Alimomeni

810100215

CA3/ESL

ALU implementation using chisel:

```
import chisel3._
import chisel3.stage.ChiselStage
import chisel3.util._

class ALU(val width: Int) extends Module {
  val io = IO(new Bundle {
    val a      = Input(UInt(width.W))
    val b      = Input(UInt(width.W))
    val aluOp  = Input(UInt(3.W))           // 3-bit opcode
    val result = Output(UInt(width.W))
  })

  io.result := 0.U // default

  switch(io.aluOp) {
    is("b000".U) { io.result := io.a }            // NOP: Res = A
    is("b001".U) { io.result := io.a + io.b }     // ADD
    is("b010".U) { io.result := io.a - io.b }     // SUB
    is("b011".U) { io.result := io.a & io.b }     // AND
    is("b100".U) { io.result := io.a | io.b }     // OR
    is("b101".U) { io.result := io.a ^ io.b }     // XOR
    is("b110".U) { io.result := io.b }            // LD: Res = B
    is("b111".U) { io.result := io.a >> 1 }       // SHR: logical right shift by 1
  }
}
```

Output verilog code after running ALU maker code :

```
object ALUMaker extends App {
  println ("(Generating Verilog file for ALU)")
  (new chisel3.stage.ChiselStage).emitVerilog(new ALU( width = 16), Array("--target-dir",
    "Verilog"))
}
```

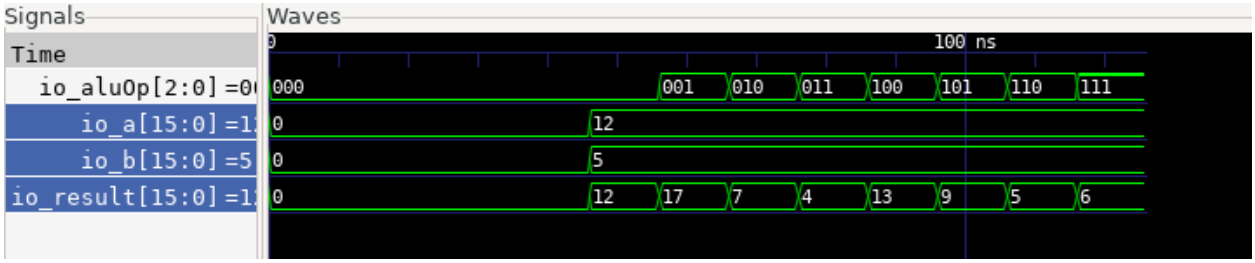
```
module ALU(
  input         clock,
  input         reset,
  input  [15:0] io_a,
  input  [15:0] io_b,
  input  [2:0]  io_aluOp,
  output [15:0] io_result
);
```

```
);
  wire  _T = 3'h0 == io_aluOp; // @[Conditional.scala 37:30]
  wire  _T_1 = 3'h1 == io_aluOp; // @[Conditional.scala 37:30]
  wire [15:0] _T_3 = io_a + io_b; // @[ALU.scala 17:38]
  wire  _T_4 = 3'h2 == io_aluOp; // @[Conditional.scala 37:30]
  wire [15:0] _T_6 = io_a - io_b; // @[ALU.scala 18:38]
  wire  _T_7 = 3'h3 == io_aluOp; // @[Conditional.scala 37:30]
  wire [15:0] _T_8 = io_a & io_b; // @[ALU.scala 19:38]
  wire  _T_9 = 3'h4 == io_aluOp; // @[Conditional.scala 37:30]
  wire [15:0] _T_10 = io_a | io_b; // @[ALU.scala 20:38]
  wire  _T_11 = 3'h5 == io_aluOp; // @[Conditional.scala 37:30]
  wire [15:0] _T_12 = io_a ^ io_b; // @[ALU.scala 21:38]
  wire  _T_13 = 3'h6 == io_aluOp; // @[Conditional.scala 37:30]
  wire  _T_14 = 3'h7 == io_aluOp; // @[Conditional.scala 37:30]
  wire [14:0] _GEN_0 = _T_14 ? io_a[15:1] : 15'h0; // @[Conditional.scala 39:67 ALU.scala 23:30 ALU.scala 13:13]
  wire [15:0] _GEN_1 = _T_13 ? io_b : {{1'd0}, _GEN_0}; // @[Conditional.scala 39:67 ALU.scala 22:30]
  wire [15:0] _GEN_2 = _T_11 ? _T_12 : _GEN_1; // @[Conditional.scala 39:67 ALU.scala 21:30]
  wire [15:0] _GEN_3 = _T_9 ? _T_10 : _GEN_2; // @[Conditional.scala 39:67 ALU.scala 20:30]
  wire [15:0] _GEN_4 = _T_7 ? _T_8 : _GEN_3; // @[Conditional.scala 39:67 ALU.scala 19:30]
  wire [15:0] _GEN_5 = _T_4 ? _T_6 : _GEN_4; // @[Conditional.scala 39:67 ALU.scala 18:30]
  wire [15:0] _GEN_6 = _T_1 ? _T_3 : _GEN_5; // @[Conditional.scala 39:67 ALU.scala 17:30]
  assign io_result = _T ? io_a : _GEN_6; // @[Conditional.scala 40:58 ALU.scala 16:30]
endmodule
```

