Interview Zen

Shayan Taheri

Total duration: 0:46

Question 1

```
//******************************************************************************
//   Feel free to write this code in your favorite IDE and cut and paste
//   your final code into interviewzen.com. Your program will need to be
//   able to read this file.
//
//   Instructions:
//   1. Copy and paste the contents of this problem into a file called
//      TopCell.v. These comments can be included in the file.
//   2. You will be writing a Python program that reads this file and
//      produces output as described below.
//   3. Your program can simply hardcode the path to this file, or
//      optionally provide command line arguments to specify the file
//
//
//
//   This is a simple verilog netlist. It contains several module definitions.
//   Each module contains input and output pins.
//   Some modules have wires.
//
//   Each module also has instances of other modules or primitives.
//   The difference between a module and a primitive is that the primitives
//   do not have "module" definitions in this file. Primitives do not contain
//   instances (cellA is a module and contains instances, invN1 is a primitive
//   and doesn't contain any instances. It is the lowest level of hierarchy).
//
//   Key words:
//      module, endmodule
//      output, input
//      wire
//
//   Instance line description:
//   ==========================
//   bufferCell IbufferCellK0 (.in(in1), .out(net17));
//   bufferCell = module name
//   IbufferCellK0 = instance name (must be unique)
//   .in1(in1) = Pin in1 on instance IbufferCellK0 of module bufferCell
//               connects to a net named in1
//   .out(net17) = Pin out on instance IbufferCellK0 of module bufferCell
//               connects to a net named net17
//
//
//   Problem description:
//   ====================
//      cellB contains:
//          4 instances of bufferCell
//          2 instances of invN1 (a primitive)
//          2 instances of nor2N1 (a primitive)
//          1 instance of nand2N1 (a primitive)
//      bufferCell contains:
//          2 instances of invN1 (a primitive)
//      This means that in the hierarchy of cellB, there are 10 instances
//      of invN1.
//
//      Write a Python program to parse this file and store it in some kind
//      of structure. Assume that all information in the netlist (except
//      comments) may someday need to be accessed.
//      Write a function or method to count the total number of instances of
//      each module/primitive in the "TopCell" hierarchy (or any module)
//      by reading your structure.
//
//      Example output for counting all of the instances in "cellB" hierarchy:
//      nand2N1         : 1 placements
//      nor2N1          : 2 placements
//      invN1           : 10 placements
//      bufferCell      : 4 placements
//
//      Tips:
//          If you are having trouble parsing the file with the instance
//          definitions split across multiple lines, then edit the input
//          so the entire statement is on one line.
//          Example:
```

```
//
//              cellA IcellAK0 (.in1(inA), .in2(inB), .in3(inC), .in4(inD),
//                  .out1(net22));
//              cellA IcellAK0 (.in1(inA), .in2(inB), .in3(inC), .in4(inD), .out1(net22));
//
//
//********************************************************************************

// Cell: designLib TopCell schematic
// Last modified: Jun 14 12:01:37 2017
module TopCell (data, out1, out2, out3, out4, out5);
    output out1;
    output out2;
    output out3;
    output out4;
    output out5;
    input  [7:0] data;

    wire [7:0] data;

    cellA IcellAK0 (.in1(data[7]), .in2(data[6]), .in3(data[5]),
        .in4(data[4]), .out1(net14));
    cellC IcellCK0 (.inA(data[3]), .inB(data[2]), .inC(data[1]),
        .inD(data[0]), .outA(out2), .outB(out3));
    cellD IcellDK0 (.data(data[7:0]), .out1(out4), .out2(out5));
    invN1 X0 (.A(net14), .Y(out1));

endmodule // TopCell

// Cell: designLib bufferCell schematic
// Last modified: Jun 14 11:00:42 2017
module bufferCell (in, out);
    output out;
    input  in;

    invN1 X0 (.A(in), .Y(net5));
    invN1 X1 (.A(net5), .Y(out));
endmodule // bufferCell


// Cell: designLib cellA schematic
// Last modified: Jun 14 10:58:49 2017
module cellA (in1, in2, in3, in4, out1);
    output out1;
    input  in1;
    input  in2;
    input  in3;
    input  in4;

    invN1 X0 (.A(in1), .Y(net13));
    invN1 X1 (.A(in2), .Y(net11));
    invN1 X2 (.A(in3), .Y(net12));
    invN1 X3 (.A(in4), .Y(net14));
    nand4N1 X4 (.A(net13), .B(net11), .C(net12), .D(net14), .Y(out1));
endmodule // cellA


// Cell: designLib cellB schematic
// Last modified: Jun 14 11:09:05 2017
module cellB (in1, in2, in3, in4, out1, out2);
    output out1;
    output out2;
    input  in1;
    input  in2;
    input  in3;
    input  in4;

    bufferCell IbufferCellK0 (.in(in1), .out(net17));
