6/20/2024

HARI PREETH D M

INTERN AT QNU LABS

TASK 2

PISO Shift register using PyVerilog

Task Description:

Using a python script, read a verilog code, extract the inputs, outputs and other variables and generate a testbench for the same verilog code with the extracted inputs and outputs.

**Verilog Code for 8 bit Shift register:**

module piso\_shift\_register (

input  clk,

input  enable,

input  reset,

output  [7:0]shift\_reg,

input  load, // Control signal to load parallel data

input  [7:0] parallel\_in, // 8-bit parallel input

output  serial\_out // Serial output

);

    reg [7:0] shift\_reg1;

    always @(posedge clk ) begin

        if (reset) begin

            shift\_reg1 <= 8'b0;

        end

        else if (enable) begin

        if (load) begin

            shift\_reg1 <= parallel\_in;

        end else begin

            shift\_reg1 <= {shift\_reg1[6:0], 1'b0}; // Shift left and insert a 0 at LSB

        end

        end

    end

    assign serial\_out = shift\_reg1[7]; // MSB is the serial output

    assign shift\_reg =shift\_reg1;

  initial begin

    $dumpfile("dump.vcd");

    $dumpvars(1, piso\_shift\_register);

  end

endmodule

**Python script using Pyverilog:**

import pyverilog.vparser.ast as vast

from pyverilog.vparser.parser import parse

def parse\_verilog(verilog\_file):

    ast, directives = parse([verilog\_file])

    return ast

def get\_module\_info(ast):

    modules = []

    for item in ast.description.definitions:

        if isinstance(item, vast.ModuleDef):

            modules.append(item)

    return modules

def extract\_signals(module):

    inputs = []

    outputs = []

    for port in module.portlist.ports:

        if isinstance(port, vast.Ioport):

            port\_name = port.first.name

            width = 1  # Default width is 1 bit

            if port.first.width:

                msb = int(port.first.width.msb.value)

                lsb = int(port.first.width.lsb.value)

                width = msb - lsb + 1

            if isinstance(port.first, vast.Input):  # Remove clk from inputs

                inputs.append((port\_name, width))

            elif isinstance(port.first, vast.Output):

                outputs.append((port\_name, width))

    return inputs, outputs

def generate\_testbench(module, inputs, outputs):

    module\_name = module.name

    # Create the input and output declarations

    input\_declarations = "\n    ".join(f"reg {'[{}:0] '.format(width-1) if width > 1 else ''}{signal};" for signal, width in inputs)

    output\_declarations = "\n    ".join(f"wire {'[{}:0] '.format(width-1) if width > 1 else ''}{signal};" for signal, width in outputs)

    # Create the port connections

    port\_connections = ",\n        ".join(f".{signal}({signal})" for signal, \_ in inputs + outputs)

    # Generate test vectors (toggle each input signal once for demonstration)

    test\_vectors = ""

    toggle\_time = 10  # Time interval for toggling signals

    for \_ in range(3):  # Three iterations

        for signal, \_ in inputs:

            #test\_vectors += f"        {signal} = 1;\n"

            #test\_vectors += f"        #{toggle\_time} {signal} = 0;\n"

            test\_vectors += f"        #{toggle\_time} {signal} = 0;\n"

        test\_vectors += "\n\n"  # Add space (new line) between each iteration

    # Define the template for the testbench

    tb\_template = f"""

module {module\_name}\_tb;

    // Inputs

    {input\_declarations}

    // Outputs

    {output\_declarations}

    // Instantiate DUT (Design Under Test)

    {module\_name} dut (

        .clk(clk),

        {port\_connections}

    );

    // Initial Block

    initial begin

        clk = 0;

        reset = 1;

        #5 reset = 0;

        // Automated test vectors

{test\_vectors}

        #100;  // Simulation end time

        $finish;

    end

    // Clock Generation

    always #5 clk = ~clk;

endmodule

"""

    return tb\_template

def save\_testbench(tb\_content, tb\_filename):

    with open(tb\_filename, 'w') as tb\_file:

        tb\_file.write(tb\_content)

def save\_signals(inputs, outputs, filename):

    with open(filename, 'w') as sig\_file:

        sig\_file.write("Inputs:\n")

        for signal, width in inputs:

            sig\_file.write(f"{signal} [{width} bits]\n")

        sig\_file.write("\nOutputs:\n")

        for signal, width in outputs:

            sig\_file.write(f"{signal} [{width} bits]\n")

if \_\_name\_\_ == "\_\_main\_\_":

    verilog\_file = '/workspaces/Qnu\_Labs/Parse\_verilog/piso\_shift\_register.v'  # Update this path to your Verilog file

    tb\_filename = 'generated\_testbench\_tb.v'

    sig\_filename = 'extracted\_signals.txt'

    ast = parse\_verilog(verilog\_file)

    modules = get\_module\_info(ast)

    for module in modules:

        inputs, outputs = extract\_signals(module)

        tb\_content = generate\_testbench(module, inputs, outputs)

