VLSI System Design (Graduate Level)
Fall 2022

HOMEWORK IV
Due day: 14:00 Dec. 28 (Wednesday), 2022

Introduction

In this homework, you need to complete the design of the CPU with the appropriate control status register (CSR) and interrupt implemented, add more instructions to the CPU and do the automatic place and route (APR).

You also need to complete the design of 10MHz watchdog timer (WDT) and clock domain crossing (CDC) circuit to face different clock domains (greater than 50MHz CPU, 10MHz WDT) problem and pass Spyglass CDC verification.

Finally you need to write a simple program to booting your CPU. The CPU, the sensor controller and the memories (ROM, SRAM and DRAM), WDT, CDC need to be attached to the AXI (mentioned in *HOMEWORK II*) to form a mini system.

General rules for deliverables

- This homework needs to be completed by INDIVIDUAL student or a TEAM. 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.
- If extra works (like synthesis, post-simulation or additional instructions) are done, please describe them in your final report clearly for bonus points.
- Please follow course policy.
- Verilog and System Verilog 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. Organize your files as the hierarchy in Appendix A.

Report Writing Format

a. Use the submission cover from the course website.
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 (100/100)

1.1 Problem Description

A CPU needs interrupts to communicate with other cores or devices. In this problem, you have to implement CSR to support interrupt operations and some new instructions for your RISC-V CPU from *HOMEWORK Ⅲ*, synthesize the top module and complete physical design using APR tool.

Your CPU should have following new features:

a. The RISC-V ISA with 8 more specified instructions.
b. Has CSR and interrupt mechanism.

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

The WDT is used to detect and recover from CPU malfunctions. When CPU fail to restart WDT, WDT will time out and generate a time-out signal to restart CPU.

There are 2 clock domains in your mini system. You have to design CDC to solve CDC metastability issues, then pass Spyglass CDC check.

