**SPEF (Standard Parasitic Extraction Format) Extractor**

**Project 2 \_ Digital Design II**

Salma Talaat

Youssef Amir

Dina Usama

**Table of Contents**

**Project description …………………………………………………..……...3**

**Milestone 1…………………………………………………………………...4**

Lef

Def

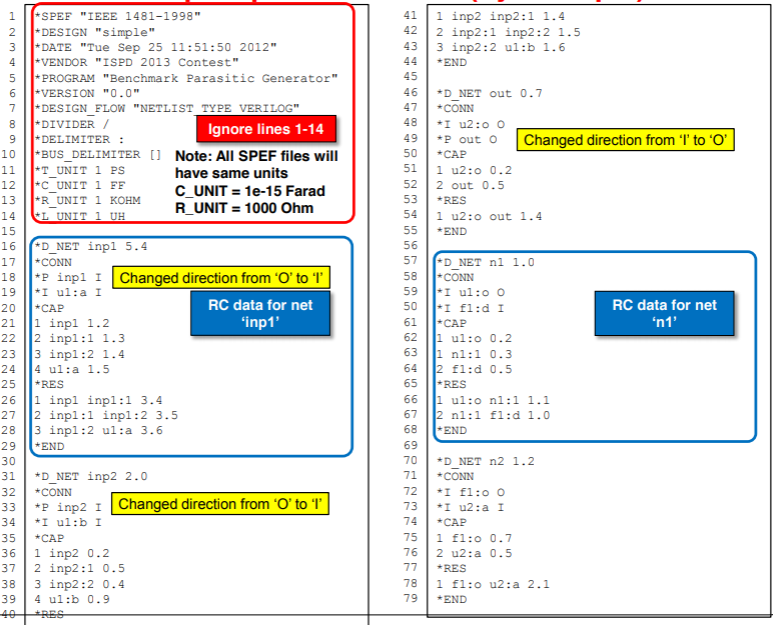
**Milestone 2…………………………………………………………………...4**

**Limitations…………………………………………………………………...5**

**Project Description**

The aim of this project is to create a SPEF file (Standard Parasitic Extraction Format). Standard Parasitic Exchange Format (SPEF) is an IEEE standard for representing parasitic data of wires in a chip in ASCII format. Non-ideal wires have parasitic resistance and capacitance that are captured by SPEF. These wires also have inductance that is not included in SPEF.

SPEF is extracted after routing in Place and route stage. This helps in accurate calculation of IR-drop analysis and other analysis after routing. This file contains the R and C parameters depending on the placement of our tile/block and the routing among the placed cells.



The above image shows an example of an expected SPEF file.

**SPEF Parts**

**1)Name Mapping**

Lists the names of all the components in the net and gives them numbers as an abbreviation

**2) Port listing**

Lists the ports (associated pins to the components) along with their direction

**3)Connections**

<pin type ( port /internal) > <name (component and pinname )> <direction ( input, output, bidirectional) >

**4) Capacitance section**

<cap number> <node> <value>

Cap number: identifying number

Nodes: could be internal or actual ports of components ( number of internal nodes will be determined from the parsing of the routing section of the DEF file)

Value: determined by the CPERSQR \* L \* W + Edge Capacitance \*L (length of wire calculated from the routing section ) This value is from node to ground

**5) Resistance section**

<resistance number> <node 1> <node 2> <value>

Resistance number: identifying number

Node1 and node 2: could be internal or actual ports of components ( number of internal nodes will be determined from the parsing of the routing section of the DEF file) . These nodes sandwich the resistance

Value: determined by the RPERSQR \* (L /W) (length of wire calculated from the routing section )

**Milestone 1**

For our first milestone, we mainly focused on parsing all the input files (the LEF and the DEF files) instead of using the parsers provided online.

1. **LEF**

The lef file provided us with info related to the individual macros of all the possible cells that can be used. The relevant info included all the pins associated with each macro and their directions. It also provides us with the resistance and capacitance per sheet for the different metal layers

1. **DEF**

The def file provided us with the details of the circuit itself.

1. This includes all the components in the circuit,

A- Nicknames,

b- Macro names,

c- Positions on the grid.

1. Netlist:

These are a set of all the nets in the circuit along with their associated components ( each component with the corresponding pin) that form the net with the other components and the routing of all the wires connecting these pins in the net.

