VLSI System Design (Graduate Level)
Fall 2018

HOMEWORK IV

Due day: 9:00am Dec. 27 (Thursday), 2018

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 write a simple program to booting your CPU. The CPU, the sensor controller and the memories (ROM, SRAM and DRAM) need to be attached to the AHB (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 (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.
- 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.

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 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.

1.2 Block Overview

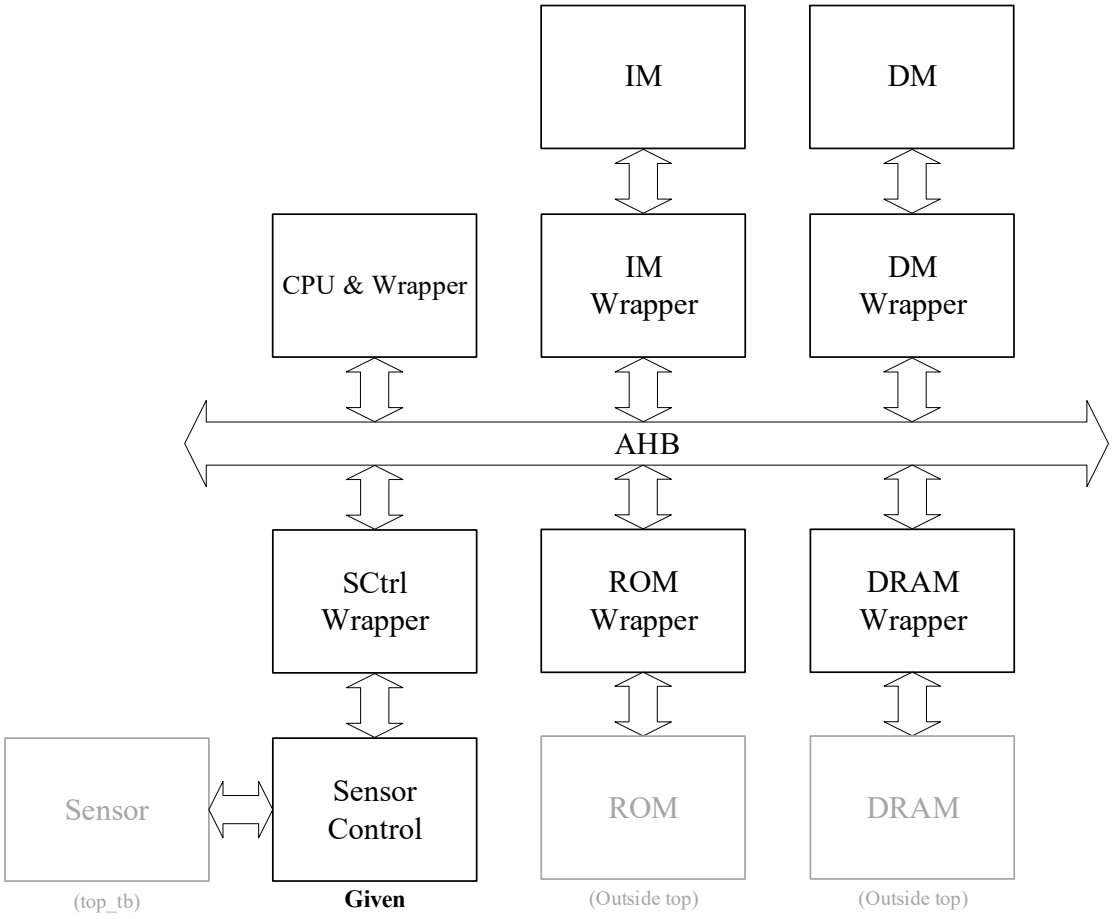

Fig. 1-1: System block diagram

1.3 Module Specification

Table 1-1: Module naming rule

Category	Name			
	File	Module	Instance	SDF
RTL	top.sv	top	TOP	
RTL	top.sv			
Gate-Level	top_syn.v			top_syn.sdf
Physical	top_pr.v			top_pr.sdf
RTL	AHB.sv	AHB	AHB1	
RTL	SRAM_wrapper.sv	SRAM_wrapper	IM1	
RTL	SRAM_wrapper.sv	SRAM_wrapper	DM1	
RTL	SRAM_rtl.sv	SRAM	SRAM1	
Behavior	ROM.v	ROM	ROM1	
Behavior	DRAM.v	DRAM	DRAM1	
RTL	sensor_ctrl.sv	sensor_ctrl	SC	