1.2 Block Overview

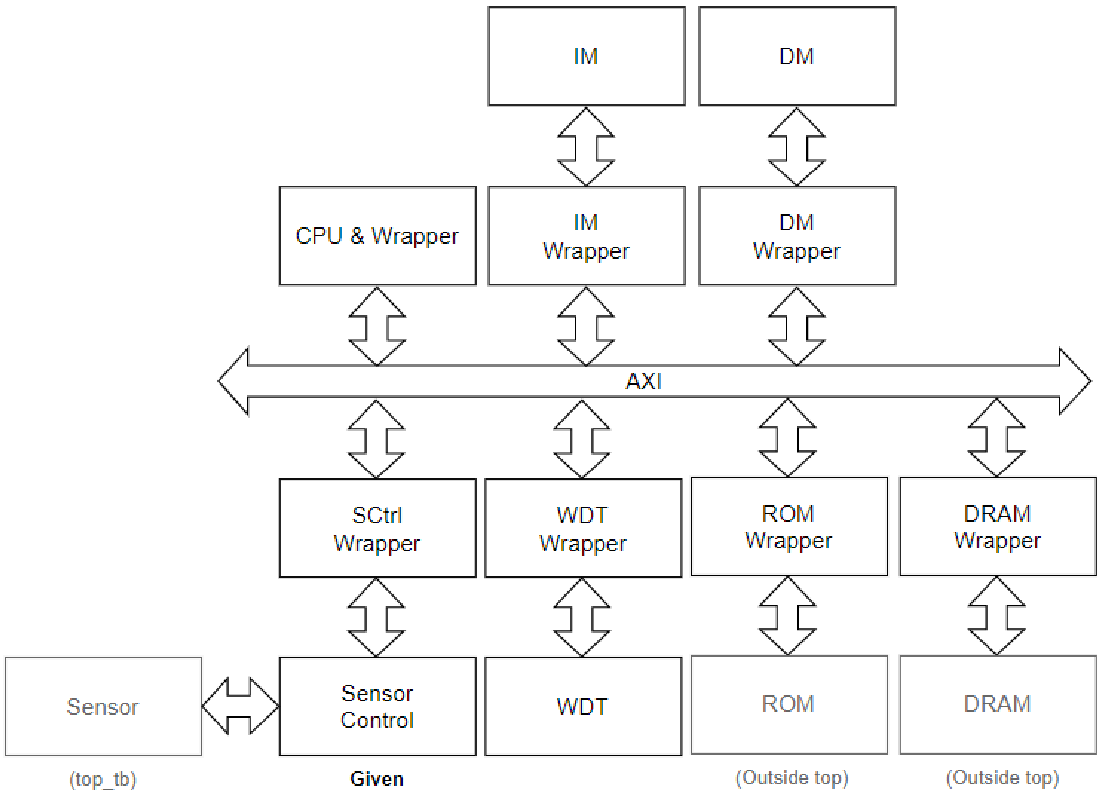

Fig. 1-1: System block diagram

1.3 Module Specification

Table 1-1: Module naming rule

Category	Name			
	File	**Module**	**Instance**	**SDF**
RTL	top.sv	top	TOP	
Gate-Level	top_syn.v		TOP	top_syn.sdf
Physical	top_pr.v		TOP	top_pr.sdf
RTL	L1C_inst.sv	L1C_inst	L1CI	
RTL	L1C_data.sv	L1C_data	L1CD	
RTL	AXI.sv	AXI	AXI	
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	
RTL	sensor_ctrl.sv	sensor_ctrl	sensor_ctrl	

Table 1-2: Module signals

Module	Specifications			
top	**Name**	**Signal**	**Bits**	**Function explanation**
	System signals			
	clk	input	1	System clock
	rst	input	1	System reset (active high)
	clk2	input	1	WDT clock
	rst2	input	1	WDT reset (active high)
	Connect with Sensor			
	sensor_ready	input	1	Ready signal from sensor

	sensor_out	input	32	Data from sensor
	sensor_en	output	1	Enable signal to sensor
	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
	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
	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)
ROM	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]
sensor_ctrl	**Name**	**Signal**	**Bits**	**Function explanation**

	System signals			
	clk	input	1	System clock
	rst	input	1	System reset (active high)
	sctrl_en	input	1	Sensor controller enable (active high)
	sctrl_clear	input	1	Sensor controller clear (active high)
	sctrl_addr	input	6	Sensor controller address
	sctrl_interrupt	output	1	Sensor controller interrupt
	sctrl_out	output	32	Sensor controller data output
	sensor_ready	input	1	Sensor data ready
	sensor_out	input	32	Data from sensor
	sensor_en	output	1	Sensor enable (active high)
	Memory space			
	mem	logic	32	Size: [0:63]
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]

1.4 Detailed Description

You should implement the additional instructions in Table 1-3 and the CSRs in Table 1-4. You only need to implement Machine Mode.

Table 1-3: Instruction lists

☞ System

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

csr		rs1	001	rd	1110011	CSRRW	rd = csr, if #rd != 0
							csr = rs1

csr		rs1	010	rd	1110011	CSRRS	rd = csr, if #rd != 0
							csr = csr | rs1, if that csr bit is writable and #rs1 != 0

csr		rs1	011	rd	1110011	CSRRC	rd = csr, if #rd != 0
							csr = csr & (~rs1), if that csr bit is writable and #rs1 != 0

csr		uimm[4:0]	101	rd	1110011	CSRRWI	rd = csr, if #rd != 0
							csr = uimm(zero-extend)

csr		uimm[4:0]	110	rd	1110011	CSRRSI	rd = csr, if #rd != 0
							csr = csr | uimm(zero-extend), if that csr bit is writable and uimm != 0

csr		uimm[4:0]	111	rd	1110011	CSRRCI	rd = csr, if #rd != 0
							csr = csr & (~uimm(zero-extend)), if that csr bit is writable and uimm != 0

0011000	00010	00000	000	00000	1110011	MRET	Return from traps in Machine Mode
0001000	00101	00000	000	00000	1110011	WFI	Wait for interrupt

Table 1-4: Control Status Register (CSR)

Address	Privilege	Name	Description
0x300	M	mstatus	Machine status register
0x304	M	mie	Machine interrupt-enable register
0x305	M	mtvec	Machine Trap-Vector Base-Address register
0x341	M	mepc	Machine exception program counter
0x344	M	mip	Machine interrupt pending register
0xB00	M	mcycle	Lower 32bits of cycle counter
0xB02	M	minstret	Lower 32bits of instruction-retired counter
0xB80	M	mcycleh	Upper 32bits of cycle counter
0xB82	M	minstreth	Upper 32bits of instruction-retired counter

In Table 1-3, MRET and WFI are listed in *The RISC-V Instruction Set Manual Volume II: Privileged Architecture*. You can treat "hardwire to 0" in description as WIRI (Reserved Write Ignored, Reads Ignore Values) for simplicity except **mtvec**. It should be WARL (Write Any Values, Reads Legal Values.)

