Homework 4

***Note:* From this homework on, students will be allowed to work as pairs. Each group should prepare only one report and each group member is expected to upload that same report. You may use Ninova message boards to find yourselves a teammate.**

1. Basic single-read/single-write (1r1w) memory block

A 1-D flip-flop based memory will be designed to be used as a basis for the memory elements within your RISC-V processor project.

a. It is required that the **memory width** and **memory depth** to be modifiable on instantiation (you will need to define these attributes as parameters, names are arbitrary).

b. The design should strictly have the following inputs and outputs:

- ❖ **clk and rst:** Clock and reset signals. Reset signal is wanted to be asynchronous low-active, and will clear the contents of the registers.
- ❖ **rd_addr0 and wr_addr0:** Address inputs for read and write ports. Appropriate lengths must be sourced from the module parameters defined at **a.** *(Hint: $clog2 function)*
- ❖ **rd_dout0:** Data output for read port. Should **asynchronously** show the data located at the address specified by rd_addr0.
- ❖ **wr_din0:** Data input for write port. Appropriate length must be sourced from the module parameters defined at **a.**
- ❖ **we0:** Single bit, write port write enable signal. When high, data from wr_din0 should be written to the address specified by wr_addr0 at the positive edge of the clock. When low, no action will be taken.

2. RV32I Instruction Set

The RISC-V 32-bit Integer Base Instruction Set is given at the end of this document, at Table-1.

a. Carefully read about the RISC-V 32-bit Integer instructions from the official **RISC-V Instruction Set Manual, Chapter 2** (*found at Ninova: Course Resources\ riscv-spec-20191213.pdf*) and provide a simple and understandable pseudo code style description **for each individual instruction**. You may take the example below as a reference; where *rd* is the destination register, *rs1* and *rs2* are source registers and *sxt(Imm)* means sign extended immediate value.

Mnemonic	Name	Description
SLTI	Set less-than immediate	if *rs1 < sxt(Iimm)* then *rd* ← *1* else *rd* ← *0*
ADD	Addition	*rd* ← *rs1 + rs2*

b. Produce a machine code set that includes 16 instructions. You should include at least one instruction from each type (R-type, I-type, S-type and so on… *Refer to RISC-V Instruction Set Manual, Chapter 2*) and do not use the same instruction more than once. Show the special fields and describe the work that the instruction does. The set does not have to produce a meaningful program, they just have to be in the correct format and individually mapping to the work that you have specified. An example is given below:

ADD x4, x2, x4: $\underbrace{0000000}_{func7}\_\_\underbrace{00100}_{rs2}\_\_\underbrace{00010}_{rs1}\_\_\underbrace{000}_{func3}\_\_\underbrace{00100}_{rd}\_\_\underbrace{0110011}_{opcode}$

x4 ← x2+x4

R-TYPE

Add the values from register addresses 2 and 4 and write the result on the register address 4.

c. Manually write the machine code forms of the instructions you specified at **b.** into a text file, in either binary or hexadecimal format. Each instruction should be separated with a newline.

d. A “NOP” instruction (No Operation) is a special name for an instruction that causes a processor to do nothing during the given instruction cycle. Learn how NOP is performed in RISC-V and write the machine code for it. Fill the rest of the text file you created at **c.** with NOP machine code, until the file contains exactly 128 lines.

3. Instruction Memory

Based on the memory block designed in the first step of the homework, you will produce a testbench model for a RISC-V instruction memory

a. Create a Verilog testbench file and instantiate a **basic single read/single write (1r1w) memory block (the design created at step 1 of the homework)** with 32-bit word length and a depth of 128.

b. Within your testbench, define a parameter named *"initFile"*. Here, specify the full path of the machine code text file you wrote in Steps 2.c and 2.d into the *"initFile"* parameter, as a string. Depending whether you've used hex or binary code format in this file, use ***$readmemb*** or ***$readmemh*** Verilog functions in order to directly load the instructions from text file within your memory. Alternatively, you can load the contents of the text file inside the memory by giving each text line as input to the memory and asserting **we0** signal at the appropriate times.

c. Perform a behavioral simulation by writing the instruction text file contents into the memory in any way you like, then read each address of the memory one by one. Verify that the write/read operations are correct (printing to console or an output text file is recommended).

4. Data Memory

Based on the memory block designed in the first step of the homework, you will create a design for a RISC-V data memory.

a. Create a new Verilog file and copy the contents of the **basic single read/single write (1r1w) memory block** designed at 1) in it. A RV32I data memory should allow byte and half-word writes in order to support SB and SH instructions. Add a new input named **wr_strb** in this design, in order to control byte-writes. You may code the functionality in any way you like, as long as you can show that it correctly supports SB and SH instructions; thus the length of **wr_strb** input is up to you. Extra module parameters can be added if deemed necessary. An example use case can be seen below, but note that you may use a different approach *(Hint: generate-for).*

```
When wr_strb == 4'b1000
          memblock[wr_addr][31:24] <= wr_din0[31:24]

When wr_strb == 4'b0010
          memblock[wr_addr][15:8] <= wr_din0[15:8]

When wr_strb == 4'b0011
          memblock[wr_addr][15:0] <= wr_din0[15:0]

When wr_strb == 4'b1100
          memblock[wr_addr][31:0] <= wr_din0[15:0]
```

b. Create a Verilog testbench file and instantiate a **data memory module** with 32-bit word length and a depth of 128. If you added any other module parameters, set them accordingly.

c. Write a test in order to show that the write strobe signal works as expected and that it can support SB, SH and SW instructions. Perform a behavioral simulation and show your results clearly.

5. RV32 Register File

Based on the memory block designed in the first step of the homework**,** you will create a register file design for a 32-bit RISC-V processor.

a. Create a new Verilog file and copy the contents of the **basic single read/single write (1r1w) memory block** designed at 1) in it. Similar to 1.b, add one more **read port** to this design (**rd_addr1, rd_dout1**). Set the default values for depth and length parameters to 32.

b. In RISC-V, register address 0 should contain constant zero value. Modify your code so that address zero cannot be written on (attempts to write on address 0 should cause no changes), and will always give zero when read.

c. Perform full OpenLane flow on your design. **You are expected to obtain a result that is clean of timing (setup and hold violations), DRV (max slew and max capacitance violations), DRC, LVS and antenna violations.** Max fanout violations are okay. Use generated reports in order to show that your design is clean of these violations (*Hint*: *signout folder*).

d. Report the final area and clock frequency of your design. What is the worst slack for this clock frequency? At what path this slack is observed? Comment on why this path is turned out to be the critical path in your design (*Hint: The file signoff/rcx-sta.rpt*). Add the screenshot of the final layout of your design in your report.

e. Create a new testbench and instantiate a new **register file** module. Do not specify values for module parameters. Write a test in order to show that all functions of the register file are correct, especially the zero register behavior. **If you find out that the zero register is trimmed by synthesis tool, make sure to fix your design code to prevent it from getting trimmed.** Perform post place and route functional simulation and clearly show your results.

imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND

Table 1: RISC-V 32 bit Base Integer Instruction Set