Homework 5

In this project, a **Function Unit (FU)** block will be designed for the RISC-V Base instruction set **(RV32I)** shared at Table 1. The FU will include arithmetic, logic and shift operations required to implement RV32I. An example design for FU is shown in Fig. 1 on 512th page of reference [1]. FU shown in Fig. 1 on 512th page of reference [1] is made of Arithmetic Logic Unit (ALU) shown in Fig. 2 on 513th page of [1] and a shifter shown in 519th page of reference [1].

1. Plan your ALU design

a. Consider the list of instructions given at Table 1, and the pseudo-codes you've derived for these instructions in Homework 4. Think on what inputs and outputs the ALU should have in order to fully support the RV32I set. Plan your ALU design and draw its top-level diagram by using logic gates, multiplexers, sub-blocks etc., and showing the data inputs, the selection inputs, the status outputs and data outputs *(Hint: Consider that you will be generating control signals depending on opcode, func3 and func7 fields. Note that you can make number comparison by subtracting the numbers and evaluating carry and overflow. Also, you may need to use more than one adder circuitry for some instructions, but assume that just one of them will be solely dedicated to and located within your ALU).*

b. Explain the purposes of all the selection signals.

c. Explain the purposes of all the status signals.

d. Explain the purposes of all the sub-blocks that you are going to use for execution of the arithmetic and logic instructions on the data inputs. Remember that a specific sub-block may be involved in the execution of more than one instruction.

e. Prepare the Function Table for your ALU as in Table 2 on 519th page of reference [1]

2. Write Verilog HDL code of your ALU.

a. Write Verilog HDL code for all the sub-blocks included in your plan drawn at Step 1 as separate modules. Choose an adder/subtractor topology depending on your findings in Homework 1 and justify your selection. Usage of Verilog addition operator (+) is not permitted.

b. Instantiate and connect all your sub-blocks in a newly created top level ALU HDL file. Your top-level design should look similar to Fig. 2 on page 513 of reference [1].

c. Synthesize the top-level code of your ALU within OpenLane flow. By using the generated post synthesis reports, obtain total cell usage, estimated area, critical path delay and estimated power consumption of the design.

d. Perform behavioral simulation for your ALU by using iverilog and gtkwave. Aim to test all functionalities of your ALU. Clearly explain your simulation results. Having a self-checking testbench with ample amount of test operands is encouraged.

3. Plan your shifter design

a. Considering the shifting instructions that's given at Table 1; SLL, SRL, SRA, SLLI, SRLI and SRAI. Think on what inputs and outputs the shifter should have in order to properly support these instructions *(Hint: Consider that you will be generating control signals depending on opcode, func3 and func7 fields).*

b. Explain the purposes of all the selection/control signals.
c. Prepare the Function Table for your shifter as in Table 3 on 521st page of reference [1]

4. Write Verilog HDL code of your shifter.

a. Write Verilog HDL code for the shifter you planned at Step 3. Shift operators can be used. For better performance, you may look up to MUX based barrel shifter topologies.
b. Synthesize the design within OpenLane flow. By using the generated post synthesis reports, obtain total cell usage, estimated area, critical path delay and estimated power consumption of the design.
c. Perform behavioral simulation for your shifter by using iverilog and gtkwave. Aim to test all functionalities of your shifter. Clearly explain your simulation results. Having a self-checking testbench with ample amount of test operands is encouraged.

5. Forming the top-level FU and top-level testing

a. Instantiate and connect your ALU and shifter in a newly created top level FU HDL file. Your top level design should look similar to Fig. 10 on page 522 of reference [1].
b. Apply full OpenLane flow on your circuit. Try to obtain a violation-free final design (fanout violations are okay). Discuss the problems you encountered and solutions you've tried to solve them.
c. Obtain total area, die dimensions and power consumption of your design. Using the timing reports of the routed design, pinpoint the critical path of your circuit. Why do you think this specific path has the most delay?
d. Perform post-synthesis functional simulation for FU, providing a bunch of test operands for each function of FU. Aim to test all arithmetic, logic, shift and relation (branches and SLT, SLTI, SLTIU) operations at least once, while including potential use scenarios (subtracting two negative numbers, adding two negative numbers, overflowing, adding two signed numbers and getting a negative result, and so on…). Explain your results clearly.

