Lecture6 Assignments 1&2

6.1 Read and understand the Verilog code, write brief function description for each module.

(1)adder : Performs a 32-bit addition of inputs a and b, assigning the result to y = a + b.

(2)alu : 32-bit Arithmetic Logic Unit, using case-alucont to control. AND 00 , OR 01 , add 10 , signed comparison 11. The zero flag is set if the result is zero.

(3)sl2 : Shifts the input a left by 2 bits, and set the 2 least sig bits as 00. This is a combinational circuit used for multiplication by 4 or address adjustment.

(4)signext : Extends a 16-bit signed input (a[15:0]) to 32 bits by replicating its MSB (sign bit) to upper 16 bits.

(5)flopr : Sequential Circuits -- A parameterized D flip-flop with synchronous operation and asynchronous reset. On the rising edge of clk, if reset is high, q is cleared to 0; otherwise, q takes the value of d. This is a sequential circuit.

(6)mux2 : A parameterized 2-to-1 multiplexer. Selects between d0 (when s is 0) and d1 (when s is 1) to assign to y. This is a combinational circuit.

(7)flopenr : A parameterized D flip-flop with asynchronous reset and an enable signal. On the rising edge of clk, if reset is high, q is cleared to 0; if en is high, q takes the value of d. This is a sequential circuit with write control.

(8)flopenrc : A parameterized D flip-flop with asynchronous reset, enable, and synchronous clear, with additional clear functionality. On the rising edge of clk, if reset is high, q is cleared to 0; if clear is high, q is cleared to 0; if en is high, q takes the value of d.

6.2 ALU design and simulation

Assignment-1: 8-bit ALU Design

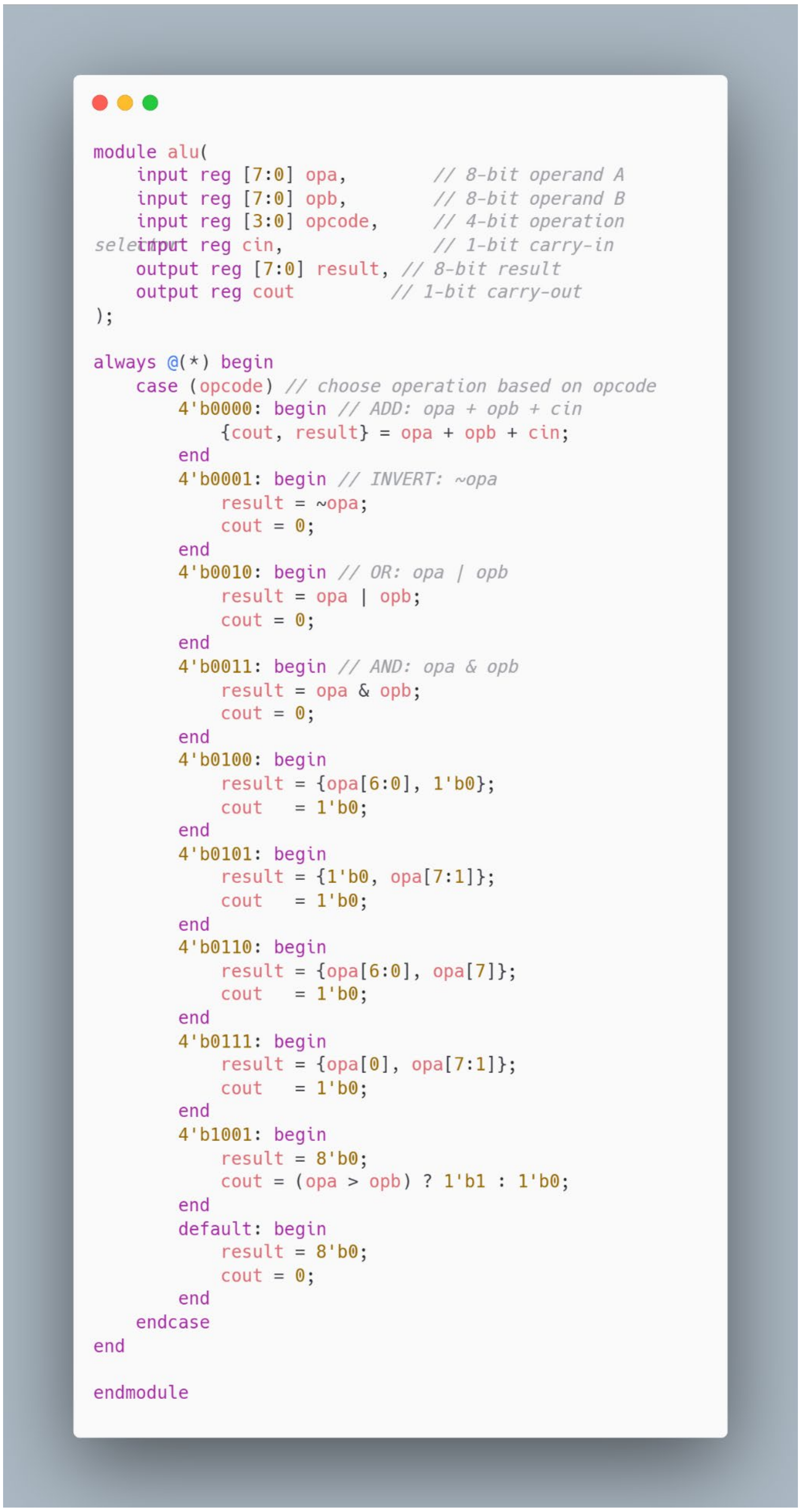
Write Verilog code for an 8-bit Arithmetic and Logic Unit, and verify your design.

