Richard Ojo

Lab 1

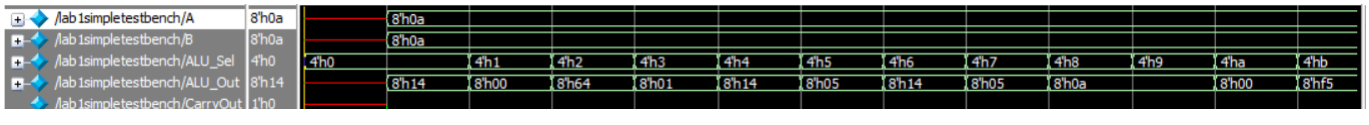
1. Simple TestBench

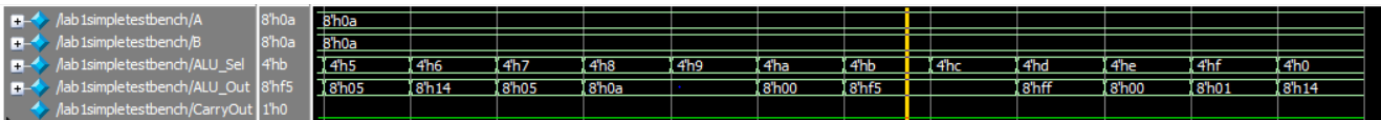
Code:

```
module lab1simpletestbench(); // Testbench has no inputs, outputs
reg [7:0]A, B;
reg [3:0] ALU_Sel; // Will be assigned in initial block
wire [7:0] ALU_Out;
wire CarryOut;
// instantiate device under test
alu dut (.A(A), .B(B), .ALU_Sel(ALU_Sel), .ALU_Out(ALU_Out), .CarryOut(CarryOut) );
// apply inputs one at a time
integer i;
//Select 0 - 15
initial begin // sequential block
	ALU_Sel = 4'b0000;
	A = 8'b00001010; B = 8'b00001010;#10; // apply inputs, wait 10ns
	for (i = 0; i < 16; i = i + 1) begin
		$display("A = %h", A);
		$display("B = %h", B);
		$display("ALU_Sel = %h", ALU_Sel);
		$display("ALU_Out = %h", ALU_Out);
		$display("CarryOut = %h", CarryOut);
		ALU_Sel = ALU_Sel + 4'b0001;#10;
	end
end
endmodule
```

Display Output: *All inputs and outputs are in hexadecimal

```
# A = 0a
# B = 0a
# ALU_Sel = 0
# ALU_Out = 14
# CarryOut = 0
# A = 0a
# B = 0a
# ALU_Sel = 1
# ALU_Out = 00
# CarryOut = 0
# A = 0a
# B = 0a
# ALU_Sel = 2
# ALU_Out = 64
# CarryOut = 0
# A = 0a
# B = 0a
# ALU_Sel = 3
# ALU_Out = 01
# CarryOut = 0
# A = 0a
# B = 0a
# ALU_Sel = 4
# ALU_Out = 14
# CarryOut = 0
# A = 0a
# B = 0a
# ALU_Sel = 5
# ALU_Out = 05
# CarryOut = 0
# A = 0a
# B = 0a
# ALU_Sel = 6
# ALU_Out = 14
# CarryOut = 0
# A = 0a
# B = 0a
# ALU_Sel = 7
# ALU_Out = 05
# CarryOut = 0
# A = 0a
# B = 0a
# ALU_Sel = 8
# ALU_Out = 0a
# CarryOut = 0
# A = 0a
# B = 0a
# ALU_Sel = 9
# ALU_Out = 0a
# CarryOut = 0
# A = 0a
# B = 0a
# ALU_Sel = a
# ALU_Out = 00
# CarryOut = 0
# A = 0a
# B = 0a
# ALU_Sel = b
# ALU_Out = f5
# CarryOut = 0
# A = 0a
# B = 0a
# ALU_Sel = c
# ALU_Out = f5
# CarryOut = 0
# A = 0a
# B = 0a
# ALU_Sel = d
# ALU_Out = ff
# CarryOut = 0
# A = 0a
# B = 0a
# ALU_Sel = e
# ALU_Out = 00
# CarryOut = 0
# A = 0a
# B = 0a
# ALU_Sel = f
# ALU_Out = 01
# CarryOut = 0
```





