VLSI System Design (Graduate Level)
Fall 2022

HOMEWORK III

Due day: 2:00pm Nov. 30 (Wednesday), 2022

Introduction

In this homework, you need to implement L1 instruction cache and L1 data cache. You also need to write a simple program to booting your CPU. Finally, complete synthesis of your system, which include CPU, CPU wrapper, L1 instruction cache, L1 data cache, AXI, IM, DM, ROM Wrapper and DRAM Wrapper.

General rules for deliverables

- This homework can be completed by INDIVIDUAL student or a TEAM (up to 2 students). Only one submission is needed for a team. You MUST write down you and your teammate's name on the submission cover of the report. Otherwise duplication of other people's work may be considered cheating.
- Compress all files described in the problem statements into one tar file.
- Submit the compressed file to the course website before the due day.
  Warning! AVOID submitting in the last minute. Late submission is not accepted.

Grading Notes

- **Important!** DO remember to include your SystemVerilog code. NO code, NO grades. Also, if your code can not be recompiled by TA successfully using tools in SoC Lab and commands in Appendix B, you will receive NO credit.
- Write your report seriously and professionally. Incomplete description and information will reduce your chances to get more credits.
- DO read the homework statements and requirements thoroughly. If you fail to comply, you are not able to get full credits.
- Please follow course policy.
- Verilog and SystemVerilog generators aren't allowed in this course.

Deliverables

1. All SystemVerilog codes including components, testbenches and machine codes for each lab exercise. NOTE: Please DO NOT include source codes in the report!
2. Write a homework report in MS word and follow the convention for the file name

of your report: N260xxxxx.docx. Please save as docx file format and replace N260xxxxx with your student ID number. (Let the letter be uppercase.) If you are a team, you should name your report, top folder and compressed file with the student ID number of the person uploading the file. The other should be written on the submission cover of your report, or you will receive NO credit.

3. Specified percentage of contributions by every team member. For example, contributions by everyone are equally divided, you can specified Axx 50%, and Byy 50%.
4. Organize your files as the hierarchy in Appendix A.

Report Writing Format

a. Use the submission cover which is already in provided *N260XXXXX.docx*.
b. A summary in the beginning to state what has been done.
c. Report requirements from each problem.
d. Describe the major problems you encountered and your resolutions.
e. Lessons learned from this homework.

Problem1

1.1 Problem Description

Caches can reduce data latency caused by bus transmission. In this problem, you need to implement a data cache module named “L1C_data” and an instruction cache module named “L1C_inst”. Detailed descriptions of this problem can be found in Section 1.4.

1.2 Module Specification

Inputs and outputs of module L1C_data and L1C_inst has declared in src/L1C_data.sv and src/L1C_inst.sv. You can modify input and output bit width if needed. If you do so, please modify bit width in /sva/cache_props.sva

Table 1-1: Module signals

Module	Specifications			
L1C_inst	clk	input	1	clock
	rst	input	1	reset (active high)
	core_addr	input	32	address from CPU
	core_req	input	1	memory access request from CPU
	core_write	input	1	write signal from CPU
	core_in	input	32	data from CPU
	core_type	input	3	write/read byte, half word , or word (listed in include/def.svh) from CPU
	I_out	input	32	data from CPU wrapper
	I_wait	input	1	wait signal from CPU wrapper
	core_out	output	32	data to CPU
	core_wait	output	1	wait signal to CPU
	I_req	output	1	request to CPU wrapper
	I_addr	output	32	address to CPU wrapper
	I_write	output	1	write signal to CPU wrapper
	I_in	output	32	write data to CPU wrapper
	I_type	output	3	write/read byte, half word, or word (listed in include/def.svh)
	valid	logic	64	valid bits of each tag
	index	logic	6	address to array
	TA_write	logic	1	write signal to tag_array
	TA_read	logic	1	read signal to tag_array
	TA_in	logic	22	write data to tag_array

	TA_out	logic	22	read data from tag_array
	DA_write	logic	16	write data to data_array
	DA_read	logic	1	read signal to data_array
	DA_in	logic	128	write data to data_array
	DA_out	logic	128	read data from data_array

