VLSI System Design (Graduate Level)
Fall 2021

HOMEWORK II

Due day: 2:00 p.m. Nov. 10 (Wednesday), 2021

Introduction

This homework is to let you be familiar with ARM bus protocol (AMBA 2.0).

You have to complete the simplified AXI. You should combine the CPU, IM, DM and the AXI to form a mini system and synthesize them.

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

Please complete a simplified Advanced eXtensible Interface (AXI) with 2 masters and 2 slaves. You DO NOT need to implement cache functions, atomic accesses, protection units. You have to verify the AXI architecture by using JasperGold AXI ABVIP. A more detailed description of this problem can be found in Section 1.4.

1.2 Block Overview

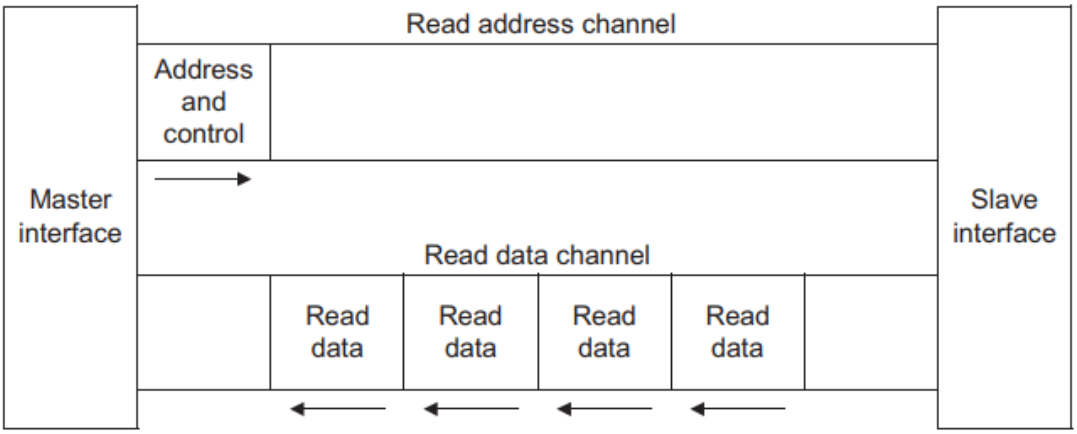

Fig. 1-1: AXI Read Channel Architecture

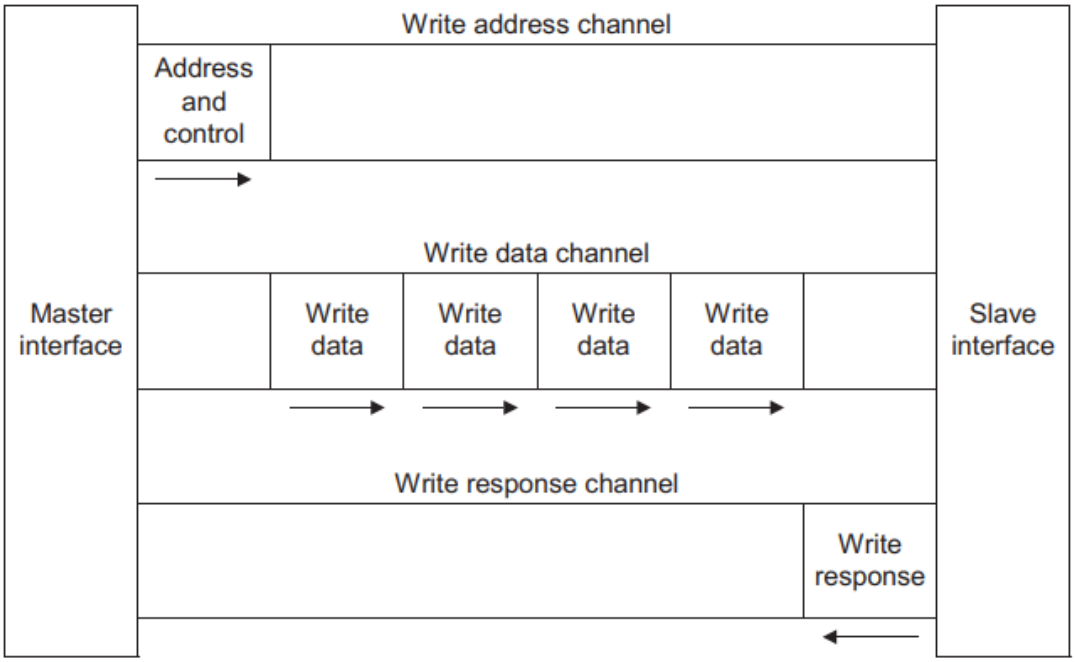

Fig. 1-2: AXI Write Channel Architecture

1.3 Module Specification

	Top file
MASTER	CPU_wrapper.sv
BRIDGE	AXI.sv
SLAVE	SRAM_wrapper.sv

1.4 Detailed Description

The simplified AXI architecture should have the following features:

a. Transfer type: INCR
b. Transfer response: OKAY and DECERR
c. Burst operation: Master is Single transfer only; Bridge and Slave are Burst Mode.
d. Others: No cache support, no protection units support, and no atomic accesses.
e. All components should be synthesizable
f. The conceptual architecture of AXI architecture is shown in Fig. 1-1 and Fig.1-2. The TOP module of AXI code will be provided on the course website. Please download and complete them by yourself.
g. Modules:

AXI (Bridge)

i. Arbiter: Implement the arbitration scheme. Round-Robin is recommended.
