

HOMEWORK III

Due day: 9:00am Nov. 29 (Thursday), 2018

Introduction

In this homework, you need to implement L1 instruction cache and L1 data cache inside your CPU. Furthermore, you need add new instructions in your CPU. Finally, complete synthesis and automatic place and route (APR) of your system, which include CPU, CPU wrapper, L1 instruction cache, L1 data cache, AHB, IM, and DM.

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

HOMEWORK III

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.

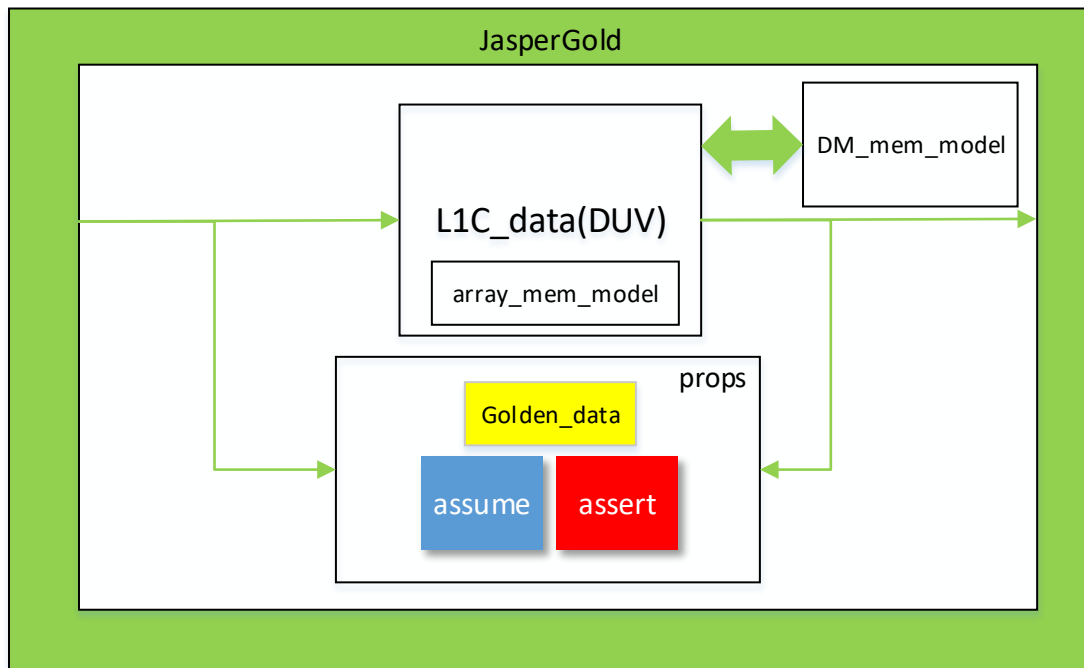
HOMEWORK III

Problem1(30/100)

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 verify its “data integrity” and “hit”, “miss” occasions by JasperGold. Detailed descriptions of this problem can be found in Section 1.4.

1.2 Block Overview



1.3 Module Specification

Inputs and outputs of module L1C_data has declared in src/L1C_data.sv. **You can only add logics inside the modules.**

Table 1-1: Module signals

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

HOMEWORK III

				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
	hit	logic	1	equal 1 when valid && TA_out equal core_addr[31:10]; otherwise, equal 0
	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

HOMEWORK III

1.4 Detailed Description

The features of data 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.

On CPU/cache interface, CPU and cache should transmit **entire word** without data shift to each other through port “core_in” and “core_out” respectively, no matter which core_type is. On cache/CPU wrapper interface, cache should **shift data based on core_addr[1:0]** before transmit to CPU_wrapper through port “D_in”. By contrast, CPU wrapper should transmit **entire word** without data shift through port “D_out”, no matter which D_type is.

Table 1-2: Data caches specification

Type	Size	Line Bits	Associativity	Write Policy	
D-cache	1 KB	128	Direct Map	hit	write through
				miss	write around

Table 1-3: Data cache actions on different condition

condition	core read	core write
hit	transmit data into core	write data into D-cache and DM
miss	read a line from DM	only write data into DM

Table 1-4: Array characteristics

Type	Word	Bits per byte	Bytes	Bits per line	Writing mode
data_array	64	8	16	128	Byte Write
tag_array		22		22	Word Write

1.5 Verification

You should use the command “make jg” in Appendix B to verify your design. **Please check the assertions in sva/cache_data.sva before design LIC_data module.** You need to get the prove result of three assertions as always true, or not found any counterexample(CEX) after 1800s. **Noted, any change within sva/ directory is NOT allowed, except for new assertions in cache_data.sva for bonus.**

1.6 Report Requirements

Your report should have following features:

- Proper explanation of your design is required for full credits.
- Block diagrams shall be drawn to depict your designs.