Module	**Specifications**			
L1C_data	clk	input	1	clock
	rst	input	1	reset (active high)
	core_addr	input	32	address from CPU
	core_req	input	1	memory access request from CPU
	core_write	input	1	write signal from CPU
	core_in	input	32	data from CPU
	core_type	input	3	write/read byte, half word , or word (listed in include/def.svh) from CPU
	D_out	input	32	data from CPU wrapper
	D_wait	input	1	wait signal from CPU wrapper
	core_out	output	32	data to CPU
	core_wait	output	1	wait signal to CPU
	D_req	output	1	request to CPU wrapper
	D_addr	output	32	address to CPU wrapper
	D_write	output	1	write signal to CPU wrapper
	D_in	output	32	write data to CPU wrapper
	D_type	output	3	write/read byte, half word, or word (listed in include/def.svh)
	valid	logic	64	valid bits of each tag
	index	logic	6	address to array
	TA_write	logic	1	write signal to tag_array
	TA_read	logic	1	read signal to tag_array
	TA_in	logic	22	write data to tag_array
	TA_out	logic	22	read data from tag_array
	DA_write	logic	16	write data to data_array
	DA_read	logic	1	read signal to data_array
	DA_in	logic	128	write data to data_array
	DA_out	logic	128	read data from data_array

1.3 Detailed Description

The features of cache are specified in Table 1-2. Actions of cache depend on different conditions are listed in Table 1-3. Characteristics of data_array and tag_array are listed in Table 1-4. “Byte Write” means you can use the bit(s) of WEB to select which byte(s) to write. “Word Write” means you can only write entire word.

Table 1-2: Caches specification

Type	**Size**	**Line Bits**	**Associativity**	**Write Policy**	
Cache	1 KB	128	Direct Map	Hit	write through
				miss	write around

Table 1-3: Cache actions on different condition

condition	**core read**	**core write**
hit	transmit data into core	write data into cache and memory
miss	read a line from memory	only write data into memory

Table 1-4: Array characteristics

Type	**Lines**	**Words per line**	**Bytes per word**	**Bits per line**	**Writing mode**
data_array	64	4	4	128	Byte Write
tag_array		1		22	Word Write

1.4 Verification

You should show the proven results of JasperGold ABVIP. If there are unproven properties or counter examples, please explain with waveforms clearly in your report.

1.5 Report Requirements

Your report should have following features:

a. Proper explanation of your design is required for full credits.
b. Block diagrams shall be drawn to depict your designs.
c. Show the result of JasperGold with full prove, explain the assertion you modified.
d. Show the hit rate of your instruction cache and data cache
e. Show the IPC (instruction per cycle) of your CPU

$$\text{cache hit rate} = \frac{Number\ of\ cache\ hits}{Number\ of\ cache\ hits + Number\ of\ cache\ misses}$$

