ALU SystemVerilog

```
typedef enum {ADD, MULT, SUB, XOR} opcode_t;

module ALU(input clk, reset, input [7:0] operand1, operand2, input opcode_t
           opcode, output [15:0] result);

  always @ *
    case opcode
      ADD: result= operand1 + operand2
      MULT:   *
      SUB:    -
      XOR:    ^
    endcase

endmodule

interface alu_intf( )
  clocking cb1 @(posedge clk)
    input/output output operand1/2, opcode //define w/ respect to testbench
                 input result;

endinterface;
```

these are testbench OUTPUTS

```
module tb;
  alu_intf my_intf
  ALU DUT(.clk(my_intf.clk), .opcode(my_intf.opcode));

  task reset()
  begin
    m_intf.cbl.reset <= 0;
    @(my_intf.cbl)
    my_intf.cbl.reset <= 1;
  end

  task one_opcode(input opcode_t op, input [7:0] op1, op2)
  begin
    my_intf.cbl.op <= op;
    my_intf.cb1.operand1 <= op1;
    my_intf.cb2.operand2 <= op2;
    @ my_intf.cbl
  end

  task n_opcode(int n);
  begin
    for(i=0; i<n; i++)
      one_opcode( $random )
  endtask
```

delay by 1 cycle

synchronizes to next clock edge

one_opcode(ADD, 0, 1)
(MULT, 7, 3)

```
initial begin
  reset();
->10
  n_opcode(10)
->10
```

interfaces → synthesizable