Assignment-2: 8-bit ALU with an accumulator

The accumulator is an edge triggered register with asynchronous reset. Write Verilog code for this 8-bit ALU with an accumulator register.

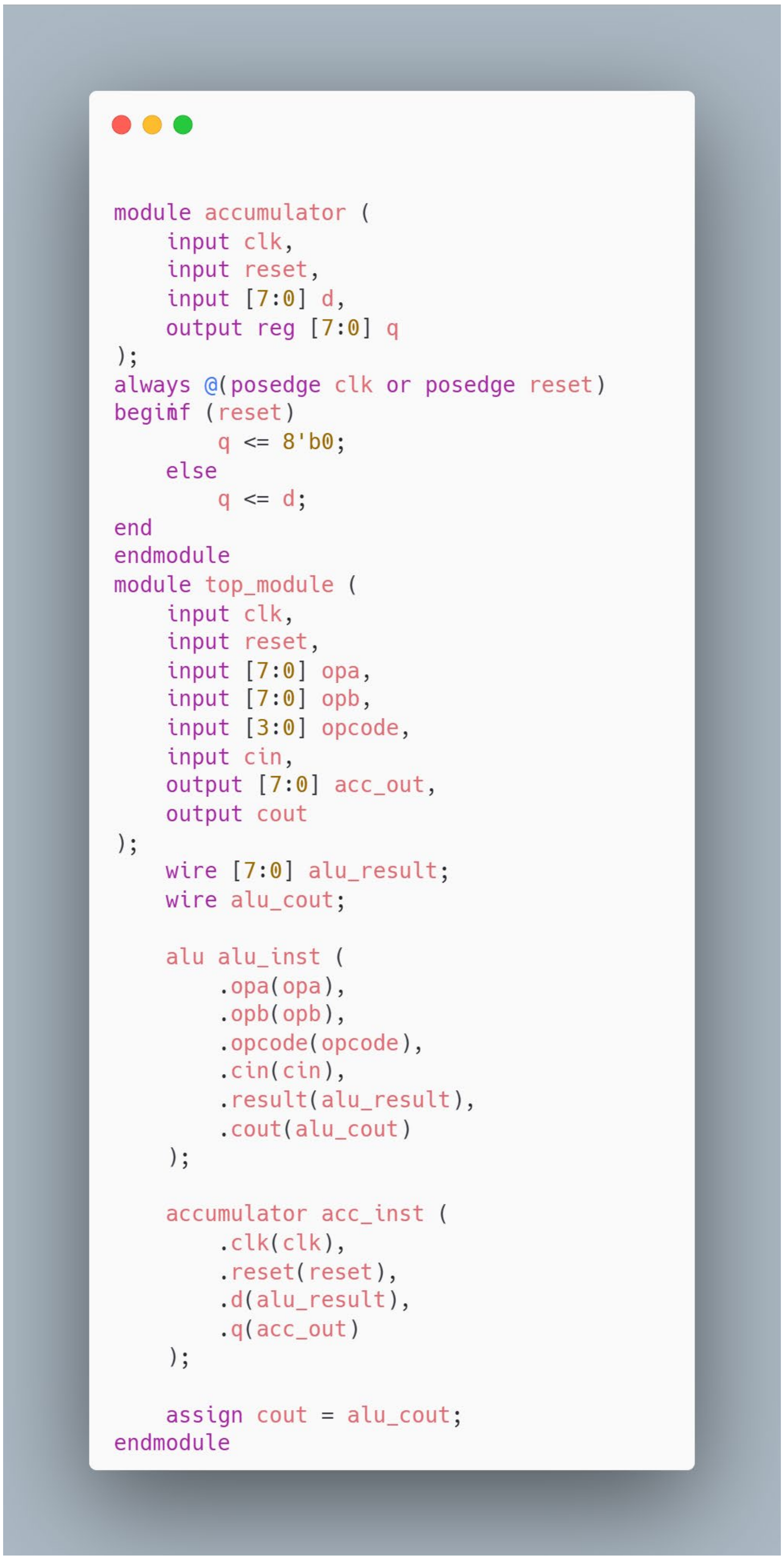
1.The first part of the assignment implements the code for the entire module

I use the module alu to finish the combinational part.