HOMEWORK III

- c. Show the Property Table of 3 assertions (data_integrity, hit_check, and miss_check) (and bonus) with full prove or no CEX found after 1800s.
- d. If you set other assertions for bonus, explain why these assertions increase your code coverage.

HOMEWORK III

Problem2 (70/100)

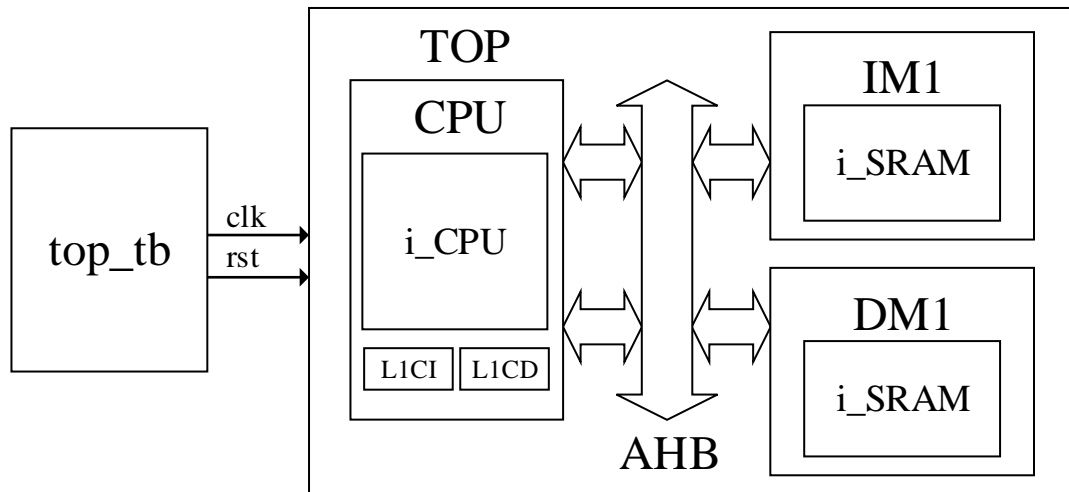
2.1 Problem Description

Add instruction cache and new instructions inside your RISC-V CPU from HOMEWORK II, complete synthesis and place and route(P&R) of your system. Your CPU should have following new features:

- The RISC-V ISA with 6 more instructions.
- Instruction cache size: 64×128-bit.
- 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

Please follow module signals specifications in HOMEWORK II. Inputs and outputs of module L1C_inst has declared in src/L1C_inst.sv. Connect L1C_inst and L1C_data to your CPU and CPU wrapper, and include “tag_array_wrapper.sv” and “data_array_wrapper.sv” in top module. Tag_array.v and data_array.v already include in top_tb.sv, so don’t re-include them in your design.

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

HOMEWORK III

Physical	top_pr.sv	top	TOP	top_pr.sdf
RTL	L1C_inst.sv	L1C_inst	L1CI	
RTL	L1C_data.sv	L1C_data	L1CD	
RTL	AHB.sv	AHB	AHB1	
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	

Table 2-2: Module signals

Module	Specifications			
	Name	Signal	Bits	Function explanation
top	clk	input	1	System clock
	rst	input	1	System reset (active high)
	Memory Space			
SRAM	Memory_byte0	logic	8	Size: [16384]
	Memory_byte1	logic	8	Size: [16384]
	Memory_byte2	logic	8	Size: [16384]
	Memory_byte3	logic	8	Size: [16384]
L1C_inst /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

HOMEWORK III

				include/def.svh) from CPU
	I_out/D_out	input	32	data from CPU wrapper
	I_wait/D_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/D_req	output	1	request to CPU wrapper
	I_addr/D_addr	output	32	address to CPU wrapper
	I_write/ D_write	output	1	write signal to CPU wrapper
	I_in/D_in	output	32	write data to CPU wrapper
	I_type/D_type	output	3	write/read byte, half word, or word (listed in include/def.svh)
	valid	logic	64	valid bits of each tag
	hit	logic	1	equal 1 when valid && TA_out equal core_addr[31:10]; otherwise, equal 0
	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

HOMEWORK III

2.4 Detailed Description

You should implement additional instructions in Table 2-3. The detailed instruction types and the immediate formats can be found in Appendix C. You can study *The RISC-V Instruction Set Manual* posted on the course website, too.