2. Self-checking test-bench

Code:

```
module testbench();
reg clk, reset; // clock and reset are internal
reg [7:0]A, B, ALU_Expected;
reg [3:0] ALU_Sel; // Will be assigned in initial block
wire [7:0] ALU_Out;
wire CarryOut;
reg CarryOut_Expected;
reg [31:0] vectornum, errors; // bookkeeping variables
reg [28:0] testvectors[15:0];// array of testvectors
// instantiate device under test
alu dut (.A(A), .B(B), .ALU_Sel(ALU_Sel), .ALU_Out(ALU_Out), .CarryOut(CarryOut) );
// generate clock
always begin
        clk = 1; #5; clk = 0; #5; // 10ns period
end

initial begin
        $readmemb("alu.tv", testvectors); // Read vectors
        vectornum = 0; errors = 0; // Initialize
        reset = 1; #27; reset = 0; // Apply reset wait
end
// Note: $readmemh reads testvector files written in
// hexadecimal

// apply test vectors on rising edge of clk
always @(posedge clk) begin
        #1; {ALU_Sel, A, B, ALU_Expected, CarryOut_Expected} = testvectors[vectornum];
end


// check results on falling edge of clk
always @(negedge clk)
        if (~reset) begin
                if (ALU_Out !== ALU_Expected || CarryOut !== CarryOut_Expected) begin
                        $display("Error: inputs = ALU_Sel:%b A:%b B:%b ALU_Out:%b",
ALU_Sel, A, B, ALU_Out);
                        $display(" outputs = ALU_Out:%b ALU_Expected:%b  CarryOut:%b
CarryOut_Expected:%b", ALU_Out,ALU_Expected, CarryOut,CarryOut_Expected);
                        errors = errors + 1;
                end
```

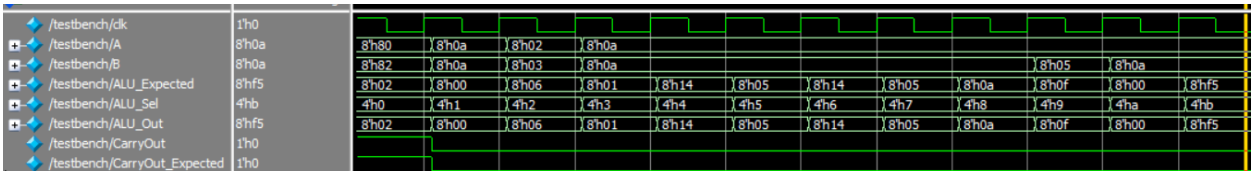
```
            vectornum = vectornum + 1;

            if (testvectors[vectornum] === 29'bx) begin
                $display("%d tests completed with %d errors",vectornum, errors);
                $finish; // End simulation
            end
        end

endmodule
```

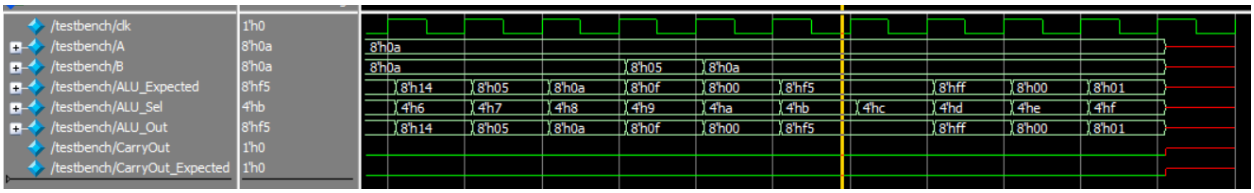
alu.tv:

```
0000_10000000_10000010_00000010_1
0001_00001010_00001010_00000000_0
0010_00000010_00000011_00000110_0
0011_00001010_00001010_00000001_0
0100_00001010_00001010_00010100_0
0101_00001010_00001010_00000101_0
0110_00001010_00001010_00010100_0
0111_00001010_00001010_00000101_0
1000_00001010_00001010_00001010_0
1001_00001010_00000101_00001111_0
1010_00001010_00001010_00000000_0
1011_00001010_00001010_11110101_0
1100_00001010_00001010_11110101_0
1101_00001010_00001010_11111111_0
1110_00001010_00001010_00000000_0
1111_00001010_00001010_00000001_0
```

Display Output: *All inputs and outputs are in hexadecimal

```
VSIM 9> run
#         16 tests completed with          0 errors
# ** Note: $finish    : C:/Modeltech_pe_edu_10.4a/examples/testbench.v(43)
#    Time: 185 ns  Iteration: 1  Instance: /testbench
# 1
# Break in Module testbench at C:/Modeltech_pe_edu_10.4a/examples/testbench.v line 43
```