Testbench using ChiselTest library :

```
import chisel3.iotesters.{PeekPokeTester, Driver, ChiselFlatSpec}

class testbench(c: ALU) extends PeekPokeTester(c) {
  def testOp(op: Int, a: BigInt, b: BigInt, expected: BigInt): Unit = {
    poke(c.io.a, a)
    poke(c.io.b, b)
    poke(c.io.aluOp, op)
    step(1)
    expect(c.io.result, expected)
  }

  val a = 12
  val b = 5

  testOp( op = 0, a, b, a)           // NOP
  testOp( op = 1, a, b, a + b)       // ADD
  testOp( op = 2, a, b, a - b)       // SUB
  testOp( op = 3, a, b, a & b)       // AND
  testOp( op = 4, a, b, a | b)       // OR
  testOp( op = 5, a, b, a ^ b)       // XOR
  testOp( op = 6, a, b, b)           // LD
  testOp( op = 7, a, b, a >> 1)      // SHR
}
```

We use ALUtester code to run the testbench and get .vcd outpt to see the waveform on gtkwave:

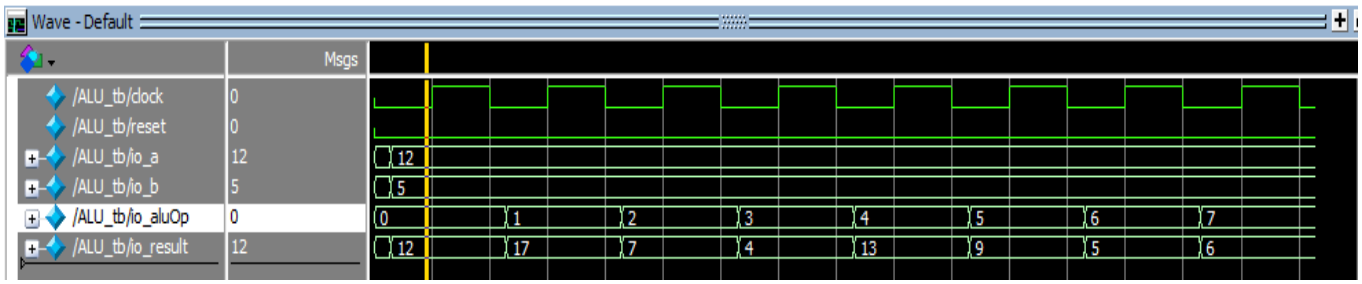


Verilog testbench :

```
task testOp;
  input [2:0] op;
  input [15:0] a;
  input [15:0] b;
  input [15:0] expected;

  begin
    io_a = a;
    io_b = b;
    io_aluOp = op;
    #10; // Wait one clock cycle
    if (io_result !== expected)
      $display("FAILED: op=%0d a=%0d b=%0d -> expected=%0d, got=%0d", op, a, b, expected, io_result);
    else
      $display("PASSED: op=%0d a=%0d b=%0d -> result=%0d", op, a, b, io_result);
  end
endtask
```

```
  // Main stimulus
  initial begin
    reset = 0;
    io_a = 0;
    io_b = 0;
    io_aluOp = 0;

    #15; // Wait for reset/setup

    // Define test inputs
    reg [15:0] a = 12;
    reg [15:0] b = 5;

    testOp(3'd0, a, b, a);          // NOP
    testOp(3'd1, a, b, a + b);      // ADD
    testOp(3'd2, a, b, a - b);      // SUB
    testOp(3'd3, a, b, a & b);      // AND
    testOp(3'd4, a, b, a | b);      // OR
    testOp(3'd5, a, b, a ^ b);      // XOR
    testOp(3'd6, a, b, b);          // LD
    testOp(3'd7, a, b, a >> 1);     // SHR

    $finish;
  end

endmodule
```

Output waveform:



1. Verilog Equivalent Testbench for the Chisel Testbench

- The Chisel testbench is written in a functional style using `poke` and `expect` methods to set inputs and check outputs step-by-step.

- The Verilog equivalent consists of a testbench module where inputs are declared as registers and outputs as wires.

- In Verilog, the clock signal is explicitly generated, and tests are executed sequentially using a `task`.

- Timing in Verilog is controlled explicitly using `#` delays and waits, while Chisel relies on the simulator's step function.

2. Similarities

- Both testbenches test the ALU with the same input values.

- Both change inputs and compare outputs with expected values.

- Both display results to verify correctness.

3. Differences

- **Clock management: In Chisel, clock management is implicit and handled by the simulation environment, whereas in Verilog the clock is explicitly generated and controlled by the user.**

- **Input/output declaration: Chisel uses classes and dynamic methods (`poke`, `expect`) to set and check signals, while Verilog declares inputs as registers and outputs as wires.**

- **Test execution: Chisel applies tests through function calls that advance the simulation step-by-step; Verilog uses tasks with explicit timing delays (`#`) to apply inputs and check outputs.**

- **Code structure: Chisel code is higher-level and object-oriented, making it more concise and abstract, whereas Verilog code follows traditional hardware description style with explicit signals and timing.**

- **Timing control: Chisel relies on simulation steps to manage timing, while Verilog requires explicit delay statements to simulate timing behavior.**

4. Waveform Analysis

- In the Verilog waveform, clock, reset, inputs (`io_a`, `io_b`, `io_aluOp`), and output (`io_result`) signals are clearly visible with explicit timing relative to the clock edges.

- In Chisel, the simulation environment internally manages timing, so input-to-output propagation is abstracted and optimized, with less visible clock behavior in waveforms.

- Inputs in Verilog are synchronized precisely with clock edges; in Chisel, timing depends on simulator steps and testbench structure.

- Verilog waveforms allow more detailed timing verification on a cycle-by-cycle basis.

Conclusion

Chisel testbenches provide higher-level abstraction and simpler test creation, ideal for fast development and verification. Verilog testbenches give detailed control over timing and clock, which is valuable for cycle-accurate simulation and debugging. The choice between the two depends on project needs, required simulation accuracy, and preferred design methodology.