Slave configuration is listed in Table 1-5, you should follow the specification. The MEIP of mip can connect to sensor_interrupt of sensor_ctrl directly, or you can design an interrupt controller to handle it. 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. For sensor controller wrapper, you should only implement read operation between 0x1000_0000 and 0x1000_03ff, i.e., read contents between SC[0] and SC[63]. Furthermore, you also need to implement write operation for 0x1000_0100 and 0x1000_0200. CPU can enable sctrl_en signal by writing non-zero data to 0x1000_0100. Similarly, writing non-zero data to 0x1000_0200 will enable sctrl_clear signal.

For WDT wrapper, you should implement write operation for 0x1001_0100 and 0x1001_0200. CPU can enable WDEN signal by writing non-zero data to 0x1001_0100. Similarly, writing non-zero data to 0x1001_0200 will enable WDLIVE signal. Writing data to 0x1001_0300 will change WTOCNT value.

WDT signal is listed in Table 1-6. During normal operation, CPU regularly restarts the watchdog timer to prevent it from timing out. If, due to a hardware fault or program error, the computer fails to restart the watchdog, the timer will elapse and generate a timeout signal to invoke CPU reboot.

Clock domain is listed in Table 1-7. There are 2 clocks which are asynchronous in your design. The phase between clocks can't be determined, you need to implement clock domain crossing circuit between two clock phase to prevent metastability. And all resets need to be design in synchronous reset.

Table 1-5: Slave configuration

Name	Number	Start address	End address
ROM	Slave 0	0x0000_0000	0x0000_3FFF
IM	Slave 1	0x0001_0000	0x0001_FFFF
DM	Slave 2	0x0002_0000	0x0002_FFFF
sensor_ctrl	Slave 3	0x1000_0000	0x1000_03FF
WDT	Slave 4	0x1001_0000	0x1001_03FF
DRAM	Slave 5	0x2000_0000	0x207F_FFFF

Table 1-6: WDT signal

Module	**Specifications**			
	Name	**Signal**	**Bits**	**Function explanation**
	System signals			
	clk	input	1	System clock
	rst	input	1	System reset (active high)
	clk2	input	1	WDT clock
WDT	rst2	input	1	WDT reset (active high)
	Connect with system			
	WDEN	input	1	Enable the watchdog timer
	WDLIVE	input	1	Restart the watchdog timer
	WTOCNT	input	32	Watchdog timeout count
	WTO	output	1	watchdog timeout

Table 1-7: Clock domain

Name	frequency	clock domain
CPU, AXI & wrapper, ROM, IM, DM, sensor ctrl	>50MHz	clk, rst
WDT	10Mhz	clk2, rst2

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. Your physical design should has following features:

a. 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.
b. Don't modify the timing constraint in *APR.sdc* except clock period. Maximum clock period is 20 ns.
c. Do Macro layout only. Don't add IO pad and bonding pad.
d. The width of power ring is fixed to 3μm. Add three wire group.
e. The width of power stripe is fixed to 2μm. At least add one group for each direction.
f. The width of block ring is fixed to 3μm.
g. Don't add dummy metal.
h. Must add core filler.
i. Pass DRC and LVS check without any violation.

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*.

1.5 Verification

You should complete following programs and use the commands in Appendix B to verify your design.

a. For *prog0*, *prog1*, *prog2*, *prog3*and *prog4*, 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. Explain the boot.c.
b. For *prog0*, use main.S to perform verification for the functionality of instructions. Show the terminal result and waveform in the report. The waveform should include new added instructions, and please explain the operation.

c. For *prog1*, the sensor controller will collect data from sensor. If its local memory is full, it will interrupt CPU to copy these data to DM by ISR procedure. After copy is done, ISR will reset the counter of sensor controller, and return to main program. After 4 groups of data is copied, the CPU will sort those data. Such process will execute 2 times. You shouldn't modify the interrupt service routine and the main program. Show the terminal result and waveform in the report. The waveform should include part of booting process and interrupt handling. Explain the operation with waveform.

d. Write a program defined as *prog2* 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.

e. For *prog3&4*, when WDT is enabled, WDT counter starts to count. First time CPU executes to self-loop instruction until WDT times out. Then WDT will interrupt CPU to restart by ISR procedure. Second time CPU regularly restarts the watchdog timer to prevent it from timing out. You shouldn't modify the interrupt service routine and the main program.

f. For CDC check, Use Spygless CDC to verify the correctness of your design. You should show the results of Spygless CDC. If there are waring reports, please explain clearly in your report. Any change in Spyglass.sgdc and Spyglass_CDC.tcl is NOT allowed.

