)
                *(.gnu.linkonce.t.*)
        }

        /* init: Code to execute before main (called by crt0.S) */
        .init :
        {
                KEEP( *(.init) )
        }

        /* fini: Code to execute after main (called by crt0.S) */
        .fini :
        {
                KEEP( *(.fini) )
        }

        /* rodata: Read-only data */
        .rodata :
        {
                *(.rdata)
                *(.rodata)
                *(.rodata.*)
                *(.gnu.linkonce.r.*)
        }

        /* End of code and read-only segment */
        PROVIDE( etext = . );
        _etext = .;

        /*-----------------------------------------------------------------------*/
        /* Global constructor/destructor segement                                */
        /*-----------------------------------------------------------------------*/
```

```
.preinit_array     :
{
    PROVIDE_HIDDEN (__preinit_array_start = .);
    KEEP (*(.preinit_array))
    PROVIDE_HIDDEN (__preinit_array_end = .);
}

.init_array     :
{
    PROVIDE_HIDDEN (__init_array_start = .);
    KEEP (*(SORT(.init_array.*)))
    KEEP (*(.init_array ))
    PROVIDE_HIDDEN (__init_array_end = .);
}

.fini_array     :
{
    PROVIDE_HIDDEN (__fini_array_start = .);
    KEEP (*(SORT(.fini_array.*)))
    KEEP (*(.fini_array ))
    PROVIDE_HIDDEN (__fini_array_end = .);
}

/*-------------------------------------------------------------------*/
/* Other misc gcc segments (this was in idt32.ld)
    */
/*-------------------------------------------------------------------*/
/* I am not quite sure about these sections but it seems they are for
    C++ exception handling. I think .jcr is for "Java Class
    Registration" but it seems to end up in C++ binaries as well. */

.eh_frame_hdr     : { *(.eh_frame_hdr)     }
.eh_frame     : { KEEP( *(.eh_frame) ) }
.gcc_except_table : { *(.gcc_except_table) }
.jcr     : { KEEP (*(.jcr))     }

/*-------------------------------------------------------------------*/
/* Initialized data segment
                                                                    */
/*-------------------------------------------------------------------*/

/* Start of initialized data segment */
. = ALIGN(16);
 _fdata = .;

/* data: Writable data */
.data :
{
    *(.data)
    *(.data.*)
    *(.gnu.linkonce.d.*)
}

/* End of initialized data segment */
PROVIDE( edata = . );
_edata = .;

/* Have _gp point to middle of sdata/sbss to maximize displacement range */
. = ALIGN(16);
/*_gp = . + 0x800;*/
__global_pointer$ = . + 0x800;

/* Writable small data segment */
.sdata :
{
    *(.sdata)
    *(.sdata.*)
    *(.srodata.*)
    *(.gnu.linkonce.s.*)
```

```
	}

	/*--------------------------------------------------------------------*/
	/* Uninitialized data segment                                         */
	/*--------------------------------------------------------------------*/

	/* Start of uninitialized data segment */
	. = ALIGN(8);
	_fbss = .;

	/* Writable uninitialized small data segment */
	.sbss :
	{
		*(.sbss)
		*(.sbss.*)
		*(.gnu.linkonce.sb.*)
	}

	/* bss: Uninitialized writeable data section */
	. = .;
	_bss_start = .;
	.bss :
	{
		*(.bss)
		*(.bss.*)
		*(.gnu.linkonce.b.*)
		*(COMMON)
	}

	/* End of uninitialized data segment (used by syscalls.c for heap) */
	PROVIDE( end = . );
	_end = ALIGN(8);
}
```

```
#include "Seg.hpp"


Seg::Seg() {
	for (int i = 0; i < 6; i++) {
		this->digit[i] = i;
	}
}


Seg::~Seg() {}


void Seg::write_seg(uint32_t* addr, uint32_t value) {
	*addr = value;
	return;
}


void Seg::set_seg_single (int index, int number) {
	uint32_t* seg_pa = &((uint32_t*)SEG_BASE)[index];
	uint32_t hex_seg_digit = number;
	write_seg(seg_pa, hex_seg_digit);
	return;
	// *(uint32_t*)seg_pa = hex_seg_digit;
}


void Seg::set_seg_digit (int index, int value) {
	this->digit[index] = value & SEG_DATA_MASK;
	return;
}


void Seg::set_seg(int value) {
	for (int i = 0; i < 6; i++) {
		this->digit[i] = value & SEG_DATA_MASK;
		value >>= 4;
	}
	return;
}


void Seg::write_seg() {
	for (int i = 0; i < 6; i++) {
		set_seg_single(i, digit[i]);
	}
	return;
}
```

```
#include "Serial.hpp"

Serial::Serial() {}

Serial::~Serial() {}

volatile uint32_t Serial::tx_char = '\0';
volatile uint32_t Serial::rx_char = '\0';
volatile uint32_t Serial::rx_buf_len = 0;

void Serial::putc(char c){
	uint32_t char_int_32 = ((uint32_t)(c)) & UART_DATA_MASK;
	tx_char = char_int_32;
	*((uint32_t*)UART_DATA_ADDR) = tx_char;
	tx_char = 0;
}


void Serial::write_string(const char* c, int l) {
	int i;
	for (i = 0; i < l; i++){
		putc(c[i]);
	}
	return;
}


uint32_t Serial::read_strlen() {
	rx_buf_len = *((uint32_t*)UART_DLEN_ADDR);
	return rx_buf_len;
}