$$\text{IPC} = \frac{Number\ of\ instructions\ CPU\ execute}{Total\ cycles}$$

Problem2

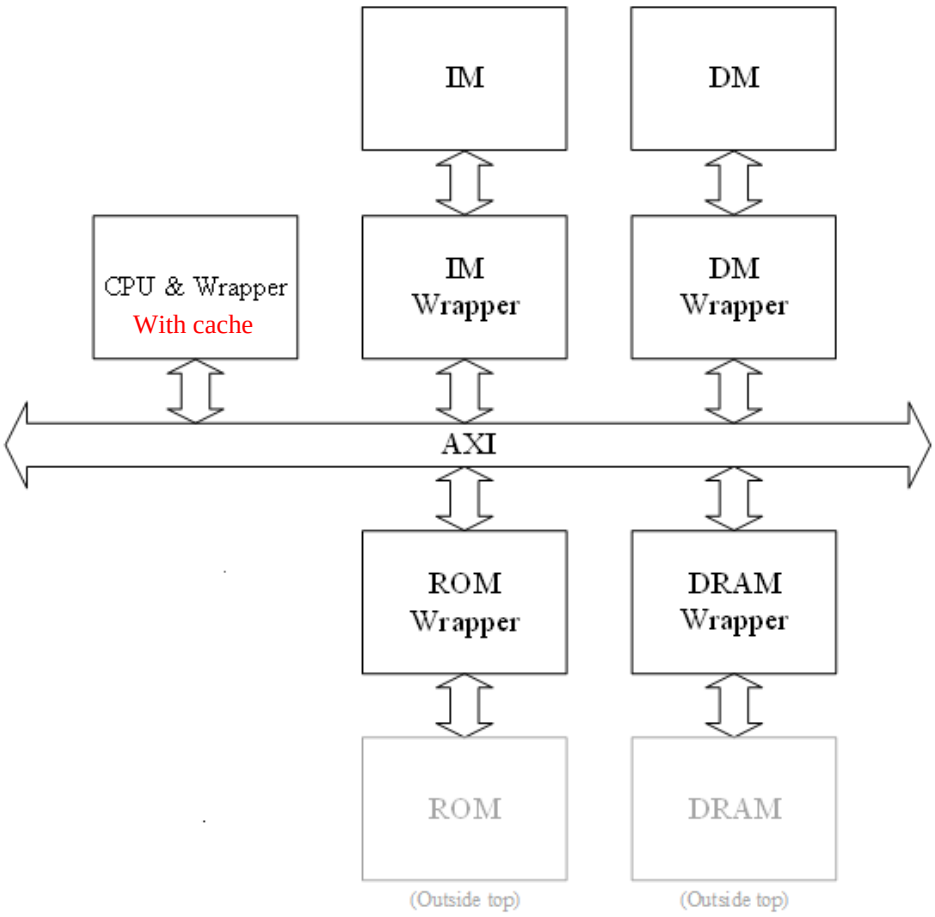
2.1 Problem Description

In this problem, you have to support booting operations for your RISC-V CPU from HOMEWORK II, complete synthesis of your system. Your CPU should have following new features:

a. Support booting operation.
b. Instruction cache size: 64×128-bit.
c. Data cache size: 64×128-bit.

A more detailed description of this problem can be found in Section 1.4.

2.2 Block Overview



2.3 Module Specification

Table 2-1: Module naming rule

Category	Name			
	File	Module	Instance	SDF
RTL	top.sv	top	TOP	
Gate-Level	top_syn.v	top	TOP	top_syn.sdf
RTL	L1C_inst.sv	L1C_inst	L1CI	
RTL	L1C_data.sv	L1C_data	L1CD	
RTL	SRAM_wrapper.sv	SRAM_ wrapper	IM1	
RTL	SRAM_wrapper.sv	SRAM_ wrapper	DM1	
RTL	SRAM_rtl.sv	SRAM	i_SRAM	
RTL	tag_array_wrapper.sv	tag_array_ wrapper	TA	
RTL	tag_array_rtl.sv	tag_array	i_tag_array	
RTL	data_array_wrapper.sv	data_array_ wrapper	DA	
RTL	data_array_rtl.sv	data_array	i_data_array	
Behavior	ROM.v	ROM	i_ROM	
Behavior	DRAM.v	DRAM	i_DRAM	

Table 2-2: Module signals

Module	Specifications			
top	Name	Signal	Bits	Function explanation
	System signals			
	clk	input	1	System clock
	rst	input	1	System reset (active high)
	Connect with ROM			
	ROM_out	input	32	Data from ROM
	ROM_read	output	1	ROM output enable
	ROM_enable	output	1	Enable ROM
	ROM_address	output	12	Address to ROM

top	**Name**	**Signal**	**Bits**	**Function explanation**
	Connect with DRAM			
	DRAM_Q	input	32	Data from DRAM
	DRAM_valid	input	1	DRAM output data valid
	DRAM_CSn	output	1	DRAM Chip Select (active low)
	DRAM_WEn	output	4	DRAM Write Enable (active low)
	DRAM_RASn	output	1	DRAM Row Access Strobe (active low)
	DRAM_CASn	output	1	DRAM Column Access Strobe (active low)
	DRAM_A	output	11	Address to DRAM
	DRAM_D	output	32	Data to DRAM
ROM	System signals			
	CK	input	1	System clock
	Memory ports			
	DO	output	32	ROM data output
	OE	input	1	Output enable (active high)
	CS	input	1	Chip select (active high)
	A	input	12	ROM address input
	Memory Space			
	Memory_byte0	reg	8	Size: [0:4095]
	Memory_byte1	reg	8	Size: [0:4095]
	Memory_byte2	reg	8	Size: [0:4095]
	Memory_byte3	reg	8	Size: [0:4095]
DRAM	System signals			
	CK	input	1	System clock
	RST	input	1	System reset (active high)
	Memory ports			
	CSn	input	1	DRAM Chip Select (active low)
	WEn	input	4	DRAM Write Enable (active low)
	RASn	input	1	DRAM Row Access

