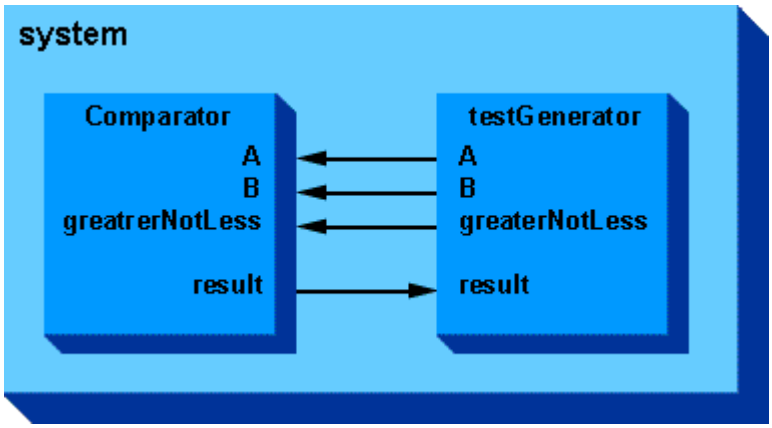
LAB IVERILOG 3

MSI

Version: nOV 28th 2016

Lưu ý là test bench và mã module được viết trong cùng một file .v

I. Comparator



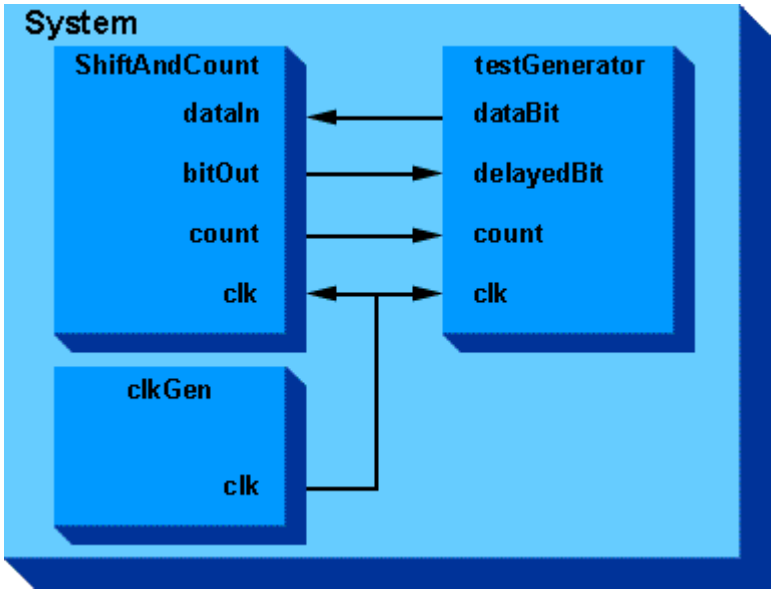

```
module system;
    /* This is a test fixture for
testing a comparator */

    wire greaterNotLess;
// sense of comparison
    wire [15:0] A, B;
// comparand values - 16 bit
    wire result;
// comparison result

    // Module instances
    comparator #(16, 2) comp
(result, A, B, greaterNotLess);
    testGenerator tg (A, B,
greaterNotLess, result);

endmodule

module comparator (result, A, B,
greaterNotLess);
    parameter width = 8;
    parameter delay = 1;
    input [width-1:0] A, B;
// comparands
    input greaterNotLess;
// 1 - greater, 0 - less than
    output result;
// 1 if true, 0 if false

    assign #delay result =
greaterNotLess ? (A > B) : (A <
B);
```

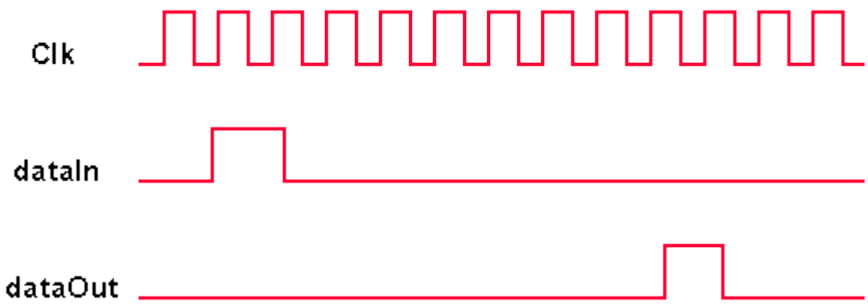
-

```
endmodule


module testGenerator (A, B,
greaterNotLess, result);
    output [15:0] A, B;
    output greaterNotLess;
    input result;
    parameter del = 5;

    reg [15:0] A, B;
    reg greaterNotLess;

    initial begin
// produce test data, check
results
        A = 16'h1234;
        B = 16'b0001001000110100;
        greaterNotLess = 0;
    #del
        check(0);
        B = 0;
        greaterNotLess = 1;
    #del
        check(1);
        A = 1;
        greaterNotLess = 0;
    #del
        check(0);
        $finish;
    end

    task check;
        input shouldBe;
        begin
            if (result !=
shouldBe)
                $display("Error!
%d %s %d, result = %b", A,
greaterNotLess?">":"<",
                            B,
result);
        end
    endtask
 endmodule
```

