VERIFICATION OF SIMPLE ALU WITH UVM

In the given task, verification of a simple arithmetic logic unit is expected with universal verification methodology (UVM). In the following sections, you can find details about the simple ALU and reference resources while doing the task.

Simple ALU

Simple ALU module has 3 input and 3 output ports. Port names and their descriptions are explained in Table 1.

Table 1: Input and Output Ports

PORT NAME	DESCRIPTION
i_clk	Clock Input
i_rst	Synchronous active high reset input
i_memData	Data from memory
o_memData	Data to be written to memory
o_memAddr	Memory address for read and write operations
o_memWrEnable	Enable memory write operation

ALU has 16-bit architecture. There are two types for instructions, R-type, and I-type. R-type instructions have 3 sections: opcode, operand1_address, and operand2_address. The opcode is 4-bit width and defines the operation will be executed by ALU. Operand1_address and operand2_address sections are 6-bit width and hold the address of operands which is used in execution step. The only difference between I-type instructions and R-type is that I-type does not have operand2_address section. Instead of that, they have immediate value, which is used directly in executions step. Instruction formats of R-type and I-type can be seen in Figure 1 and Figure 2 respectively.

OPCODE[15:12]	OPERAND1_ADDRESS[11:6]	OPERAND2_ADDRESS[5:0]

Figure 1: R-type Instruction Format

OPCODE[15:12]	OPERAND1_ADDRESS[11:6]	IMMEDIATE_VALUE[5:0]

Figure 2: I-type Instruction Format

Instruction names, opcodes, and short descriptions are given in Table 2.

Table 2: Instruction Details

INSTRUCTION NAME	OPCODE	DESCRIPTION
ADD	0000	Addition of operand1 to operand2
ADDi	1000	Addition of operand1 to immediate value.
SUB	0001	Subtraction of operand2 from operand1
SUBi	1001	Subtraction of immediate value from operand1
SRA	0010	Arithmetic right shift of operand1 by operand2
SRAi	1010	Arithmetic right shift of operand1 by immediate value
SRL	0011	Logical right shift of operand1 by operand2
SRLi	1011	Logical right shift of operand1 by immediate value
SLL	0100	Logical left shift of operand1 by operand2
SLLi	1100	Logical left shift of operand1 by immediate value
AND	0101	Bitwise and of operand1 and operand2
ANDi	1101	Bitwise and of operand1 and immediate value
OR	0110	Bitwise or of operand1 and operand2
ORi	1110	Bitwise or of operand1 and immediate value
XOR	0111	Bitwise xor of operand1 and operand2
XORi	1111	Bitwise xor of operand1 and immediate value

The state diagram of the simple ALU can be seen in Figure 3. After i_rst deasserted, system enters FETCH state. In fetch state ALU reads instruction and send address of first operand to memory. Then, ALU enters DECODE state. In this state, ALU decides the operation will be done in the next state and sends the second operand address to memory if the instruction is R-type. In EXECUTE state, ALU performs the operation according to opcode. ALU sends calculated value and write address to memory and assert memWrEnable for one clock cycle. Write address always same with operand1_address. In the last state, which is PCCOUNTER, ALU increments the program counter by 1 and returns to FETCH state.

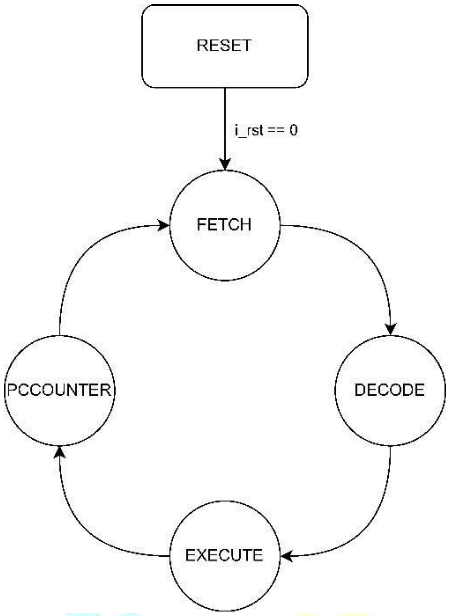

Figure 3: State Diagram

While writing test, use the first 32 addresses for instructions, and last 32 addresses for data.

The design will be given to you one week after the assignment date.

Reference Resources

There are lots of online resources available for UVM. We recommend verification guide to finish the task as much as possible. You can reach the website here. The first table is more than enough to finish the task. Also, you can check the testbench example which has the same architecture with what we want from you. Use same file organization with the testbench example. We are expecting you to verify that all instructions work as expected. You do not have to go deeper of UVM. Keep the task as simple as possible. You will use EDA Playground for your task. Open a free account and send the EDA playground link of task before deadline.