				Strobe(active low)
	CASn	input	1	DRAM Column Access Strobe (active low)
	A	input	11	DRAM Address input
	D	input	32	DRAM data input
	Q	output	32	DRAM data output
	VALID	output	1	DRAM data output valid
	Memory space			
	Memory_byte0	reg	8	Size: [0:2097151]
	Memory_byte1	reg	8	Size: [0: 2097151]
	Memory_byte2	reg	8	Size: [0: 2097151]
	Memory_byte3	reg	8	Size: [0: 2097151]

2.4 Detailed Description

Slave configuration is listed in Table 2-3, you should follow the specification. You should design slave wrappers by yourself. You should only implement read operation in ROM wrapper and implement read and write operations in IM, DM and DRAM wrapper.

You SHOULD use the timing constraint file, *DC.sdc*, provided in the course website to synthesize your top.sv. Don't modify any constraint except clock period.

Plus, Your RTL code needs to comply with Superlint within 95% of your code, i.e., the number of errors & warnings in total shall not exceed 5% of the number of lines in your code. HINT: You can use the command in Appendix B to get the number of lines in your code. Remember to exclude *top_tb.sv*.

Table 2-3: Slave configuration

Name	Number	Start address	End address
ROM	Slave 0	0x0000_0000	0x0000_1FFF
IM	Slave 1	0x0001_0000	0x0001_FFFF
DM	Slave 2	0x0002_0000	0x0002_FFFF
DRAM	Slave 4	0x2000_0000	0x201F_FFFF

2.5 Verification

You should use the commands in Appendix B to verify your design.

a. For prog0, prog1 and prog2, you should write a boot program defined as boot.c to copy data between _dram_i_start and _dram_i_end to _imem_start, from __data_paddr_start to __data_start and __data_end, also from __sdata_paddr_start to __sdata_start and __sdata_end. The booting program should be stored at ROM.
b. Use *prog0* to perform verification for the functionality of instructions.
c. Write a program defined as *prog1* to perform a sort algorithm. The number of sorting elements is stored at the address named *array_size* in ".rodata" section defined in *data.S*. The first element is stored at the address named *array_addr* in ".rodata" section defined in *data.S*, others are stored at adjacent addresses. The maximum number of elements is 128. All elements are **signed 2-byte half-word** and you should sort them in **ascending order**. Rearranged data should be stored at the address named *_test_start* in "_test" section defined in *link.ld*.
d. Write a program defined as prog2 to turn image into gray scale. Image data is stored byte(8 bits) by byte in order of {blue, green, red} from address named "_binary_image_bmp_start" to "_binary_image_bmp_end". The number of bytes is stored at "_binary_image_bmp_size". Store gray scale result byte by byte from "_test_start" to "test_start"+_binary_image_bmp_size-1. The gray scale formula is y = 0.11*blue + 0.59*green + 0.3*red
e. Write a program defined as *prog3* to perform the matrix multiplication. The row size & column size of matrix is stored at the address named *array_size_i*, *array_size_j* and *array_size_k* in ".rodata" section defined in *data.S*. The first element is stored at the address named *array_addr* in ".rodata" section defined in *data.S*, others are stored at adjacent addresses. All elements in matrix are **signed 2-byte half-word** and you should store result byte by byte from "_test_start" to "test_start"+ array_size_i*array_size_j-1

Don't forget to return from main function to finish the simulation in each program. Save your assembly code or C code as main.S or main.c respectively. You should also explain the result of this program in the report. In addition to these verification, TA will use another program to verify your design. Please make sure that your design can execute the listed instructions correctly.