Biên dịch đoạn code trên và quan sát kết quả.

II. Shift Reg with Counter





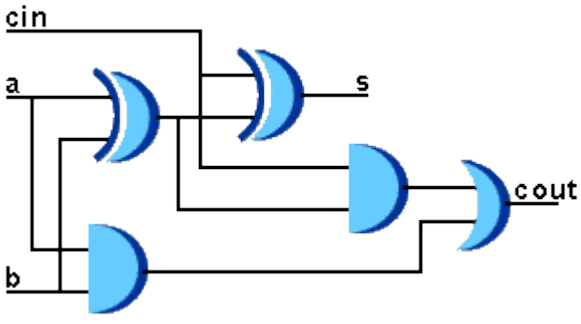

```
module system;
    /* This is a test fixture for
testing a combination shift
register
       and counter */

    wire data, clk;
// nets to connect up the pieces
    wire delayedData;
// data out of the fifo
    wire [31:0] nOnes;
// number of ones contained in
fifo

    // Module instances
    shiftAndCount SandC
(delayedData, nOnes, data, clk);
// shift register
    clkGen #(10) cg (clk);
// generate the clock
    testGenerator tg (data,
delayedData, nOnes, clk);
// create data, check result

endmodule

module shiftAndCount (bitOut,
count, dataIn, clk);
    parameter width = 8;
    output bitOut;
```

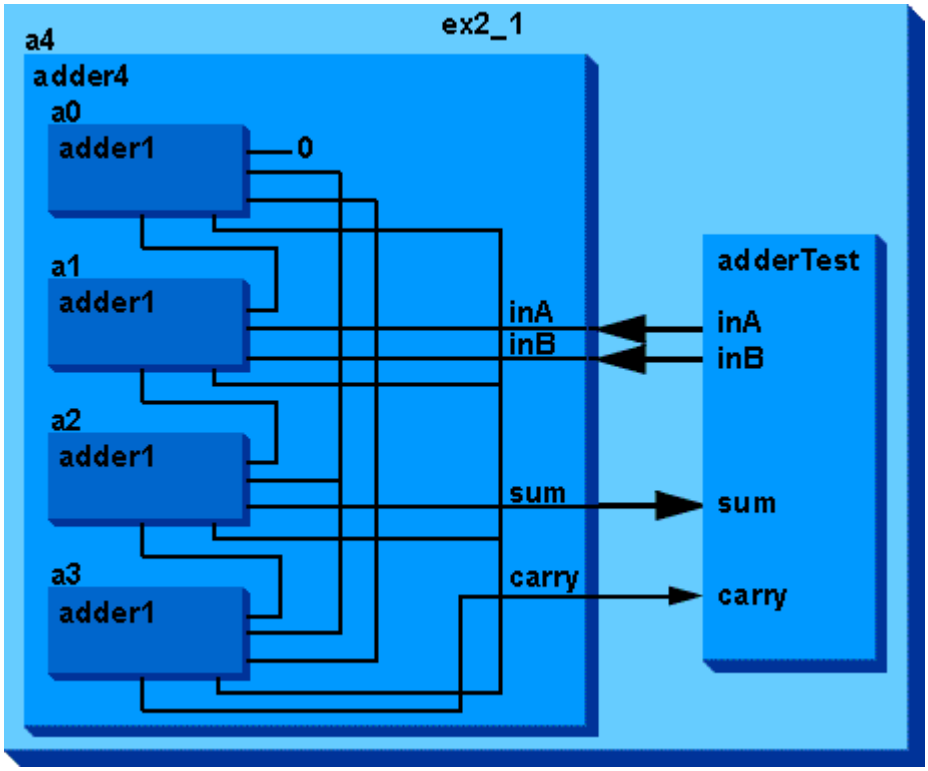
-

```
// data shifted out
    output [31:0] count;
// count of ones
    input dataIn, clk;
// inputs

    integer count;
// the counter
    reg bitOut;
// temporary
    reg [width-1:0] lastBits;
// shift register

    initial begin count = 0;
lastBits = 0; end

    always @(posedge clk) begin
        bitOut = lastBits[width-
1];
        lastBits = (lastBits<<1)
| dataIn;
        if (bitOut > dataIn)
            count = count - 1;
        else
        if (bitOut < dataIn)
            count = count + 1;
    end
endmodule

module clkGen (clk);
    parameter period = 2;
    output clk;
    reg clk;

    initial clk = 0;
// start off with 0
    always
// clock loop
        #(period/2) clk = ~clk;
endmodule

module testGenerator (dataBit,
delayedBit, count, clk);
    output dataBit;
    input delayedBit;
    input [31:0] count;
    input clk;
    reg dataBit;

    task emitBits;  // helper
task to emit n bits
        input [7:0] bits, n;
// task inputs
        begin
            repeat (n) begin
// assume clk is at negedge
                dataBit =
bits[0];  // take just the low
order bit
```

```
                bits = bits >> 1;
            @(negedge clk) ;
            end
// leave at negative edge
        end
    endtask

    task check;
        input bit;
        input [31:0] shouldBe;
        begin
            if (delayedBit !=
bit)
                $display($time,"
delayed bit is %b but should be
%b",

delayedBit, bit);
            if (count !=
shouldBe)
                $display($time,"
Count is %d but should be %d",

count, shouldBe);
        end
    endtask

    initial begin
// produce test data, check
results
        $monitor($time," dataBit:
%b delayedBit: %b", dataBit,
delayedBit);
        emitBits(0, 1);
// take care of first cycle
        emitBits('b10010, 5);
        check(0, 2);
        emitBits('b101101, 6);
        check(0, 5);
        emitBits('b01, 2);
        check(1, 5);
        $stop;
    end
endmodule
```

