

## HOMEWORK II

**Due day: 3:00 p.m. Nov. 6 (Wednesday), 2024**

### 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 is an **INDIVIDUAL** assignment, must completed by yourself.
- 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 II

of your report: N260xxxxx.docx. Please save as docx file format and replace N260xxxxx with your student ID number. (Let the letter be uppercase.)

3. 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 II

### 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

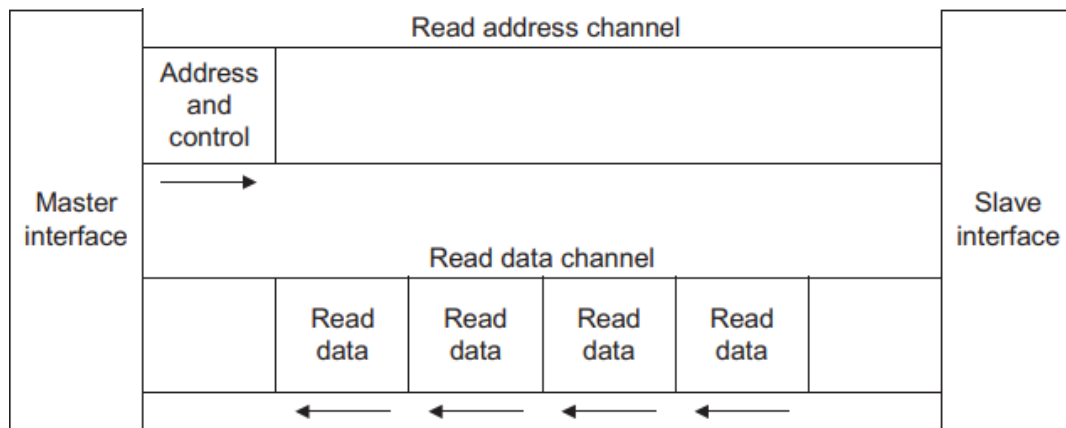


Fig. 1-1: AXI Read Channel Architecture

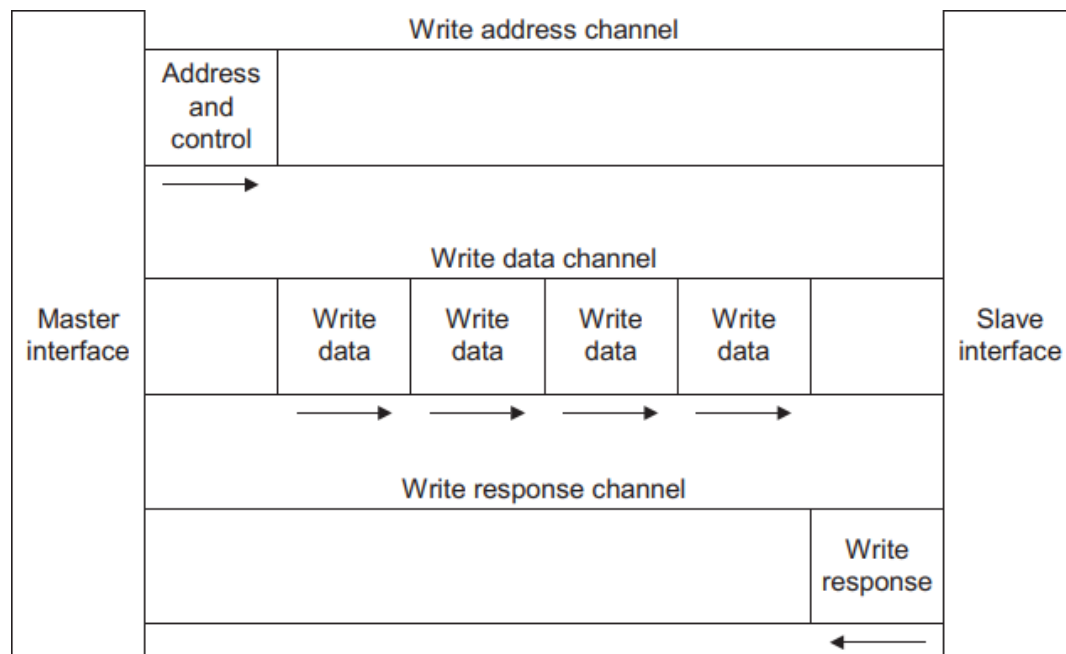


Fig. 1-2: AXI Write Channel Architecture

#### 1.3 Module Specification

HOMEWORK II

|        |                 |
|--------|-----------------|
|        | 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:

- Transfer type: INCR
- Transfer response: OKAY and DECERR
- Burst operation: Master is Single transfer only; Bridge and Slave are Burst Mode.
- Others: No cache support, no protection units support, and no atomic accesses.
- Only consider **in-order** situation
- All components should be synthesizable**
- 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.
- Modules:

### **AXI (Bridge)**

- Arbiter: Implement the arbitration scheme. **Round-Robin** is recommended.
- 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.
- Default Slave: Default slave has responsibility to respond DECERR when the master tries to access unmapped address.
- Default Master: Default master need to perform valid = LOW transfers. You don't have to create a default master module.
- Working at the **positive edge** of clock

### **CPU wrapper (Master)**

- Design the master wrapper between CPU and AXI.