2.6 Report Requirements

Your report should have the following features:

f. Proper explanation of your design is required for full credits.

g. Block diagrams shall be drawn to depict your designs.

h. Show your screenshots of the waveforms and the simulation results on the terminal for the different test cases in your report and illustrate the correctness of your results. Explain cache read hit/read miss/write hit/write miss with waveform.

i. Explain your codes of program1, program2 and program3.

j. Explain your codes of boot.c.

k. Report the number of lines of your RTL code, the final results of running Superlint and 3~5 most frequent warning/errors in your code. Describe how you modify your code to comply with Superlint.

l. Report and show screenshots of your prog0 to prog3 simulation time after synthesis and total cell area of your design.

Appendix

A. File Hierarchy Requirements

All homework SHOULD be uploaded and follow the file hierarchy and the naming rules, especially letter case, specified below. You should create a main folder named your student ID number. It contains your homework report and other files. The names of the files and the folders are labeled in red color, and the specifications are labeled in black color. Filenames with * suffix in the same folder indicate that you should provide one of them. Before you submit your homework, you can use Makefile macros in Appendix B to check correctness of the file structure.

Fig. A-1 File hierarchy

- *N260XXXXX.tar* (**Don't** add version text in filename, e.g. *N260XXXXX_v1.tar*)
  - *N260XXXXX* (Main folder of this homework)
    - *N260XXXXX.docx* (Your homework report)
    - *StudentID* (Specify your student ID number in this file)
    - *StudentID2* (Specify your partner's student ID number in this file. Please delete it if you don't have partner)
    - *Makefile* (You shouldn't modify it)
    - *src* (Your RTL code with *sv* format)
      - *top.sv*
      - *L1C_inst.sv*
      - *L1C_data.sv*
      - *SRAM_wrapper.sv*
      - *tag_array_wrapper.sv*
      - *data_array_wrapper.sv*
      - ROM_wrapper.sv
      - DRAM_wrapper.sv
      - Other submodules (**.sv*)
      - *AXI*
        - *AXI.sv*
        - Submodules of AXI (**.sv*)
    - *include* (Your RTL definition with *svh* format)
      - *AXI_def.svh*
      - *def.svh*
      - Definition files (**.svh*)
    - *syn* (Your synthesized code and timing file)

- *top_syn.v*
- *top_syn.sdf*

- *script* (Any scripts of verification, synthesis or place and route)
  - script files (*.*sdc*, *.*tcl* or *.*setup*)
- *sim* (Testbenches and memory libraries)
  - *top_tb.sv* (Main testbench. You shouldn't modify it)
  - *CYCLE* (Specify your clock cycle time in this file)
  - *MAX* (Specify max clock cycle number in this file)
  - *SRAM* (SRAM libraries and behavior models)
    - Library files (*.*lib*, *.*db*, *.*lef* or *.*gds*)
    - *SRAM.ds* (SRAM datasheet)
    - *SRAM_rtl.sv* (SRAM RTL model)
    - *SRAM.v* (SRAM behavior model)
  - *ROM* (ROM behavior models)
    - *ROM.v* (ROM behavior model)
  - *DRAM* (DRAM behavior models)
    - *DRAM.v* (DRAM behavior model)
  - *data_array* (data_array libraries and behavior models)
    - Library files (*.*lib*, *.*db*, *.*lef* or *.*gds*)
    - *data_array.ds* (data_array datasheet)
    - *data_array_rtl.sv* (data_array RTL model)
    - *data_array.v* (data_array behavior model)
  - *tag_array* (tag_array libraries and behavior models)
    - Library files (*.*lib*, *.*db*, *.*lef* or *.*gds*)
    - *tag_array.ds* (tag_array datasheet)
    - *tag_array_rtl.sv* (tag_array RTL model)
    - *tag_array.v* (tag_array behavior model)
  - *prog0* (Subfolder for Program 0)
    - *Makefile* (Compile and generate memory content)
    - *boot.c** (C code for verification)
    - *main.S* (Assembly code for verification)
    - *setup.S* (Assembly code for testing environment setup)
    - *link.ld* (Linker script for testing environment)
    - *golden.hex* (Golden hexadecimal data)
  - *prog1* (Subfolder for Program 1)
    - *Makefile* (Compile and generate memory content)
    - *boot.c** (C code for verification)