III. 4-bit adder (gate version)






```
module system;
    /* This is a test fixture for
testing a combination shift
register
       and counter */

    wire data, clk;
// nets to connect up the pieces
    wire delayedData;
// data out of the fifo
    wire [31:0] nOnes;
// number of ones contained in
fifo

    // Module instances
    shiftAndCount SandC
(delayedData, nOnes, data, clk);
// shift register
    clkGen #(10) cg (clk);
// generate the clock
    testGenerator tg (data,
delayedData, nOnes, clk);
// create data, check result

endmodule

module shiftAndCount (bitOut,
count, dataIn, clk);
    parameter width = 8;
    output bitOut;
// data shifted out
    output [31:0] count;
// count of ones
    input dataIn, clk;
// inputs
```

-

```
    integer count;
// the counter
    reg bitOut;
// temporary
    reg [width-1:0] lastBits;
// shift register

    initial begin count = 0;
lastBits = 0; end

    always @(posedge clk) begin
        bitOut = lastBits[width-
1];
        lastBits = (lastBits<<1)
| dataIn;
        if (bitOut > dataIn)
            count = count - 1;
        else
        if (bitOut < dataIn)
            count = count + 1;
    end
endmodule

module clkGen (clk);
    parameter period = 2;
    output clk;
    reg clk;

    initial clk = 0;
// start off with 0
    always
// clock loop
        #(period/2) clk = ~clk;
endmodule

module testGenerator (dataBit,
delayedBit, count, clk);
    output dataBit;
    input delayedBit;
    input [31:0] count;
    input clk;
    reg dataBit;

    task emitBits;  // helper
task to emit n bits
        input [7:0] bits, n;
// task inputs
        begin
            repeat (n) begin
// assume clk is at negedge
                dataBit =
bits[0];  // take just the low
order bit
                bits = bits >> 1;
            @(negedge clk) ;
            end
// leave at negative edge
        end
```

```
    endtask

    task check;
        input bit;
        input [31:0] shouldBe;
        begin
            if (delayedBit !=
bit)
                $display($time,"
delayed bit is %b but should be
%b",

delayedBit, bit);
            if (count !=
shouldBe)
                $display($time,"
Count is %d but should be %d",

count, shouldBe);
        end
    endtask

    initial begin
// produce test data, check
results
        $monitor($time," dataBit:
%b delayedBit: %b", dataBit,
delayedBit);
        emitBits(0, 1);
// take care of first cycle
        emitBits('b10010, 5);
        check(0, 2);
        emitBits('b101101, 6);
        check(0, 5);
        emitBits('b01, 2);
        check(1, 5);
        $stop;
    end
endmodule
```

IV. Practice

1. Thực hiện mạch cộng 16 bits dựa vào module cộng 4 bits đã có.