5. Designing the Control Word

Design a control word for your processor by considering instruction structures of RV32I set as shown on page 18 of "ITU_VLSI_II_RF_Datapath_SingleCycle_Control.pdf" lecture slides. The length and the format of the control word will differ from the slides because you're using RISC-V ISA.

a. Determine what control signals you will need in order to implement RV32I. According to this, draw your control word structure as shown on page 18 of "ITU_VLSI_II_RF_Datapath_SingleCycle_Control.pdf" lecture slides.
b. Produce control word encoding table as shown on page 20 of "ITU_VLSI_II_RF_Datapath_SingleCycle_Control.pdf" lecture slides.
c. Give explanations and justifications regarding to your work. Briefly explain why you need each specific control signal.

6. Forming the Datapath

Make required connections between the modules you designed in previous homeworks; Register File (RF), Function Unit (FU) and multiplexers as shown on page 6 of "ITU_VLSI_II_RF_Datapath_SingleCycle_Control.pdf" lecture slides. Note that your RF

will be sized 32x32 as required by RV32I set. Give the same names to the signals as much as possible in order to make traceability easier for us. Your top module should be named as "Datapath.v" and look similar to the grey box on page 28 of "ITU_VLSI_II_RF_Datapath_SingleCycle_Control.pdf" lecture slides. Be aware that the slides just introduce an example, you will have a different architecture for RISC-V in the end.

a. Check out the algorithm given at Figure 1. When the rows starting with "if" and "for" keywords are removed and C is initialized to 1, we get the following operations:

   C=1, C=2C, C=C-N, C=C+A, C=C-N.

   Assume that "A" and "N" are located in the Data Memory, and C will be formed from scratch in one of the registers. State that by which instructions you can perform these operations in the given order. Produce needed control word and input signal list in order to implement these five operations in the given order.
b. Write a testbench in order to apply the list which you produced in Step (a) to your DP.
c. Instantiate the DP first, then instantiate the Data Memory that you've written in Homework 4 in your testbench with word size 32 and with an arbitrary depth (do not synthesize it) and connect it to your DP. You may initiate your memory with a text file using $readmemb function, as you did in Homework 4.
d. Perform behavioral simulation using the testbench you created in the previous steps. Test your circuit with 3 different pairs of {A,N}. Pick A and N in such a way that you can obtain a positive valued C, a negative valued C and C=0 in the end.
e. Present and explain your simulation 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

Left-to-right modular multipcation algorithm

Input: $A=(a_{k-1},a_{k-2},\cdots,a_1,a_0)_2$, $B=(b_{k-1},b_{k-2},\cdots,b_1,b_0)_2$, $N=(n_{k-1},n_{k-2},\cdots,n_1,n_0)_2$

Output: $C=AB \bmod N=(c_{k-1},c_{k-2},\cdots,c_1,c_0)_2$

```
Step1: C=0
Step2: for i=k-1:0
Step3:          C=2C
                if C≥N
Step4:                  C=C-N
Step5:          if b_i=1
Step6:                  C=C+A
                        if C≥N
Step7:                          C=C-N
```

Figure 1

References

1) Morris R. Mano, Charles R. Kime, Tom Martin, "Logic and Computer Design Fundamentals", Pearson; 5th edition, August 28, 2015.
2) David A. Patterson, John L. Hennessy, "Computer Organization and Design MIPS Edition: The Hardware/Software Interface", Morgan Kaufmann; 6th edition (December 4, 2020).
3) David A. Patterson, John L. Hennessy, "Computer Organization and Design RISC-V Edition: The Hardware Software Interface", Morgan Kaufmann; 2nd edition (December 31, 2020).
4) Andrew Waterman, Krste Asanovic, The RISC-V Instruction Set Manual Volume I: Unprivileged ISA Document Version 20191213, December 13, 2019