**Milestone 2**

For our last milestone, we worked on outputting the SPEF files using the parsers we built on the first milestone

1.Create function for calculating the length of the wire and the node it connects using the components and routing parsed from the DEF.

2.Create a function that calculates the new coordinates of the rects taking into consideration the different orientations on the grid to check for intersections between wires and pins of components

3. From the length of the wire, the capacitance and resistance will be calculated based on the aforementioned equations using the capacitance data and resistance data from the LEF file.

4.Once this is done we will just output to the SPEF file

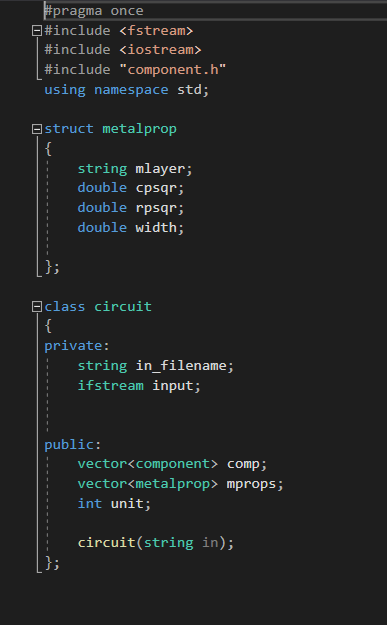
**Main Classes**

1) pin.h

2) component.h

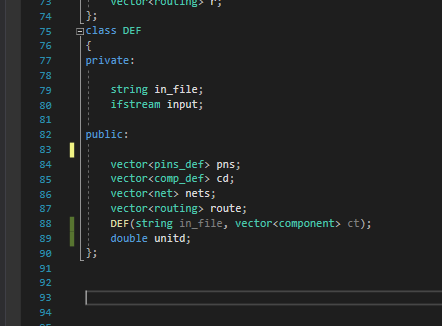
3) Circuit.h

This class is responsible for parsing the LEF file and it stores all the metal properties and all the info related to each component along with their pins. It stores info related to components in vector <component> ct.



4) Def.h

This is the main class that parses the DEF file to store all their components along with their positions, ports, and nets. It uses vector<component> ct to allow the compilation of all the info needed in one data structure.

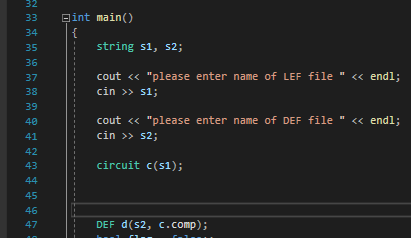


5) main22.cpp

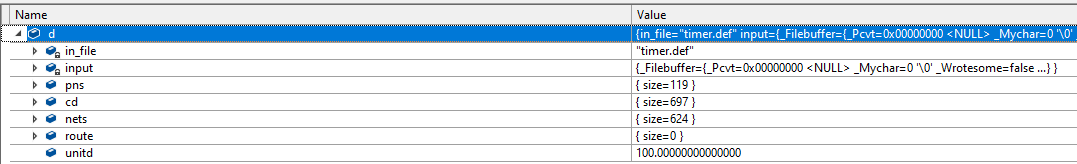
This where the def and circuit are called then the unified data structure (d) is used to output the spef file.