Table 2-3: Instruction lists

I-type

31	20	19	15	14	12	11	7	6	0		
imm[11:0]		rs1		funct3		rd		opcode		Mnemonic	Description
imm[11:0]		rs1		000		rd		0000011		LB	$rd = M[rs1+imm]_{bs}$
imm[11:0]		rs1		001		rd		0000011		LH	$rd = M[rs1+imm]_{hs}$
imm[11:0]		rs1		100		rd		0000011		LBU	$rd = M[rs1+imm]_{bu}$
imm[11:0]		rs1		101		rd		0000011		LHU	$rd = M[rs1+imm]_{hu}$

S-type

31	25	24	20	19	15	14	12	11	7	6	0		
imm[11:5]		rs2		rs1		funct3		imm[4:0]		opcode		Mnemonic	Description
imm[11:5]		rs2		rs1		000		imm[4:0]		0100011		SB	$M[rs1+imm]_b = rs2_b$
imm[11:5]		rs2		rs1		001		imm[4:0]		0100011		SH	$M[rs1+imm]_h = rs2_h$

The “b” notation means “byte” and the “h” notation means “half word”. For load operation, you should do signed extension and zero extension for signed and unsigned notation respectively. For store operation, you should store **the lowest byte or the lowest half word** of rs2.

Table 2-4: Instruction cache specification

Type	Size	Line Bits	Associativity	Write Policy
I-cache	1 KB	128	Direct Map	

The features of instruction cache are specified in Table 2-4. You SHOULD use *data_array.v* as the data array and *tag_array.v* as the tag array of the cache. Characteristics of data_array and tag_array are listed in Table 1-4, which are the same as data cache.

Synthesize top module with DC.sdc in script/ directory, and complete physical design with following features:

- Use *Default.globals* as your global variable file. It will use *MMMC.view* as your analysis configuration and use *APR.sdc* as your timing constraint file.
- Don't modify the timing constraint in *APR.sdc* except clock period. Maximum clock period is 20 ns.**
- Do Macro layout only. Don't add IO pad and bonding pad.
- The width of power ring is fixed to **2μm**. Add **one wire group** only.

HOMEWORK III

- e. The width of power stripe is fixed to $2\mu\text{m}$. At least add **one group for each direction**.
- f. The width of block ring is fixed to $2\mu\text{m}$.
- g. Don't add dummy metal.
- h. Must add core filler.
- i. Pass DRC and LVS check without any violation.

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.

2.5 Verification

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

- a. Use *prog0* to perform verification for the functionality of instructions.
- b. 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*.
- c. 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 $y = 0.11 \cdot \text{blue} + 0.59 \cdot \text{gray} + 0.3 \cdot \text{red}$

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:

- e. Proper explanation of your design is required for full credits.
- f. Block diagrams shall be drawn to depict your designs.

HOMEWORK III

- g. 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.
- h. 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.