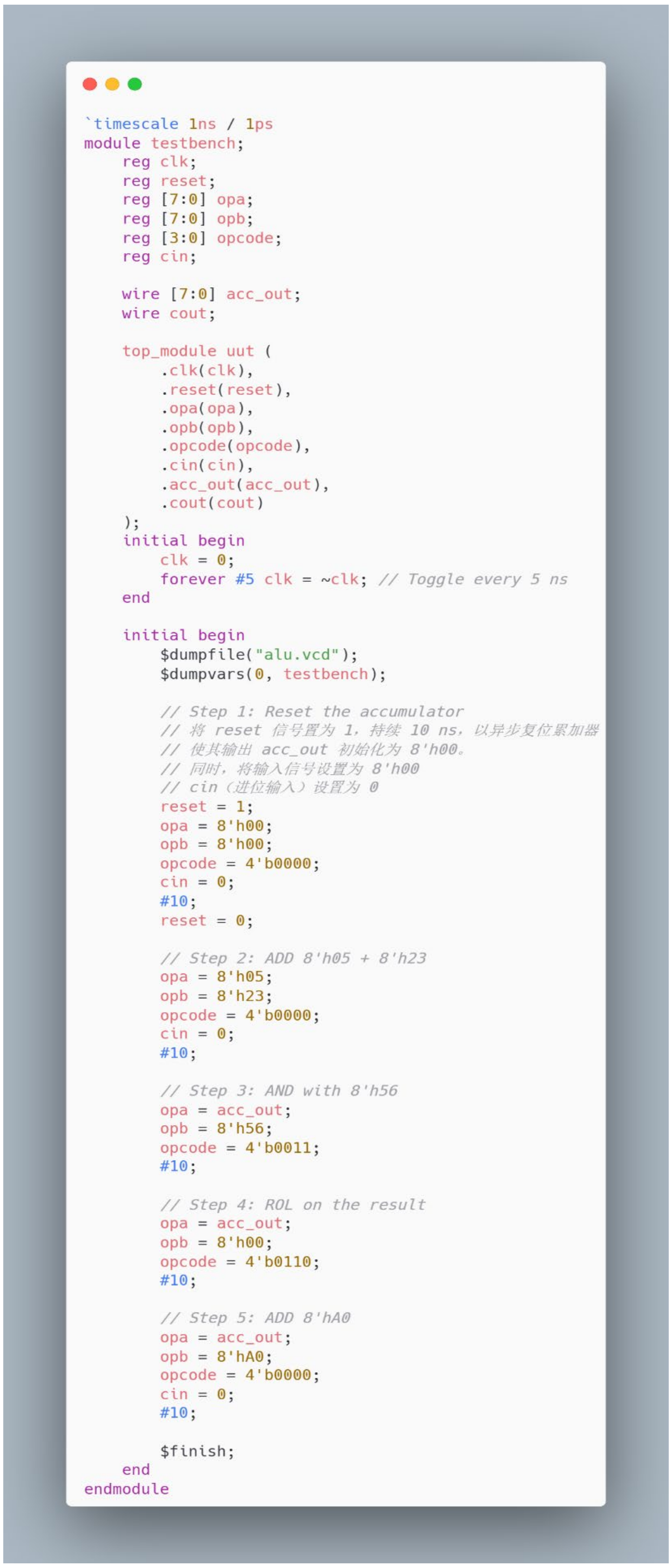
```
module alu(
    input reg [7:0] opa,        // 8-bit operand A
    input reg [7:0] opb,        // 8-bit operand B
    input reg [3:0] opcode,     // 4-bit operation
seleinput reg cin,              // 1-bit carry-in
    output reg [7:0] result, // 8-bit result
    output reg cout         // 1-bit carry-out
);

always @(*) begin
    case (opcode) // choose operation based on opcode
        4'b0000: begin // ADD: opa + opb + cin
            {cout, result} = opa + opb + cin;
        end
        4'b0001: begin // INVERT: ~opa
            result = ~opa;
            cout = 0;
        end
        4'b0010: begin // OR: opa | opb
            result = opa | opb;
            cout = 0;
        end
        4'b0011: begin // AND: opa & opb
            result = opa & opb;
            cout = 0;
        end
        4'b0100: begin
            result = {opa[6:0], 1'b0};
            cout   = 1'b0;
        end
        4'b0101: begin
            result = {1'b0, opa[7:1]};
            cout   = 1'b0;
        end
        4'b0110: begin
            result = {opa[6:0], opa[7]};
            cout   = 1'b0;
        end
        4'b0111: begin
            result = {opa[0], opa[7:1]};
            cout   = 1'b0;
        end
        4'b1001: begin
            result = 8'b0;
            cout = (opa > opb) ? 1'b1 : 1'b0;
        end
        default: begin
            result = 8'b0;
            cout = 0;
        end
    endcase
end

endmodule
```

The sequential part is below.



```verilog
module accumulator (
    input clk,
    input reset,
    input [7:0] d,
    output reg [7:0] q
);
always @(posedge clk or posedge reset)
begin if (reset)
        q <= 8'b0;
    else
        q <= d;
end
endmodule
module top_module (
    input clk,
    input reset,
    input [7:0] opa,
    input [7:0] opb,
    input [3:0] opcode,
    input cin,
    output [7:0] acc_out,
    output cout
);
    wire [7:0] alu_result;
    wire alu_cout;

    alu alu_inst (
        .opa(opa),
        .opb(opb),
        .opcode(opcode),
        .cin(cin),
        .result(alu_result),
        .cout(alu_cout)
    );

    accumulator acc_inst (
        .clk(clk),
        .reset(reset),
        .d(alu_result),
        .q(acc_out)
    );

    assign cout = alu_cout;
endmodule
```