Part-B: Memory testing using tasks

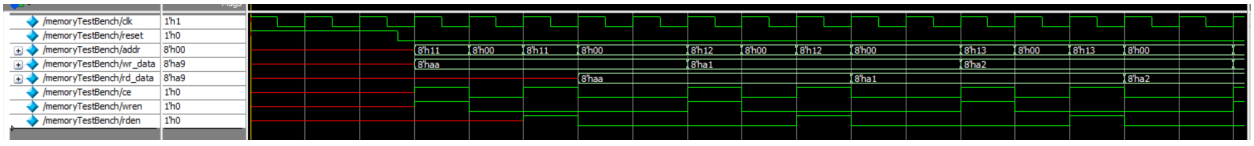
Memory.v

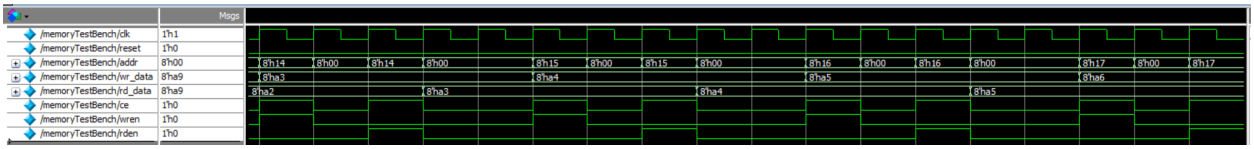
```
module memory (addr, ce, wren, rden, wr_data, rd_data, clk);

input [7:0] addr, wr_data;
output reg [7:0] rd_data;
input ce, wren, rden;
input clk;
reg [7:0] mem [0:255];

always @ (posedge clk)
if (ce)
begin
   if (rden)
        rd_data <= mem[addr];
   else if (wren)
        mem[addr] <= wr_data;
end

endmodule
```

