Logic CAD of VLSI Design - 0460880

Topic: Hierarchical Connectivity Model - HCM

Introduction

Almost all CAD tools require an efficient and clear data model to represent the design being analyzed, synthesized, or verified by the tool. In this course, we provide a reference design of such a hierarchical "folded" connectivity model that aims to clarify the concepts learned in the class and provide services like Gate Level Verilog parser which should ease the development of algorithms without the need for recoding the platform.

In this assignment

1. You will learn about the hierarchical connectivity model (HCM) and get familiar with the model functionality in exercise 1.
2. You will implement a "ranking" algorithm that requires a traversal of the HCM in exercise 2.

Detailed Tasks

1. Follow the Virtual Machine (VM) workshop available on Moodle and setup your VM.
2. Refer to **Task 6, Stages A and B**, for details on how to run the code and execute the tests yourself. You will use these commands after writing your code to test it.
3. Write a program that allows you to answer questions 4.a to 4.f (**60%**).
   These questions are also included as comments inside the **HW1ex1.cc** file. Each comment explains which variable you should assign your answer to. The program will automatically handle the printing needed to test your code.
   For example, according to the comment below, the answer to question 4.a should be assigned to the variable **topLevelNodeCounter**:
   /* assign your answer for section a to topLevelNodeCounter */

   Note that both the hierarchical and flattened versions of the top cell are provided. You may implement additional helper functions as needed.

   NOTE: VDD and VSS are considered global nodes.

4. answers the following questions in file **HW1ex1.cc** :
   a. How many nodes exist in the folded top-level cell? Note, VSS and VDD are global node, hence should be excluded.
   b. How many instances exist in the folded top-level cell?
   c. How many instances of the cell "nand" exist in **cells** of the folded model? Don't count "nand" instances that are contained within other instances.
   d. How many instances of cell "nand" exist in the entire hierarchy (means the number of "nand"s that are needed for full implementation of the top cell)? It is recommended to use here the flat model cell.
   e. How many levels of hierarchy traverse the top cell node with the deepest reach? Reach refers to the number of hierarchical levels a traversal moves through, starting from a specific node and continuing until it reaches a cell that contains no further instances.
      For example, a top cell with no instances in it will have reach of 1. A cell with node connected to one instance will have reach of 2.
   f. What are the **hierarchical names** of the deepest nodes, i.e. the nodes that are in the lowest cells levels. Order the node names lexicographically. assign your answer for section f to the given

list in the code. Remember VDD and VSS are globally connected, i.e. no need to traverse through those nodes.

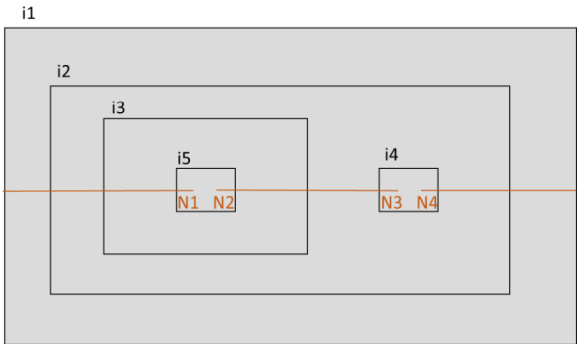


Figure 1 Example – the hirarchical names of the deepest nodes are: i1/i2/i3/i5/N1 and i1/i2/i3/i5/N2

5. Implement "max rank" algorithm on a flattened design (**40%**).
Read the **HW1ex2.cc** file carefully and write your answers in the designated places.

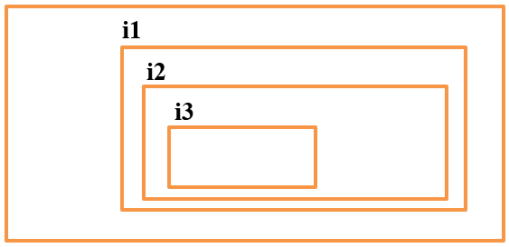
Notice to insert the answer to the relevant variable, the program will manage the printing to test your code automatically.

A maximal ranking algorithm provides each occurrence instance its ***maximal distance*** from any of the **input** ports of the blocks. Use the example below for a better understanding.

NOTE: "VDD" and "VSS" are considered global nodes and are not used to traverse in the graph.