ii. Decoder: Decode address to slaves' ID. The addresses between 0x0000_0000 and 0x0000_ffff belong to slave 1. The addresses between 0x0001_0000 and 0x0001_ffff belong to slave 2. Others belong to default slave.
iii. Default Slave: Default slave has responsibility to respond DECERR when the master tries to access unmapped address.
iv. Default Master: Default master need to perform valid = LOW transfers. You don't have to create a default master module.
v. Working at the positive edge of clock

CPU wrapper (Master)

i. Design the master wrapper between CPU and AXI.

Sram wrapper (Slave)

i. Design the SRAM_wrapper to be compatible with AXI.

1.5 Verification

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

a. Complete *vip/master_duv/top.v* , *vip/slave_duv/top.v* to bind your AXI module to verify with ABVIP tools. *vip/bridge_duv/top.v* is bound for you. DO NOT modify the parameters that are defined in top.v.

Category	Name	
	File	Module/Instance
RTL	vip/bridge_duv/top.v	AXI/axi_duv_bridge
		axi4_slave/axi_slave_0
		axi4_slave/axi_slave_1
		axi4_master/axi_master_0
		axi4_master/axi_master_1
RTL	vip/master_duv/top.v	CPU/CPU1
		CPU_wrapper/axi_duv_master
		axi4_slave/axi_monitor_0
		axi4_slave/axi_monitor_1
RTL	vip/slave_duv/top.v	SRAM_wrapper/axi_duv_slave
		axi4_master/axi_monitor

b. Use JasperGold ABVIP to verify the correctness of your AXI.

c. You should pass all assertions to get full credits.

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. Any change in *jg_bridge.tcl, jg_master.tcl, jg_slave.tcl* and *jg.f* is **NOT** allowed.

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 results of verification with proper explanation.

Problem2

2.1 Problem Description

Combine AXI with your CPU in *Homework 1* to complete a mini system whose architecture is shown in Fig. 2-1.