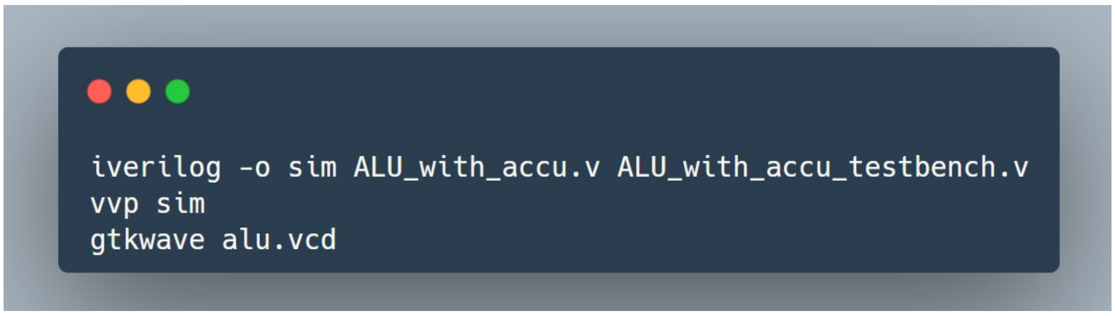
2. The second part is to test my code.



```verilog
`timescale 1ns / 1ps
module testbench;
    reg clk;
    reg reset;
    reg [7:0] opa;
    reg [7:0] opb;
    reg [3:0] opcode;
    reg cin;

    wire [7:0] acc_out;
    wire cout;

    top_module uut (
        .clk(clk),
        .reset(reset),
        .opa(opa),
        .opb(opb),
        .opcode(opcode),
        .cin(cin),
        .acc_out(acc_out),
        .cout(cout)
    );
    initial begin
        clk = 0;
        forever #5 clk = ~clk; // Toggle every 5 ns
    end

    initial begin
        $dumpfile("alu.vcd");
        $dumpvars(0, testbench);

        // Step 1: Reset the accumulator
        // 将 reset 信号置为 1，持续 10 ns，以异步复位累加器
        // 使其输出 acc_out 初始化为 8'h00。
        // 同时，将输入信号设置为 8'h00
        // cin（进位输入）设置为 0
        reset = 1;
        opa = 8'h00;
        opb = 8'h00;
        opcode = 4'b0000;
        cin = 0;
        #10;
        reset = 0;

        // Step 2: ADD 8'h05 + 8'h23
        opa = 8'h05;
        opb = 8'h23;
        opcode = 4'b0000;
        cin = 0;
        #10;

        // Step 3: AND with 8'h56
        opa = acc_out;
        opb = 8'h56;
        opcode = 4'b0011;
        #10;

        // Step 4: ROL on the result
        opa = acc_out;
        opb = 8'h00;
        opcode = 4'b0110;
        #10;

        // Step 5: ADD 8'hA0
        opa = acc_out;
        opb = 8'hA0;
        opcode = 4'b0000;
        cin = 0;
        #10;

        $finish;
    end
endmodule
```

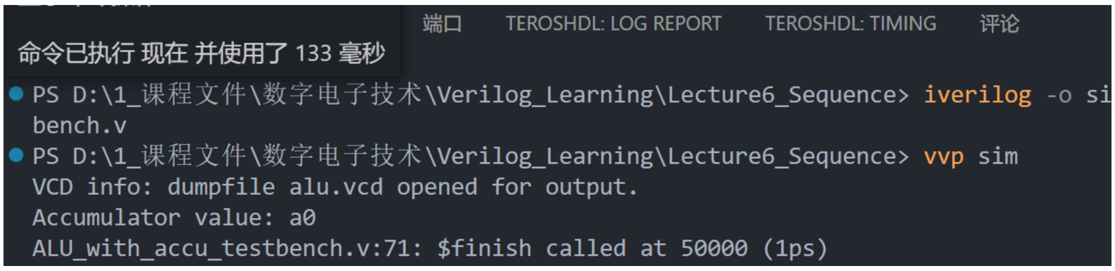
This testbench completes the inspection of the module and implements addition based on the example given in the question. The following is the waveform file and the measured waveform we obtained. Here we use the VSCode plug-in iVerilog and WaveTrace, which can more conveniently display the waveform.