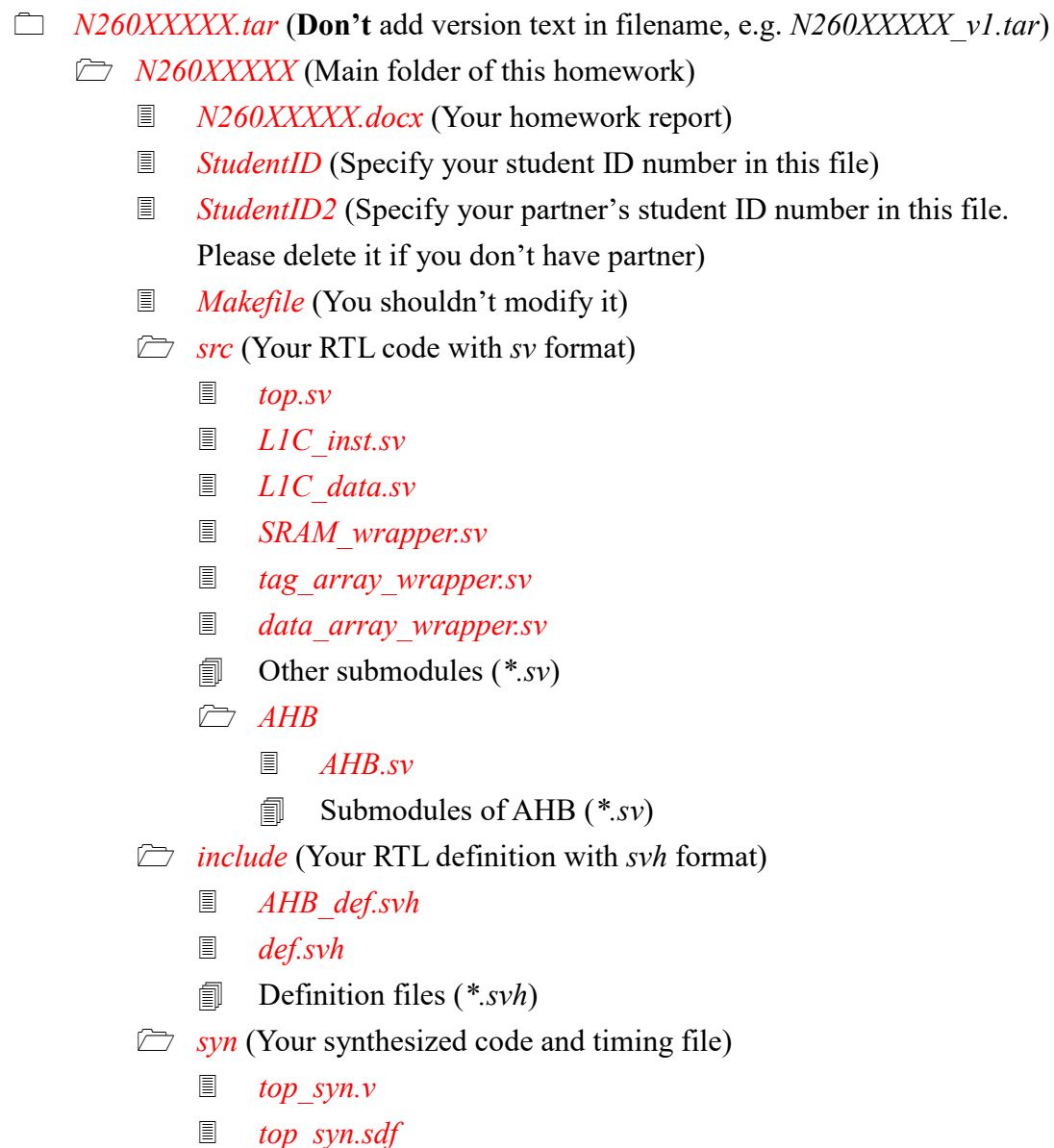
HOMEWORK III

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



HOMEWORK III

- 📁 *pr* (Your post-layout netlist and timing file)
 - 📄 *top_pr.v*
 - 📄 *top_pr.sdf*
 - 📄 *top_pr.gds*
- 📁 *sva* (SystemVerilog assertion file and memory model)
 - 📄 *cache_data.sva* (You shouldn't modify it except for bonus)
 - 📄 *array_mem_model.sv* (You shouldn't modify it)
 - 📄 *DM_mem_model.sv* (You shouldn't modify it)
- 📁 *script* (Any scripts of verification, synthesis or place and route)
 - 📄 *jg.tcl* (You shouldn't modify it)
 - 📄 other 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)
 - 📁 *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)
 - 📄 *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)

HOMEWORK III

- *Makefile* (Compile and generate memory content)
- *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)
 - *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)

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
Problem1	
JasperGold	<i>make cache</i>
Problem2	
RTL	<i>make rtl_all</i>
Pre-layout Gate-level	<i>make syn_all</i>
Post-layout Gate-level	<i>make pr_all</i>

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	<i>make rtlX</i>

HOMEWORK III

Post-synthesis simulation for progX	make synX
Post-layout simulation for progX	make prX
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 GUI with L1 data cache verification without file pollution	make jg
Open DesignVision without file pollution	make dv
Synthesize your RTL code (You need write <i>synthesis.tcl</i> in <i>script</i> folder by yourself)	make synthesize
Open Innovus without file pollution	make innovus
Delete built files for simulation, synthesis or verification	make clean
Check correctness of your file structure	make check
Compress your homework to <i>tar</i> format	make tar

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

```
wc -l src/* src/AHB/* 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	

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
----	--	--	--	--	----	----	---	---	---

HOMEWORK III

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.