2.2 Block Overview

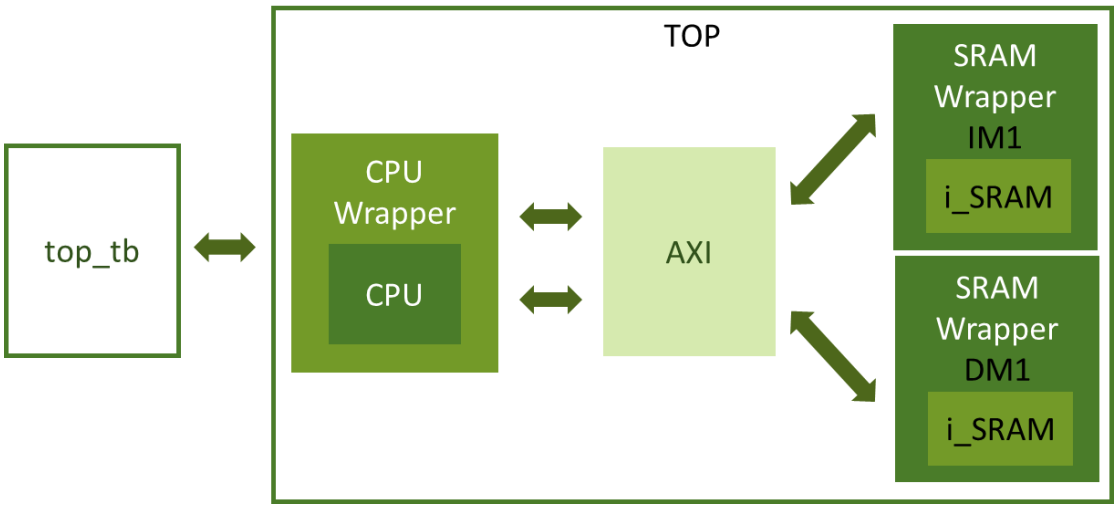

Fig. 2-1 Architecture of AXI with CPU and Memories

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	SRAM_wrapper.sv	SRAM_wrapper	IM1	
RTL	SRAM_wrapper.sv	SRAM_wrapper	DM1	
RTL	SRAM_rtl.sv	SRAM	i_SRAM	

Table 2-2: Module signals

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

2.4 Detailed Description

The mini system should have the following features:

a. Don't modify any timing constraint except clock period in *DC.sdc*. Maximum clock period is 20 ns.
b. Design the master wrapper between CPU and AXI.
c. Modify SRAM_wrapper to be compatible with AXI.
d. Connect the system as shown in Fig. 2-1. Your CPU should occupy two masters, one for instruction, and another for data. IM must be Slave 1 and DM must be Slave 2. Start address of IM is 0x0000_0000 and start address of DM is 0x0001_0000.
e. 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 64. All elements are **signed 4-byte integers** 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 perform multiplication. The multiplicand is stored at the address named *mul1* in ".rodata" section defined in *data.S*. The multiplier is stored at the address named *mul2* in ".rodata" section defined in *data.S*. The multiplicand and the multiplier are **signed 4-byte integers**. Their product is **signed 8-byte integers** and should be stored at the address named *_test_start* in "_test" section defined in *link.ld*.
d. Write a program defined as prog3 to perform greatest common divisor(GCD). The first number is stored at the address named div1 in ".rodata" section defined in data.S. The second data is stored at the address named div2 in ".rodata" section defined in data.S. These two numbers are **unsigned 4-byte integers**. The result should be stored at the address named _test_start in "_test" section defined in link.ld. The values of the quotient and the remainder should follow C99 specification. "When integers are divided,

the result of the / operator is the algebraic quotient with any fractional part discarded. If the quotient **a/b** is representable, the expression **(a/b)*b + a%b** shall equal **a**.

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 verifications, **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:

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 test cases in your report and illustrate the correctness of your results.
d. 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 the Superlint.

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*
      - *SRAM_wrapper.sv*
      - Other submodules (**.sv*)
      - *AXI*
        - *AXI.sv*
        - Submodules of AXI (**.sv*)
    - *include* (Your RTL definition with *svh* format)
      - *AXI_def.svh*
      - Definition files (**.svh*)
    - *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)

📂 *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)

📂 *prog2* (Subfolder for Program 2)

- *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)

📂 *prog3* (Subfolder for Program 3)

- *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)

📂 *syn* (Your synthesized code and timing file)

- 📂 *top_syn.v*
- 📂 *top_syn.sdf*

📂 *vip* (JasperGold ABVIP files)

HOMEWORK II

- *bridge_duv* (verify for AXI bridge)
  - *jg.f* (You shouldn't modify it)
  - *top.v* (top module for AXI bridge verification with ABVIP)
- *master_duv* (verify for AXI master)
  - *jg.f* (You shouldn't modify it)
  - *top.v* (top module for AXI master verification with ABVIP)
- *slave_duv* (verify for AXI slave)
  - *jg.f* (You shouldn't modify it)
  - *top.v* (top module for AXI slave verification with ABVIP)

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	
RTL	make vip_b
RTL	make vip_m
RTL	make vip_s
Problem2	
RTL	make rtl_all
Post-synthesis	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} FSDB=1
Dump waveform (with array)	make {rtlX,synX} 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

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.