In addition to these verifications, TA will use another program to verify your design. Please make sure that your design can execute the listed instructions correctly.

1.6 Report Requirements

Your report should have the 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 your screenshots of the waveforms and the simulation results on the terminal(RTL,SYN,APR) for the different test cases in your report and illustrate the correctness of your results.
d. Explain your codes of boot.c.
e. Show your screenshots of the Spyglass CDC reports and explain why your CDC circuit can work correctly.

f. Show your snapshots of Floorplan View, Amoeba View and Physical View in Innovus. Also, show the results of Geometry Verification, Connectivity Verification, and Antenna Verification have no violation.
   - If there are some violations, please explain the meaning of the violation

g. 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.

h. Report and show screenshots of your prog0 to prog4 simulation time after synthesis and total cell area of your design. 20% homework credit will be given based on your design performance & area.

Appendix

A. File Hierarchy Requirements

All homework SHOULD be uploaded and follow the file hierarchy and the naming rules, especially the uppercase and the lowercase, specified below. You should create a main folder named your student ID number. It contains your homework report and every subfolder of the problems. The names of the files and the folders are labeled in red color, and the specifications are labeled in black color.

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
      - sctrl_wrapper.sv
      - sensor_ctrl.sv (Sensor controller module)
      - 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*

- *pr* (Your post-layout netlist and timing file)
  - *top_pr.v*
  - *top_pr.sdf*
  - *top_pr.gds*
- *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)
    - *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)
  - 🗏 *main.c* (C code for verification)
  - 🗏 *boot.c* (C code for booting)
  - 🗏 *isr.S* (Interrupt service routine)
  - 🗏 *setup.S* (Assembly code for testing environment setup)
  - 🗏 *link.ld* (Linker script for testing environment)
  - 🗏 *Sensor_data.dat* (Data for sensor)
  - 🗏 *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)
  - 🗏 *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)
- 🗁 *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)
  - 🗏 *isr.S (Interrupt service routine)*
  - 🗏 *setup.S* (Assembly code for testing environment setup)
  - 🗏 *link.ld* (Linker script for testing environment)
  - 🗏 *golden.hex* (Golden hexadecimal data)
- 🗁 *prog4* (Subfolder for Program 4)
  - 🗏 *Makefile* (Compile and generate memory content)
  - 🗏 *boot.c** (C code for verification)
  - 🗏 *main.S* * (Assembly code for verification)
  - 🗏 *main.c* * (C code for verification)
  - 🗏 *isr.S (Interrupt service routine)*
  - 🗏 *setup.S* (Assembly code for testing environment setup)
  - 🗏 *link.ld* (Linker script for testing environment)
  - 🗏 *golden.hex* (Golden hexadecimal data)

🗁 *vip* (JasperGold ABVIP files)

- 🗁 *bridge_duv* (verify for AXI bridge)
  - 🗎 *jg.f* (You shouldn't modify it)
  - 🗎 *top.v* (top module for AXI bridge verification with ABVIP)
- 🗐 Any other files for your design, e.g. submodules and headers
- ✗ **No waveform files allowed**, e.g. files of *fsdb* and *vcd* format
- ✗ **No temporary files allowed,** e.g. *INCA_libs*, *ncverilog.log*, *novas**

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.** You can use macros in Table B-2 to help your verification.

Table B-1: Simulation commands

Simulation Level	Command
Problem1	
RTL	make rtl_all
Pre-layout Gate-level	make syn_all
Post-layout Gate-level	make pr_all

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
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
Open DesignVision without file pollution	make dv
Synthesize your RTL code (You need write *synthesis.tcl* in *script* 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 *tar* format	make tar
Run JasperGold VIP on AXI bridge without file pollution (RTL only)	make vip_b
Open Spyglass without file pollution	make spyglass

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

☞ 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.