The program starts at the inputs of the given top cell and provides one line of output for every occurrence instance found.
The output should be inserted into the given vector in the code, such that each element in the vector is of type $pair < int, string >$ - **the rank followed by the occurrence instance name**.
The names of occurrence instances are a concatenation of all their parent instance name from top to bottom.

E.g. **i1/i2/i3** is the name of an occurrence instance describe in the following drawing:



NOTE: The order of the element in the output vector will be according to their rank and their lexicographic name - first according to the rank, and for the same rank – lexicographically. example in the next page.

The run time (and complexity) should be linear in relation to the number of instance ports.

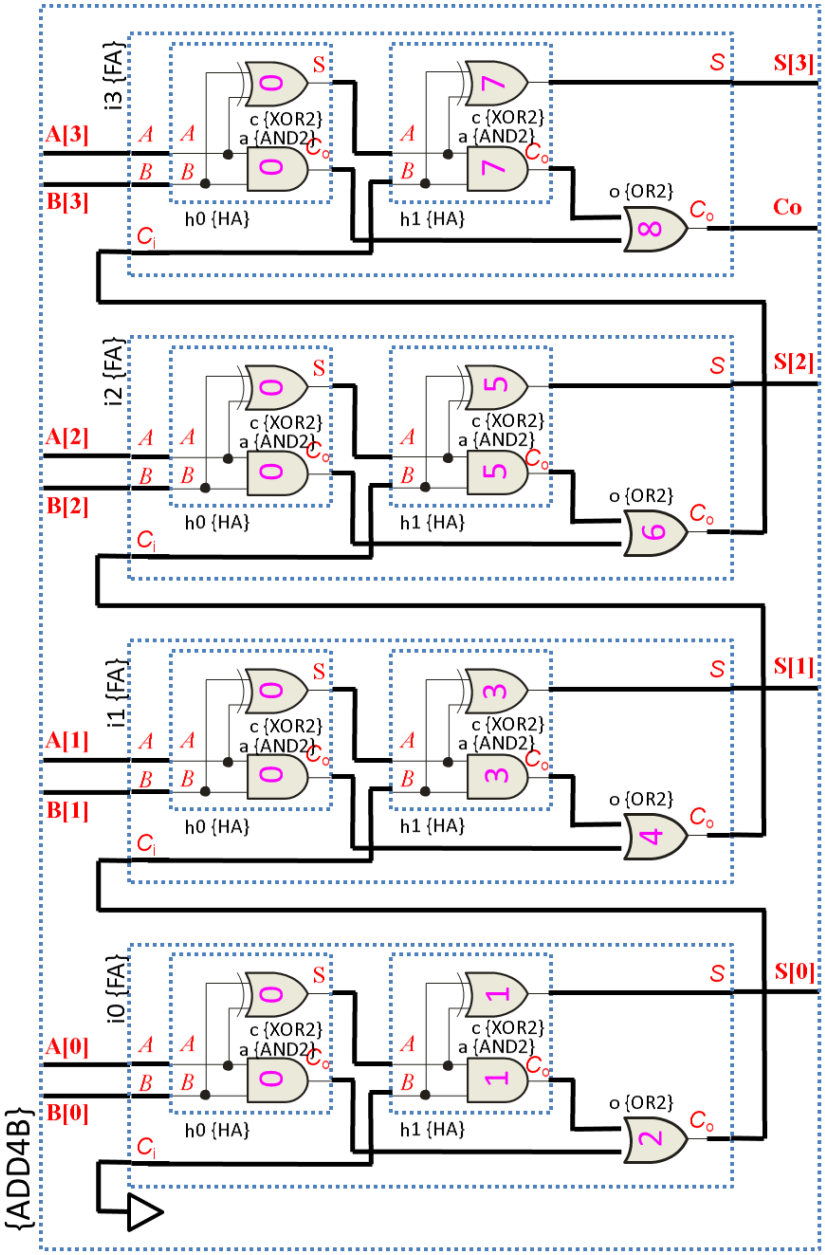
All your code must include sufficient documentation.

Note: To achieve linear complexity, consider using a levelization algorithm and avoid unnecessary updates to instance levels during traversal.

Example Circuit and Ranks file

The circuit described below can be used for understanding the ranks files format.

The Verilog is also provided.