### **Sram wrapper (Slave)**

- Design the SRAM\_wrapper to be compatible with AXI.

## 1.5 Verification

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

## HOMWORK II

- 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:

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

## HOMEWORK II

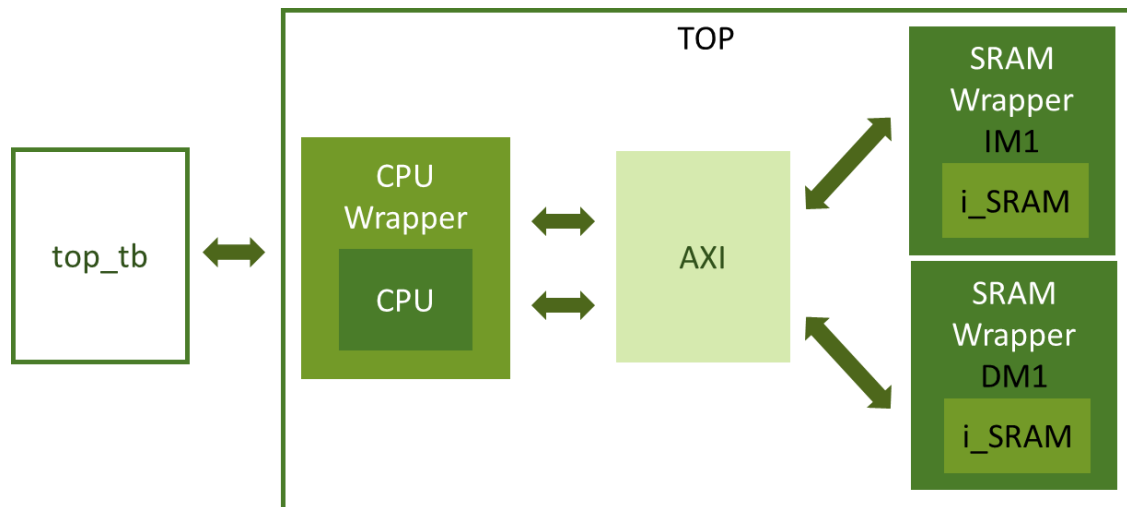


Fig. 2-1 Architecture of AXI with CPU and Memories

## 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       |
| TS1N16ADFPCLLL<br>VTA<br>512X45M4SWSHOD | Memory Space   |        |      |                            |
|   | MEMORY         | logic  | 32   | Size: [512][32]            |

## 2.4 Detailed Description

## HOMEWORK II

The mini system should have the following features:

- Don't modify any timing constraint except clock period in *DC.sdc*.**  
Maximum clock period is **10 ns**.
- Design the master wrapper between CPU and AXI.
- Modify SRAM\_wrapper to be compatible with AXI.
- 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.**
- 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.

- Use *prog0* to perform verification for the functionality of instructions.
- 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*.
- 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*.
- 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,

## HOMEWORK II

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:

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



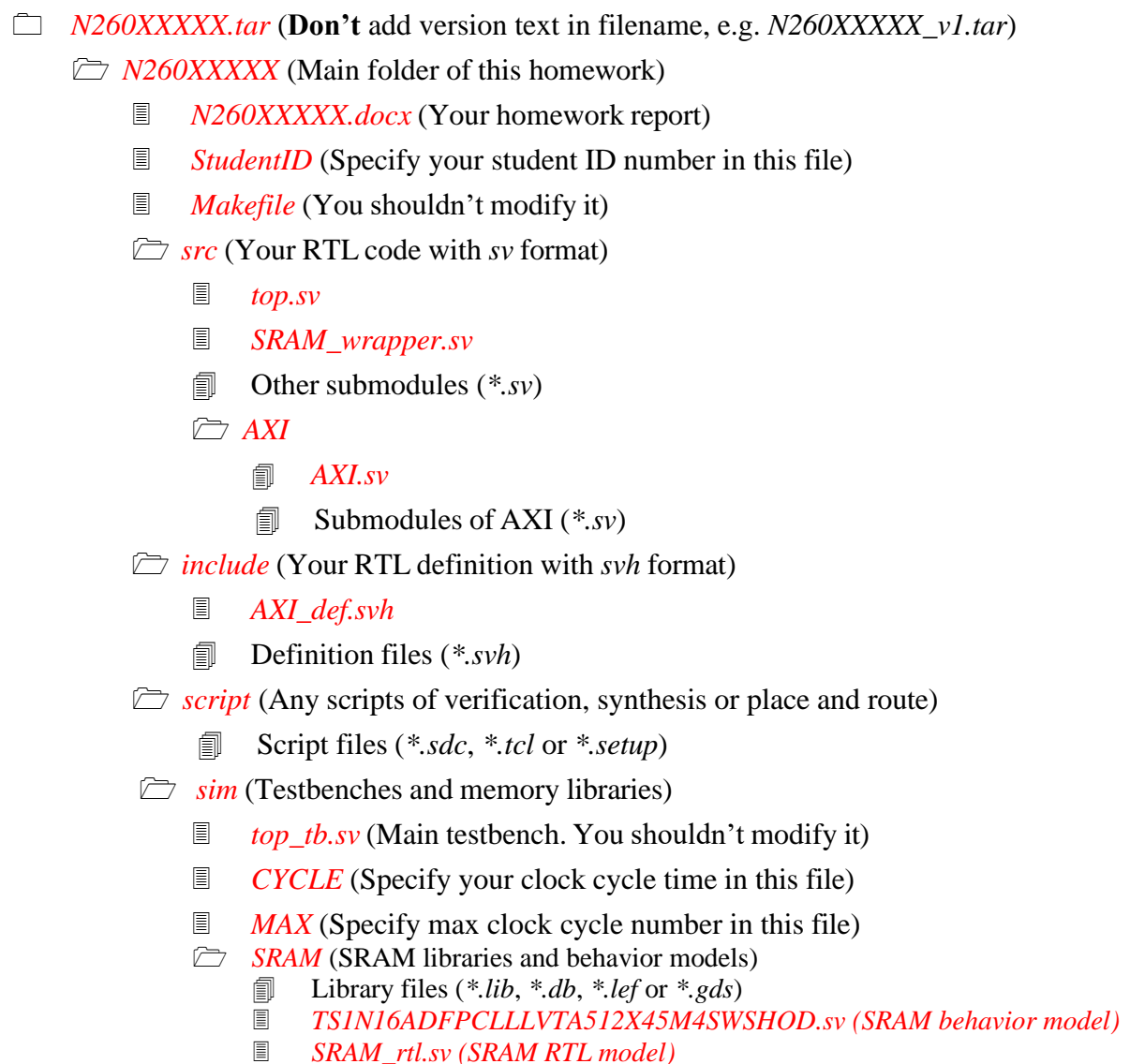
## HOMEWORK II

# 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 II

- 📁 *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.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)

HOMEWORK II

- 📁 *prog4* (Subfolder for Program 4)
  - 📄 *Makefile* (Compile and generate memory content)
  - 📄 *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)
  - 📄 *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)
- 📁 *prog6* (Subfolder for Program 6)
  - 📄 *Makefile* (Compile and generate memory content)
  - 📄 *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 area file)
  - 📄 *top\_syn.v*
  - 📄 *top\_syn.sdf*
  - 📄 *area\_rpt.txt*
- 📁 *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)
  - 📁 *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)

HOMEWORK II

## 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              | <code>make vip_b</code>   |
| RTL              | <code>make vip_m</code>   |
| RTL              | <code>make vip_s</code>   |
| Problem2         |                           |
| RTL              | <code>make rtl_all</code> |
| Post-synthesis   | <code>make syn_all</code> |

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   | <code>make rtlX</code>               |
| Post-synthesis simulation for progX  | <code>make synX</code>               |
| Dump waveform (no array)   | <code>make {rtlX,synX} FSDB=1</code> |
| Dump waveform (with array)   | <code>make {rtlX,synX} FSDB=2</code> |
| Open nWave without file pollution  | <code>make nWave</code>              |
| Open Superlint without file pollution  | <code>make superlint</code>          |
| Open DesignVision without file pollution   | <code>make dv</code>                 |
| Synthesize your RTL code (You need write <i>synthesis.tcl</i> in <i>script</i> folder by yourself) | <code>make synthesize</code>         |
| Delete built files for simulation, synthesis or verification                                       | <code>make clean</code>              |
| Check correctness of your file structure   | <code>make check</code>              |
| Compress your homework to <i>tar</i> format  | <code>make tar</code>                |

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

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

HOMEWORK II

## 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 — |   |

HOMEWORK II

☞ 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.