    bufferCell IbufferCellK1 (.in(in2), .out(net16));
    bufferCell IbufferCellK2 (.in(in3), .out(net15));
    bufferCell IbufferCellK3 (.in(in4), .out(net14));
    invN1 X0 (.A(net13), .Y(out2));
    invN1 X1 (.A(net19), .Y(net18));
    nand2N1 X2 (.A(net17), .B(net16), .Y(net19));
    nor2N1 X3 (.A(net15), .B(net14), .Y(net13));
    nor2N1 X4 (.A(net18), .B(net13), .Y(out1));
endmodule // cellB


// Cell: designLib cellC schematic
// Last modified: Jun 14 11:16:40 2017
module cellC (inA, inB, inC, inD, outA, outB);
    output outA;
    output outB;
    input  inA;
    input  inB;
    input  inC;
    input  inD;

    cellA IcellAK0 (.in1(inA), .in2(inB), .in3(inC), .in4(inD),
        .out1(net22));
    cellA IcellAK1 (.in1(inD), .in2(inA), .in3(inB), .in4(inC),
```

```
        .out1(net18));
    cellA IcellAK2 (.in1(inC), .in2(inD), .in3(inA), .in4(inB),
        .out1(net17));
    cellA IcellAK3 (.in1(inB), .in2(inC), .in3(inD), .in4(inA),
        .out1(net21));
    invN1 X0 (.A(net22), .Y(net20));
    invN1 X1 (.A(net21), .Y(net19));
    nand2N1 X2 (.A(net20), .B(net18), .Y(outA));
    nand2N1 X3 (.A(net17), .B(net19), .Y(outB));
endmodule // cellC


// Cell: designLib cellD schematic
// Last modified: Jun 14 11:34:42 2017
module cellD (data, out1, out2);
    output out1;
    output out2;
    input  [7:0] data;

    wire [7:0] data;

    cellB IcellBK0 (.in1(data[0]), .in2(data[1]), .in3(data[2]),
        .in4(data[3]), .out1(net10), .out2(net8));
    cellB IcellBK1 (.in1(data[4]), .in2(data[5]), .in3(data[6]),
        .in4(data[7]), .out1(net7), .out2(net9));
    cellC IcellCK0 (.inA(net10), .inB(net8), .inC(net7), .inD(net9),
        .outA(out1), .outB(out2));
endmodule // cellD
```

```python
# Author: Dr. Shayan Taheri.
# File Content: The Python program for parsing a Verilog netlist and ...
# ... counting the number of instances of each module.

import re
import sys

# Function for parsing a Verilog netlist and ...
# ... positioning the extracted information into a dictionary/structure.

def parse_verilog(filename):
    with open(filename, 'r') as f:
        data = f.read()

    modules = re.findall(r'module ([a-zA-Z0-9_]+) \(', data)
    module_data = {}
    for module in modules:
        module_section = re.search(r'module ' + module + r'[^;]*;(.*?)endmodule', data, re.DOTALL).group(1)
        inputs = re.findall(r'input ([a-zA-Z0-9_\[\]:]+);', module_section)
        outputs = re.findall(r'output ([a-zA-Z0-9_\[\]:]+);', module_section)
        wires = re.findall(r'wire ([a-zA-Z0-9_\[\]:]+);', module_section)
        regs = re.findall(r'reg ([a-zA-Z0-9_\[\]:]+);', module_section)
        instances = re.findall(r'([a-zA-Z0-9_]+) ([a-zA-Z0-9_]+) \(', module_section)
        instance_data = {}
        for inst_module, inst_name in instances:
            inst_ports = re.findall(r'\.([a-zA-Z0-9_]+)\(([a-zA-Z0-9_\[\]:]+)\)', module_section)
            instance_data[inst_name] = {
                'module': inst_module,
                'input_ports': {port_name: net_name for port_name, net_name in inst_ports if port_name in inputs},
                'output_ports': {port_name: net_name for port_name, net_name in inst_ports if port_name in outputs}
            }

        module_data[module] = {
            'Inputs': inputs,
            'Outputs': outputs,
            'Wires': wires,
            'Registers': regs,
            'Instances': instance_data
        }

    return module_data

# Function for counting the number of instances of each module and primitive.

def count_instances(module_data, start_module):
    count = {}
    instances = module_data[start_module]['Instances']

    for inst_name, inst_details in instances.items():
        if inst_details['module'] in count:
            count[inst_details['module']] += 1
        else:
            count[inst_details['module']] = 1

        if inst_details['module'] in module_data:
            sub_counts = count_instances(module_data, inst_details['module'])
            for sub_module, sub_count in sub_counts.items():
                if sub_module in count:
                    count[sub_module] += sub_count
                else:
                    count[sub_module] = sub_count
```

```
    return count

# The main function for running the program.

if __name__ == "__main__":
    filename = sys.argv[1] # Assuming the filename is the first command-line argument.
    module_data = parse_verilog(filename)
    print("Modules found:", list(module_data.keys()))
    selected_module = input("Select a module: ")

    if selected_module not in module_data:
        print("Module not found!")
    else:
        counts = count_instances(module_data, selected_module)
        print(f"In the {selected_module} module:")
        for mod, cnt in counts.items():
            print(f"There are {cnt} instances of {mod}")

# The end of the program.
```

1x 2x 5x 0:00 / 0:46