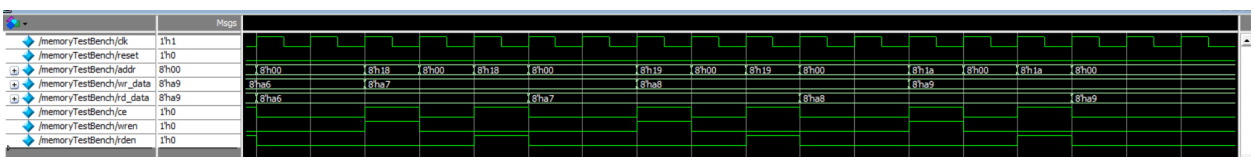
```
module testbench();
reg clk, reset; // clock and reset are internal
reg [7:0]addr, wr_data;
wire [7:0] rd_data;
reg ce, wren, rden;

// instantiate device under test
memory dut (.addr(addr), .ce(ce), .wren(wren), .rden(rden), .wr_data(wr_data),
.rd_data(rd_data), .clk(clk) );

// generate clock
always begin
        clk = 1; #5; clk = 0; #5; // 10ns period
end

// Memory write task
task mem_write;
  input [7:0] address;
  input [7:0] data;
  begin
    $display ("%g Memory Write task with address : %h Data : %h", $time, address,data);
    $display ("%g  -> Driving CE, WREN, WR data and ADDRESS on to bus", $time);
    @ (posedge clk);
// addr, ce, wren, wr_data are the signals connected to the memory model
    addr = address;
    ce = 1;
```

```
    wren = 1;
    wr_data = data;
    @ (posedge clk);
    addr = 0;
    ce = 0;
    wren = 0;
    $display ("======================");
  end
endtask

// Memory Read task
task mem_read;
  input [7:0] address;
  input [7:0] data;
  begin
    $display ("%g Memory Read task with address : %h", $time, address);
    $display ("%g  -> Driving CE, RDEN, RD data and ADDRESS on to bus", $time);
    @ (posedge clk);
// addr, ce, rden, rd_data are the signals connected to the memory model
    addr = address;
    ce = 1;
    rden = 1;
    @ (posedge clk);
    addr = 0;
    ce = 0;
    rden = 0;
    @ (posedge clk);
    data = rd_data;
    $display ("%g Value at address : %h = %h", $time, address, data);
    $display ("======================");
  end
endtask

initial begin
        reset = 1; #27; reset = 0; // Apply reset wait

        #1 mem_write (8'h11, 8'hAA);    // writing 0xAA to the memory address 0x11
        #1 mem_read  (8'h11, rd_data);   // reading from memory address 0x11 to the rd_data
signal

        #1 mem_write (8'h12, 8'hA1);    // writing 0xA1 to the memory address 0x12
        #1 mem_read  (8'h12, rd_data);   // reading from memory address 0x12 to the rd_data
signal

        #1 mem_write (8'h13, 8'hA2);    // writing 0xA2 to the memory address 0x13
```

```
        #1 mem_read  (8'h13, rd_data);   // reading from memory address 0x13 to the rd_data
signal

        #1 mem_write (8'h14, 8'hA3);    // writing 0xA3 to the memory address 0x14
        #1 mem_read  (8'h14, rd_data);   // reading from memory address 0x14 to the rd_data
signal

        #1 mem_write (8'h15, 8'hA4);    // writing 0xA4 to the memory address 0x15
        #1 mem_read  (8'h15, rd_data);   // reading from memory address 0x15 to the rd_data
signal

        #1 mem_write (8'h16, 8'hA5);    // writing 0xA5 to the memory address 0x16
        #1 mem_read  (8'h16, rd_data);   // reading from memory address 0x16 to the rd_data
signal

        #1 mem_write (8'h17, 8'hA6);    // writing 0xA6 to the memory address 0x17
        #1 mem_read  (8'h17, rd_data);   // reading from memory address 0x17 to the rd_data
signal

        #1 mem_write (8'h18, 8'hA7);    // writing 0xA7 to the memory address 0x18
        #1 mem_read  (8'h18, rd_data);   // reading from memory address 0x18 to the rd_data
signal

        #1 mem_write (8'h19, 8'hA8);    // writing 0xA8 to the memory address 0x19
        #1 mem_read  (8'h19, rd_data);   // reading from memory address 0x19 to the rd_data
signal

        #1 mem_write (8'h1a, 8'hA9);    // writing 0xA9 to the memory address 0x1a
        #1 mem_read  (8'h1a, rd_data);   // reading from memory address 0x1a to the rd_data
signal
end
endmodule
```







```
VSIM 52> run
# 28 Memory Write task with address : 11 Data : aa
# 28  -> Driving CE, WREN, WR data and ADDRESS on to bus
# ====================
# 41 Memory Read task with address : 11
# 41  -> Driving CE, RDEN, RD data and ADDRESS on to bus
# 70 Value at address : 11 = aa
# ====================
# 71 Memory Write task with address : 12 Data : a1
# 71  -> Driving CE, WREN, WR data and ADDRESS on to bus
# ====================
# 91 Memory Read task with address : 12
# 91  -> Driving CE, RDEN, RD data and ADDRESS on to bus
# 120 Value at address : 12 = a1
# ====================
# 121 Memory Write task with address : 13 Data : a2
# 121  -> Driving CE, WREN, WR data and ADDRESS on to bus
# ====================
# 141 Memory Read task with address : 13
# 141  -> Driving CE, RDEN, RD data and ADDRESS on to bus
# 170 Value at address : 13 = a2
# ====================
# 171 Memory Write task with address : 14 Data : a3
# 171  -> Driving CE, WREN, WR data and ADDRESS on to bus
# ====================
# 191 Memory Read task with address : 14
# 191  -> Driving CE, RDEN, RD data and ADDRESS on to bus
# 220 Value at address : 14 = a3
# ====================
# 221 Memory Write task with address : 15 Data : a4
# 221  -> Driving CE, WREN, WR data and ADDRESS on to bus
# ====================
# 241 Memory Read task with address : 15
# 241  -> Driving CE, RDEN, RD data and ADDRESS on to bus
# 270 Value at address : 15 = a4
# ====================
# 271 Memory Write task with address : 16 Data : a5
# 271  -> Driving CE, WREN, WR data and ADDRESS on to bus
# ====================
# 291 Memory Read task with address : 16
# 291  -> Driving CE, RDEN, RD data and ADDRESS on to bus
# 320 Value at address : 16 = a5
# ====================
# 321 Memory Write task with address : 17 Data : a6
# 321  -> Driving CE, WREN, WR data and ADDRESS on to bus
# ====================
# 341 Memory Read task with address : 17
# 341  -> Driving CE, RDEN, RD data and ADDRESS on to bus
# 370 Value at address : 17 = a6
# ====================
# 371 Memory Write task with address : 18 Data : a7
# 371  -> Driving CE, WREN, WR data and ADDRESS on to bus
# ====================
# 391 Memory Read task with address : 18
# 391  -> Driving CE, RDEN, RD data and ADDRESS on to bus
# 420 Value at address : 18 = a7
# ====================
# 421 Memory Write task with address : 19 Data : a8
# 421  -> Driving CE, WREN, WR data and ADDRESS on to bus
# ====================
# 441 Memory Read task with address : 19
# 441  -> Driving CE, RDEN, RD data and ADDRESS on to bus
# 470 Value at address : 19 = a8
# ====================
# 471 Memory Write task with address : 1a Data : a9
# 471  -> Driving CE, WREN, WR data and ADDRESS on to bus
# ====================
# 491 Memory Read task with address : 1a
# 491  -> Driving CE, RDEN, RD data and ADDRESS on to bus
# 520 Value at address : 1a = a9
# ====================
```