I output this calculated value in the integrated terminal

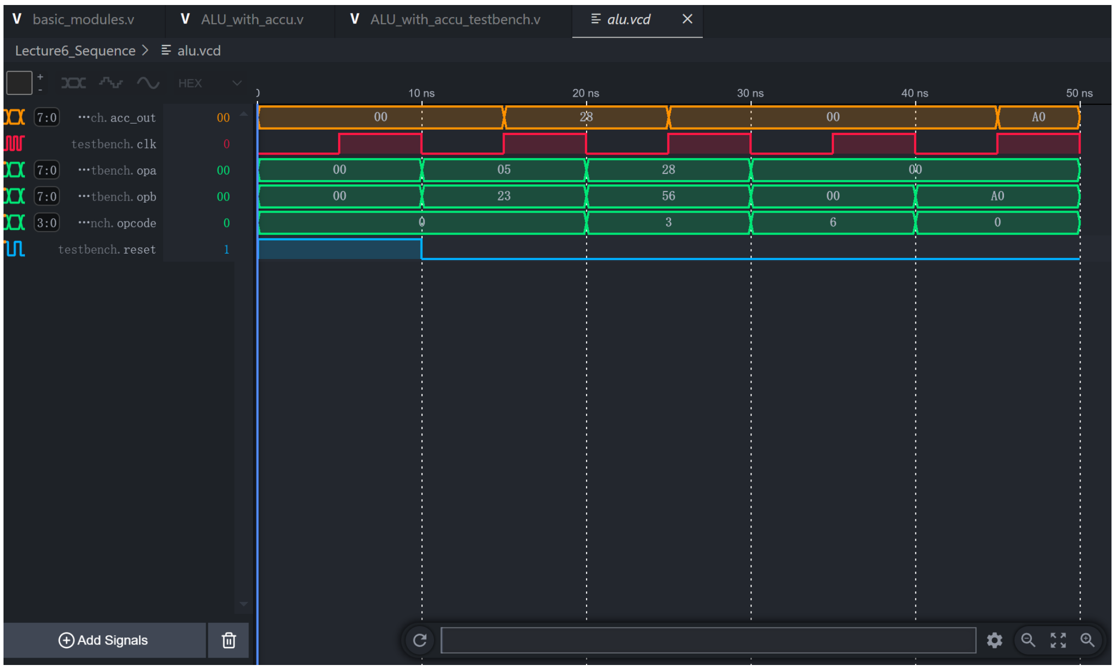


```
iverilog -o sim ALU_with_accu.v ALU_with_accu_testbench.v
vvp sim
gtkwave alu.vcd
```

– the result is 8hA0



```
● PS D:\1_课程文件\数字电子技术\Verilog_Learning\Lecture6_Sequence> iverilog -o si
  bench.v
● PS D:\1_课程文件\数字电子技术\Verilog_Learning\Lecture6_Sequence> vvp sim
  VCD info: dumpfile alu.vcd opened for output.
  Accumulator value: a0
  ALU_with_accu_testbench.v:71: $finish called at 50000 (1ps)
```



basic_modules.v
ALU_with_accu.v
ALU_with_accu_testbench.v
alu.vcd
Lecture6_Sequence > alu.vcd
HEX
10 ns
20 ns
30 ns
40 ns
50 ns
7:0 ···ch. acc_out 00
testbench. clk 0
7:0 ···tbench. opa 00
7:0 ···tbench. opb 00
3:0 ···nch. opcode 0
testbench. reset 1
00 28 00 A0
00 05 28 00
00 23 56 00 A0
0 3 6 0
Add Signals