char Serial::read_char() {
	rx_char = ((char)(*((uint32_t*)UART_DATA_ADDR))) & UART_DATA_MASK;
	return rx_char;
}


void Serial::print(const char* str) {
	int l = 0;
	while (str[l] != '\0') {
		l++;
	}
	write_string(str, l);
	return;
}


void Serial::println(const char* str) {
	print(str);
	print(linefeedstr);
	return;
}


char Serial::pull_input() {
	int available = 0;
	while (1) {
		available = read_strlen();
		if (available > 0) {
			break;
		}
	}
	char c = read_char() & UART_DATA_MASK;
	return c;
}


int Serial::char2int(char c) {
```

```
	switch (c)
	{
		case '0':
			return 0;
		break;

		case '1':
			return 1;
		break;

		case '2':
			return 2;
		break;

		case '3':
			return 3;
		break;

		case '4':
			return 4;
		break;

		case '5':
			return 5;
		break;

		case '6':
			return 6;
		break;

		case '7':
			return 7;
		break;

		case '8':
			return 8;
		break;

		case '9':
			return 9;
		break;

		default:
			return -1;
	}
}
```

```
#include "ttt.hpp"
#include "Serial.hpp"

Serial* myserial;

void ttt() {
   myserial = new Serial();
   char board[row][col];
   game(board);
}

int game(char board[row][col]) {
   int rowInput;
   int colInput;
   setBoard(board);
   printBoard(board);

   // game loop
   while(checkWinner(board) == 0) {

      getInput(board, &rowInput, &colInput, 'X');
      printBoard(board);
      if (checkWinner(board) == 1) {
         printf_("X Wins!\r\n");
         break;
      }

      getInput(board, &rowInput, &colInput, 'O');
      //bot(board);
      printBoard(board);
      if (checkWinner(board) == -1) {
         printf_("O Wins!\r\n");
         break;
      }
   }

   return 0;
}

void printBoard(char board[row][col]) {
   // print board
   printf_("\r\n");
   for (int i = 0; i < row; i++) {
      for (int j = 0; j < col; j++) {
         printf_("%c",board[i][j]);
      }
      printf_("\r\n");
   }
}

void setBoard(char board[row][col]) {
   // reset board
   for (int i = 0; i < row; i++) {
      for (int j = 0; j < col; j++) {
         board[i][j] = '0';
      }
   }
}

void getInput(char board[row][col], int *rowInput, int *colInput, char team) {
   // get input from user (row and column)

   printf_("Team %c's turn\r\n", team);

   printf_("Enter row: ");
   char c = myserial->pull_input();
   int n = myserial->char2int(c);
   *rowInput = n;
   printf_("\r\n");
```

```
    printf_("Enter column: ");
    c = myserial->pull_input();
    n = myserial->char2int(c);
    *colInput = n;
    printf_("\r\n");

    if(checkAvailable(board, *rowInput, *colInput) == -1) {
        getInput(board, rowInput, colInput, team);
    }
    else {
        board[*rowInput][*colInput] = team;
    }
}

int checkAvailable(char board[row][col], int rowInput, int colInput) {
    // check if square is available
    if (board[rowInput][colInput] == '0') {return 1;}
    else {return -1;}

    return 0;
}

int checkWinner(char board[row][col]) {
    // check if there has been a winner

    // rows
    for (int i = 0; i < row; i++) {
        if (board[i][0] == 'X' && board[i][1] == 'X' && board[i][2] == 'X') {return 1;}
        if (board[i][0] == 'O' && board[i][1] == 'O' && board[i][2] == 'O') {return -1;}
    }

    // columns
    for (int i = 0; i < row; i++) {
        if (board[0][i] == 'X' && board[1][i] == 'X' && board[2][i] == 'X') {return 1;}
        if (board[0][i] == 'O' && board[1][i] == 'O' && board[2][i] == 'O') {return -1;}
    }

    // diagonals
    if (board[0][0] == 'X' && board[1][1] == 'X' && board[2][2] == 'X') {return 1;}
    if (board[0][0] == 'O' && board[1][1] == 'O' && board[2][2] == 'O') {return -1;}
    if (board[2][0] == 'X' && board[1][1] == 'X' && board[0][2] == 'X') {return 1;}
    if (board[2][0] == 'O' && board[1][1] == 'O' && board[0][2] == 'O') {return -1;}

    return 0;
}
```

```
#include "riscy.hpp"
#include "ttt.hpp"

int main() asm ("main");

int main() {
	_riscy_start();

	printf_("Hello Riscy \r\n");

	sanity_test_seg();
	sanity_test_serial();

	Seg* dbg_seg = new Seg;

	ttt();

	dbg_seg->set_seg(0xabcdef);
	dbg_seg->write_seg();


	Serial* s = new Serial;
	int buflen = 0;
	char c;
	for(;;) {
		buflen = s->read_strlen();
		if (buflen > 0) {
			c = s->read_char();
			s->putc(c);
		}
	}


	halt_riscy();
}


void _riscy_start() {
	//init sp
	__asm__("li sp, 0x03fffffc");
}


void sanity_test_seg() {
	Seg* seg = new Seg();
	seg->set_seg(0xabcdef);
	seg->write_seg();
	delete seg;
}


void sanity_test_serial() {
	const char* hellostr = "0123456789abcdef\r\n";
	Serial* serial = new Serial();
	serial->print(hellostr);
	delete serial;
}


void halt_riscy() {
	__asm__("li a0,42");
	__asm__("li a7,93");
	__asm__("ebreak");
	return;
}
```