**Data Structures**

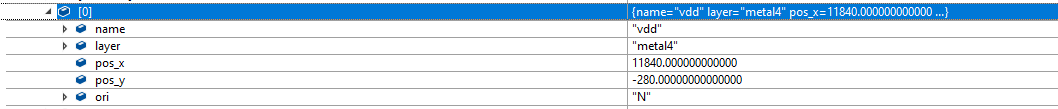
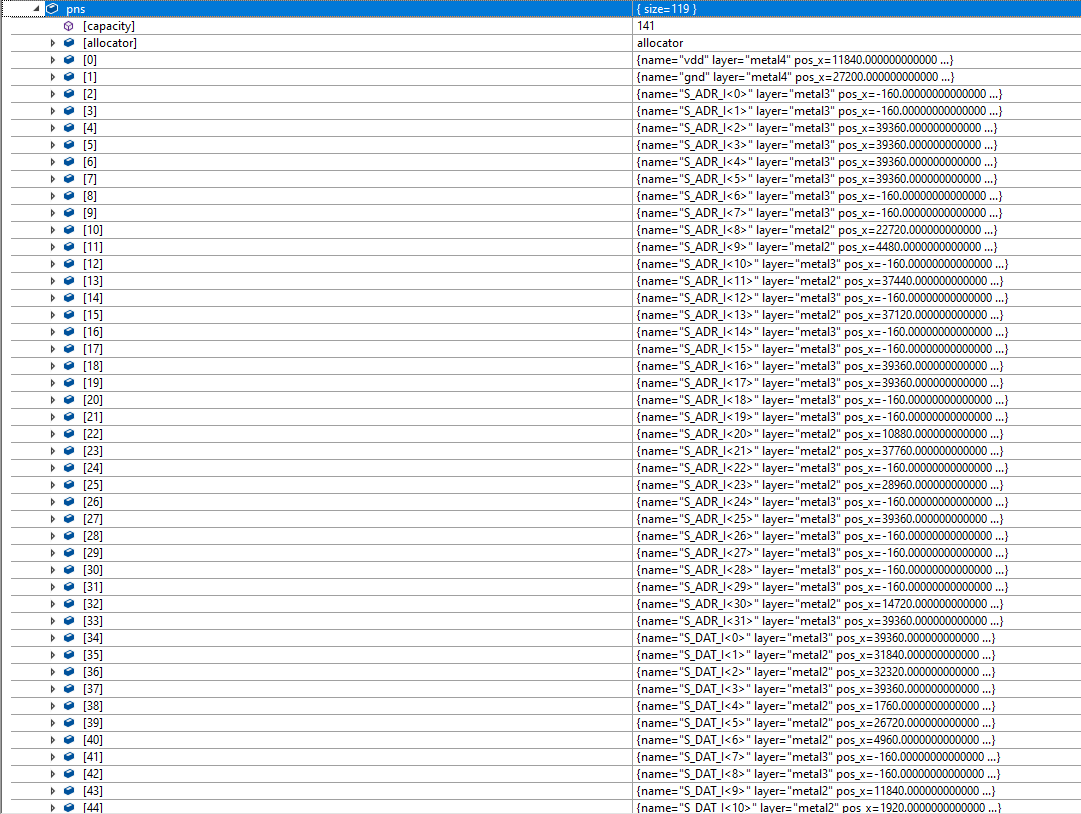
**To create the main data structure used to output the SPEF the two classes are called in the following way.**

****

***Now, we will explore the data structure “d”***

**`**

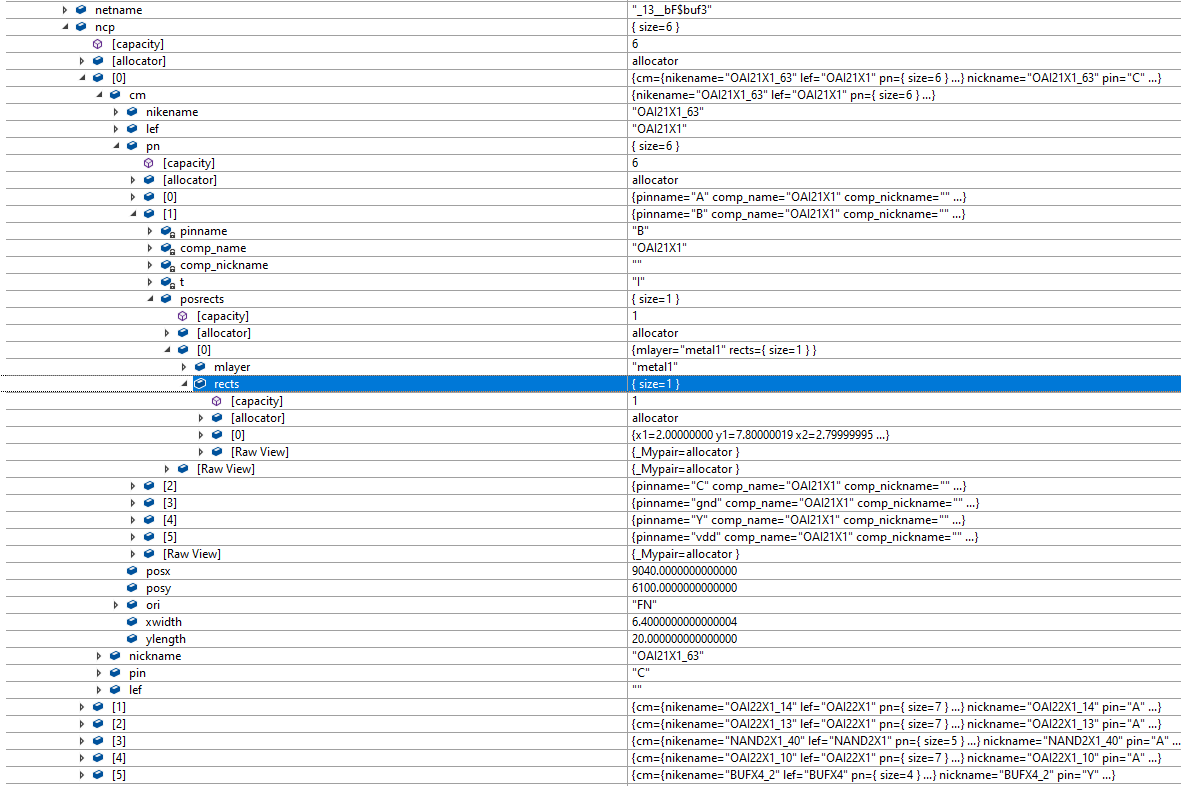
1. **Pns: These contain the main ports in the def file along with detailed info about them as shown below**