- *main.S* * (Assembly code for verification)
- *main.c* * (C code for verification)
- *data.S* (Assembly code for testing data)
- *setup.S* (Assembly code for testing environment setup)
- *link.ld* (Linker script for testing environment)
- *golden.hex* (Golden hexadecimal data)

prog2 (Subfolder for Program 2)

- *Makefile* (Compile and generate memory content)
- *boot.c** (C code for verification)
- *main.S* * (Assembly code for verification)
- *main.c* * (C code for verification)
- *setup.S* (Assembly code for testing environment setup)
- *link.ld* (Linker script for testing environment)
- *image.bmp* (bmp file)
- *golden.hex* (Golden hexadecimal data)

prog3 (Subfolder for Program 3)

- *Makefile* (Compile and generate memory content)
- *boot.c** (C code for verification)
- *main.S* * (Assembly code for verification)
- *main.c* * (C code for verification)
- *data.S* (Assembly code for testing data)
- *setup.S* (Assembly code for testing environment setup)
- *link.ld* (Linker script for testing environment)
- *golden.hex* (Golden hexadecimal data)

B. Simulation Setting Requirements

You **SHOULD** make sure that your code can be simulated with specified commands in Table B-1. **TA will use the same command to check your design under SoC Lab environment. If your code can't be recompiled by TA successfully, you receive NO credit.**

Table B-1: Simulation commands

Simulation Level	Command
Problem2	
RTL	make rtl_all
Gate-level	make syn_all

TA also provide some useful Makefile macros listed in Table B-2. Braces {} means that you can choose one of items in the braces. X stands for 0,1,2,3…, depend on which verification program is selected.

Table B-2: Makefile macros

Situation	Command
RTL simulation for progX	make rtlX
Post-synthesis simulation for progX	make synX
Dump waveform (no array)	make {rtlX,synX, prX} FSDB=1
Dump waveform (with array)	make {rtlX,synX, prX} FSDB=2
Open nWave without file pollution	make nWave
Open Superlint without file pollution	make superlint
Run JasperGold VIP on AXI bridge without file pollution (RTL only)	make vip_b
Open DesignVision without file pollution	make dv
Synthesize your RTL code (You need write *synthesis.tcl* in *script* folder by yourself)	make synthesize
Delete built files for simulation, synthesis or verification	make clean
Check correctness of your file structure	make check
Compress your homework to *tar* format	make tar

You can use the following command to get the number of lines:

```
wc -l src/* src/AXI/* include/*
```

C. RISC-V Instruction Format

Table C-1: Instruction type

☞ R-type

31 25	24 20	19 15	14 12	11 7	6 0
funct7	rs2	rs1	funct3	rd	opcode

☞ I-type

31 20	19 15	14 12	11 7	6 0
imm[31:20]	rs1	funct3	rd	opcode

☞ S-type

31 25	24 20	19 15	14 12	11 7	6 0
imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode

HOMEWORK III

☞ B-type

31	30 25	24 20	19 15	14 12	11 8	7	6 0
imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode

☞ U-type

31 12	11 7	6 0
imm[31:12]	rd	opcode

☞ J-type

31	30 21	20	19 12	11 7	6 0
imm[20]	imm[10:1]	imm[11]	imm[19:12]	rd	opcode

Table C-2: Immediate type

☞ I-immediate

31 11	10 5	4 1	0
─ inst[31] ─	inst[30:25]	inst[24:21]	inst[20]

☞ S-immediate

31 11	10 5	4 1	0
─ inst[31] ─	inst[30:25]	inst[11:8]	inst[7]

☞ B-immediate

31 12	11	10 5	4 1	0
─ inst[31] ─	inst[7]	inst[30:25]	inst[11:8]	0

☞ U-immediate

31	30 20	19 12	11 0
Inst[31]	inst[30:20]	inst[19:12]	─ 0 ─

☞ J-immediate

31 20	19 12	11	10 5	4 1	0
─ inst[31] ─	inst[19:12]	inst[20]	inst[30:25]	inst[24:21]	0

"─ X ─" indicates that all the bits in this range is filled with X.