Table 1-2: Module signals

Module	Specifications			
	Name	Signal	Bits	Function explanation
top	System signals			
	clk	input	1	System clock
	rst	input	1	System reset (active high)
	Connect with Sensor (top_tb)			
	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	32	Address to ROM

	Name	Signal	Bits	Function explanation
top	Connect with DRAM			
	DRAM_Q	input	32	Data from DRAM
	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: [0:1023]
	Memory_byte1	reg	8	Size: [0:1023]
	Memory_byte2	reg	8	Size: [0:1023]
	Memory_byte3	reg	8	Size: [0:1023]
sensor_ctrl	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
	Memory space			
	Memory_byte0	reg	8	Size: [0:2097152]
	Memory_byte1	reg	8	Size: [0: 2097152]
	Memory_byte2	reg	8	Size: [0: 2097152]
	Memory_byte3	reg	8	Size: [0: 2097152]

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. You can study *The RISC-V Instruction Set Manual* posted on the course website.

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 = (#rd == 0)? rd: csr csr = rs1
csr	rs1	010	rd	1110011	CSRRS	rd = csr csr = (#rs1 == 0)? csr: (csr | rs1)
csr	rs1	011	rd	1110011	CSRRC	rd = csr csr = (#rs1 == 0)? csr: (csr & (~rs1))
csr	zimm	101	rd	1110011	CSRRWI	rd = (#rd == 0)? rd: csr csr = imm
csr	zimm	110	rd	1110011	CSRRSI	rd = csr csr = (imm == 0)? csr: (csr | imm)
csr	zimm	111	rd	1110011	CSRRCI	rd = csr csr = (imm == 0)? csr: (csr & (~imm))
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	Requirement
0x300	MRW	mstatus	MIE, MPIE, MPP. Others hardwire to 0
0x304	MRW	mie	MEIE. Others hardwire to 0
0x305	MRW	mtvec	Hardwire to 32'h0001_0000
0x341	MRW	mepc	MEPC[31:2]. Others hardwire to 0
0x344	MRW	mip	MEIP. Others hardwire to 0
0xB00	MRW	mcycle	All
0xB02	MRW	minstret	All
0xB80	MRW	mcycleh	All
0xB82	MRW	minstreth	All

In Table 1-3, MRET and WFI are listed in *The RISC-V Instruction Set Manual Volume II: Privileged Architecture*. In Table 1-4, MRW in privilege means "machine mode read and write". 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_00ff, 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. The reset vector of you program counter should change to 32'h0000_0000.

Table 1-5: 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
sensor_ctrl	Slave 3	0x1000_0000	0x1000_03FF
DRAM	Slave 4	0x2000_0000	0x201F_FFFF

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.
c. Do Macro layout only. Don't add IO pad and bonding pad.
d. The width of power ring is fixed to 2μm. Add one wire group only.
e. The width of power stripe is fixed to 2μm. At least add one group for each direction.
f. Don't add dummy metal.
g. Must add core filler.
h. 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. Use *prog0* to perform verification for the functionality of instructions. Show the result in the report.
b. Write a program defined as *prog1* to perform a booting simulation. you should write a boot program copy data between _dram_i_start and _dram_i_end to _imem_start. You should also write a boot program copy data between __data_start and __data_end to __data_paddr_start. Address named _dram_i_start, _dram_i_end, __data_start and __data_end are defined in link.ld. The booting program should be stored at ROM. You shouldn't modify the interrupt service routine and the main program. 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. The main program will sort these data and store in the same space. After 4 groups of data are sorted, main program will store performance counters in CSR to DM and end the program.

In addition to these verifications, TA will use another program or pattern 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 snapshots of the waveforms and the simulation results on the terminal for the different verification cases in your report and illustrate the correctness of your results.
d. Show your snapshots of Floorplan View, Amoeba View and Physical View in SOC Encounter. Also, show the results of Geometry Verification and Connectivity Verification have no violation.
e. 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.

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*
      - CSR.sv
      - ROM_wrapper.sv
      - DRAM_wrapper.sv
      - sctrl_wrapper.sv
      - sensor_ctrl.sv (Sensor controller module)
      - Other submodules (**.sv*)
      - *AHB*
        - *AHB.sv*
        - Submodules of AHB (**.sv*)
    - *include* (Your RTL definition with *svh* format)
      - *AHB_def.svh*
      - *def.svh*
      - Definition files (**.svh*)

HOMEWORK IV

- *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.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)
- 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

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
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.