VLSI System Design (Graduate Level)
Fall 2024

HOMEWORK III

Due day: 2:00pm Dec. 4 (Wednesday), 2024

Introduction

In this homework, you need to implement CSR instructions, mret, and WFI instructions. Also, there are 4 new IPs that will need to be implemented, the DRAM Wrapper, WDT, ROM Wrapper and DMA. The WDT is clocked by 2 separate clock domain, so you need to debug CDC issues with Spyglass. You also need to write a simple program to boot your CPU, and write some programs to test your SoC. Finally, complete synthesis of your system.

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: N261xxxxx.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 *N261XXXXX.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

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 HW 2, synthesize the top module. 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 circuit to solve CDC metastability issues, then pass Spyglass CDC check.

Also, in this problem, you have to implement a simple DRAM, from which to load your main programs and to which to store your main data. You have to implement ROM wrapper, so as to access ROM to fetch boot code. And you have to implement DMA, so as to carry data from DRAM to IM or DM..

1.2 Module Specification

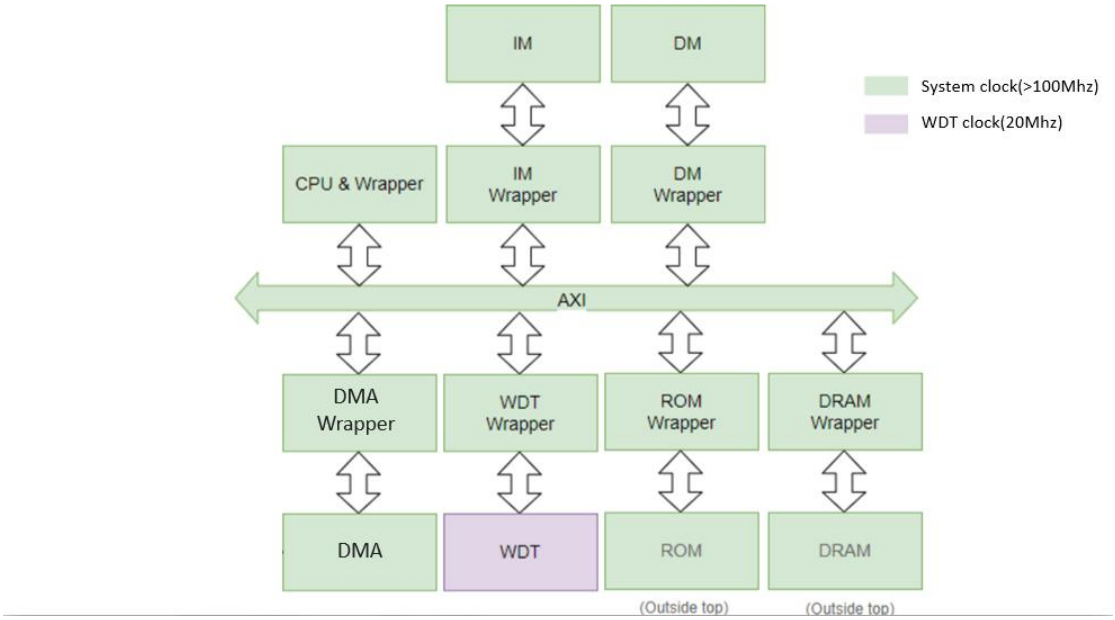

Table 1-1: given modules

Category	Name			
	File	Module	Instance	SDF
RTL	top.sv	top	TOP	
Gate-Level	top_syn.v	top	TOP	top_syn.sdf
RTL	SRAM_wrapper.sv	SRAM_ wrapper	IM1	
RTL	SRAM_wrapper.sv	SRAM_ wrapper	DM1	
RTL	TS1N16ADFPCLLLV TA512X45M4SWSH OD.sv	SRAM	i_SRAM	
Behavior	ROM.v	ROM	i_ROM	
Behavior	DRAM.v	DRAM	i_DRAM	

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 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
	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]
DMA	System Signals			
	clk	input	1	System clock
	rst	input	1	System reset (active high)
	DMAEN	input	1	Enable the DMA
	DMASRC	input	32	Source address of DMA
	DMADST	input	32	Destination address of DMA
	DMALEN	input	32	Total length of the data
	DMA_interrupt	output	1	DMA interrupt

1.3 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: CSR instructions, mret, and WFI