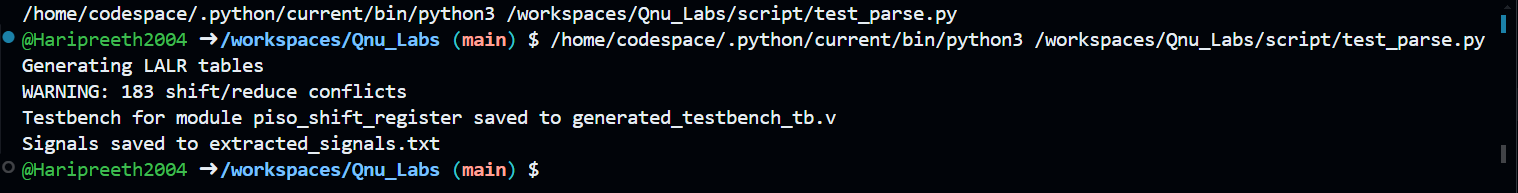
        save\_testbench(tb\_content, tb\_filename)

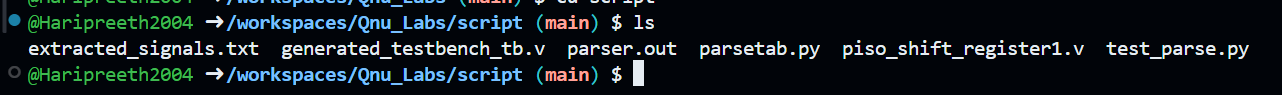
        save\_signals(inputs, outputs, sig\_filename)

        print(f"Testbench for module {module.name} saved to {tb\_filename}")

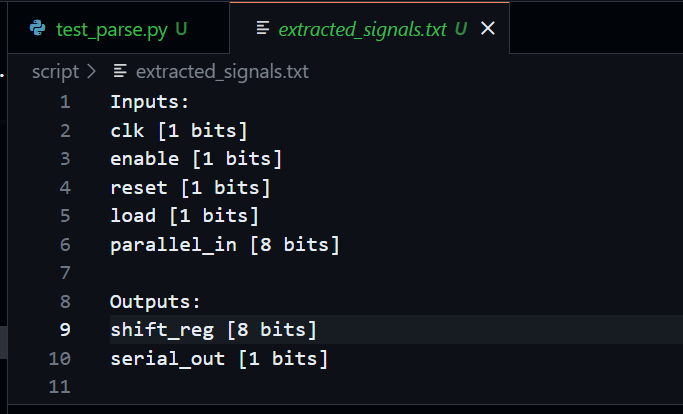
        print(f"Signals saved to {sig\_filename}")

**After run the python file outputs:**





**Extracted output text file:**



**Generated testbench using**

module piso\_shift\_register\_tb;

    // Inputs

    reg clk;

    reg enable;

    reg reset;

    reg load;

    reg [7:0] parallel\_in;

    // Outputs

    wire [7:0] shift\_reg;

    wire serial\_out;

    // Instantiate DUT (Design Under Test)

    piso\_shift\_register dut (

        .clk(clk),

        .clk(clk),

        .enable(enable),

        .reset(reset),

        .load(load),

        .parallel\_in(parallel\_in),

        .shift\_reg(shift\_reg),

        .serial\_out(serial\_out)

    );

    // Initial Block

    initial begin

        clk = 0;

        reset = 1;

        #5 reset = 0;

        // Automated test vectors

        #10 clk = 0;

        #10 enable = 0;

        #10 reset = 0;

        #10 load = 0;

        #10 parallel\_in = 0;

        #10 clk = 0;

        #10 enable = 0;

        #10 reset = 0;

        #10 load = 0;

        #10 parallel\_in = 0;

        #10 clk = 0;

        #10 enable = 0;

        #10 reset = 0;

        #10 load = 0;

        #10 parallel\_in = 0;

        #100;  // Simulation end time

        $finish;

    end

    // Clock Generation

    always #5 clk = ~clk;

endmodule