ADD4B.ranks:

```
0 i0/h0/a
0 i0/h0/c
0 i1/h0/a
0 i1/h0/c
0 i2/h0/a
0 i2/h0/c
0 i3/h0/a
0 i3/h0/c
1 i0/h1/a
1 i0/h1/c
2 i0/o
3 i1/h1/a
3 i1/h1/c
4 i1/o
5 i2/h1/a
5 i2/h1/c
6 i2/o
7 i3/h1/a
7 i3/h1/c
8 i3/o
```

adder_4b.v:

```
module or2 (A, B, Y );
  input A, B;
  output Y;
endmodule

module and2 (A, B, Y );
  input A, B;
  output Y;
endmodule

module xor2 (A, B, Y );
  input A, B;
  output Y;
endmodule

module HA (A, B, S, Co );
  input A, B;
  output S, Co;
  and2 a ( .A(A), .B(B), .Y(S) );
  xor2 c ( .A(A), .B(B), .Y(Co) );
endmodule // HA

module FA (A, B, Ci, S, Co );
  input A, B, Ci;
  output S, Co;
  wire  Sab, Cab, Csc;
  HA h0 ( .A(A),  .B(B),  .S(Sab), .Co(Cab) );
  HA h1 ( .A(Sab), .B(Ci), .S(S),  .Co(Csc) );
  or2 o ( .A(Cab), .B(Csc), .Y(Co) );
endmodule // FA

module ADDER4B (A, B, S, Co);
  input [3:0] A;
  input [3:0] B;
  output [3:0] S;
  output      Co;
  wire        C0, C1, C2;
  FA i0 ( .A(A[0]), .B(B[0]), .Ci(1'b0), .S(S[0]),
.Co(C0) );
  FA i1 ( .A(A[1]), .B(B[1]), .Ci(C0),   .S(S[1]),
.Co(C1) );
  FA i2 ( .A(A[2]), .B(B[2]), .Ci(C1),   .S(S[2]),
.Co(C2) );
  FA i3 ( .A(A[3]), .B(B[3]), .Ci(C2),   .S(S[3]),
.Co(Co) );
endmodule // ADDER4B
```

6.

Stage A – Build and write your own programs with HCM

1. **cd HCM**
2. **make** – not required if you followed the PDF on moodle with the instructions for downloading the Virtual Machine
3. **cd wet01**
4. Write your code in HW1ex1.cc and HW1ex2.cc files.
   Notice to insert the answer to the relevant variable, the program will manage the printing to test your code automatically.
5. **make**
   NOTE: the tests for this exercise are automatic!
   If your submission will not compile on the virtual machine – it will be grated ZERO!

Stage B – Test your own programs with HCM

After finishing Stage A, make sure you run the following command from wet01 directory.

Now you are ready to generate the required output files for self-testing before submitting your work.

To generate **TopLevel####.stat** and **TopLevel####.rank** files run the following line in the Virtual machine, in "wet01" directory.

- **./gl_stat TopLevel#### stdcell.v c####high.v**
- **./gl_rank TopLevel#### stdcell.v c####high.v**

We provided you the files **TopLevel####.stat** and **TopLevel####.rank** to compare with your outputs.

Stage C – Create your submission

After finishing Stage B, make sure your wet01 directory include only the .cc files.
Run the following command from wet01/ directory -

1. **cd ..**
2. **tar czf wet01_<id1>_<id2>.tar.gz wet01**
   (<id1> and <id2> are the id numbers of the students)
3. Upload wet01_<id1>_<id2>.tar.gz to Moodle
   (Example of the file name: wet01_123456789_147852369.tar.gz)

- Make sure that only the .cc files are in the folder !

Note: Bold lines are execution commands for the Linux environment (LUX).

Topic	Hierarchical Connectivity Model - HCM
Submission date	
Exercise owner	Gilad Zilberman – zilberman@campus.technion.ac.il
Given Code files	**HW1_ex1.cc** **HW1_ex2.cc** **Makefile**
Given Input Verilog files	**c1355high.v** **c2670high.v**
Given Extra Verilog files	stdcell.v
Given Output files for self-testing	**TopLevel1355.stat** **TopLevel1355.rank** **TopLevel2670.stat** **TopLevel2670.rank**
Files to submit	**HW1_ex1.cc** **HW1_ex2.cc**
Submission format	**wet01_<id1>_<id2>.tar.gz**
NOTE: the tests for this exercise are automatic! wrong submission format will be grated ZERO!	

Questions about this WET exercise should be posted on Moodle. You required to follow the answers of the course staff and write your code according to the answers.

Good luck!