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

31 20		19 15	14 12	11 7	6 0		
imm[11:0]		rs1	funct3	rd	opcode	Mnemonic	Description
0011000	00010	00000	000	00000	1110011	MRET	Return from traps in Machine Mode

31 20		19 15	14 12	11 7	6 0		
imm[11:0]		rs1	funct3	rd	opcode	Mnemonic	Description
0001000	00101	00000	000	00000	1110011	WFI	Wait for interrupt

Table 1-4: m mode CSRs

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

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 DMA_interrupt of DMA directly, or you can design an interrupt controller to handle it. You should design 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 the DMA wrapper, it could be written at addresses 0x1002_0100, 0x1002_0200, 0x1002_0300, and 0x1002_0400. Writing a data of 1 to address 0x1002_0100 enables the DMA. Writing a data to address 0x1002_0200 defines the starting point for the DMA data transfer, writing to 0x1002_0300 defines the endpoint, and writing to 0x1002_0400 specifies the length of the data to be transferred by the DMA.

During booting, the CPU first writes the source address, destination address, and data length into the corresponding registers of the DMA. After completing the register configuration, the CPU asserts the DMAEN signal, at which point the DMA begins transferring data from ROM to the IM or DM. After the DMA completes the data transfer, it will stop functioning and send a DMA interrupt to the CPU. After completing the ISR, the booting program needs to set DMAEN to 0.

Table 1-5: Slave Configuration

NAME	Number	Start address	End address	Property
ROM	Slave 0	0x0000_0000	0x0000_1FFF	memory
IM	Slave 1	0x0001_0000	0x0001_FFFF	memory
DM	Slave 2	0x0002_0000	0x0002_FFFF	memory
DMA	Slave 3	0x1002_0000	0x1002_0400	I/O
WDT	Slave 4	0x1001_0000	0x1001_03FF	I/O
DRAM	Slave 5	0x2000_0000	0x201F_FFFF	memory

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.

Table 1-6: WDT signals

WDT	System Signals			
	clk	input	1	System clock
	rst	input	1	System reset (active high)
	clk2	input	1	WDT counter clock
	rst2	input	1	WDT counter reset (active high)
	Connect with system			
	WDEN	input	1	Enable watchdog timer
	WDLIVE	input	1	Restart watch dog timer
	WTOCNT	input	32	WDT timeout count
	WTO	output	1	WDT timeout, serve as timer interrupt

Table 1-7: Clock domain

name	Frequency	Clock domain
CPU, Bus, Wrappers, ROM, RAMs, DMA	>100MHz	clk
WDEN, WDLIVE, WTOCNT, WTO	20MHz	clk2

Problem2

2.1 Problem Description

In this problem, you have to support booting operations for your RISC-V CPU , and implement some testing programs.

2.2 Verification

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

a. For prog0, to prog4, you should write a boot program defined as boot.c or boot.S 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 by DMA. 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. Test your SoC using a program defined as *prog3* to perform interrupt check on WDT, without clock delay. CPU will start executing program and enter dead loop. When WDT assert timer interrupt, CPU will be reset. After CPU is reset, it will periodically reset timer.
f. Test your SoC using a program defined as *prog4* to perform interrupt check on WDT, with clock delay. The testing program is the same as prog4, except that the clock is delayed in testbench.

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 may use another program to verify your design. Please make sure that your design can execute the listed instructions correctly.

Problem3

3.1 Problem Description

In this problem, you should use Spyglass to do CDC check on your design

Report Requirements

- Proper explanation of your design is required for full credits.
- Block diagrams shall be drawn to depict your designs.
- 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 your codes of prog1, prog2.
- Explain your codes of boot.c.
- Report your Superlint coverage
- Report and show screenshots of your prog0 to prog5 simulation time after synthesis and total cell area of your design.
- Explain how the interrupt mechanism work
- Show your screenshots of the Spyglass CDC reports and explain why your CDC circuit can work correctly

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

- *N261XXXXX.tar* (**Don't** add version text in filename, e.g. *N261XXXXX_v1.tar*)
  - *N261XXXXX* (Main folder of this homework)
    - *N261XXXXX.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*
      - *SRAM_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)
  - *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)
- *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)
  - *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)
- *prog5* (Subfolder for Program 5)
  - *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
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
Do CDC check with Spyglass	make spyglass

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

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