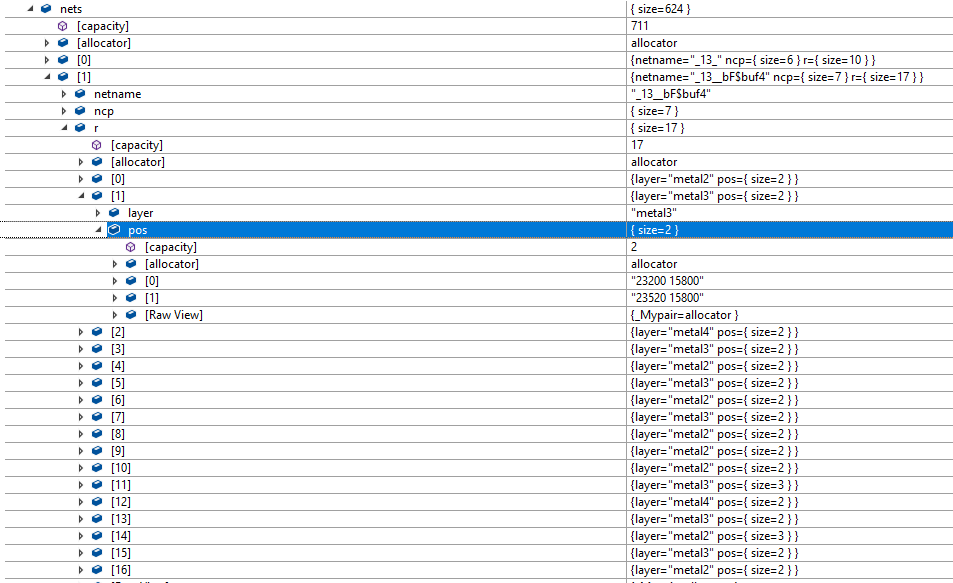


1. **Cd: contain all the components with their positions on the grid, orientation, and dimensions (from the lef)**



1. **Nets:**

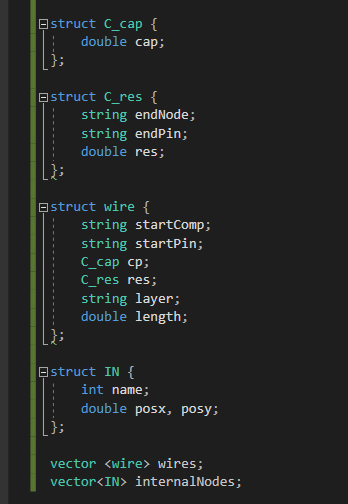




For each net:

1. Net name
2. Net components
   1. Associated pin in the net section
   2. Macro name
   3. All pins for this macro
      1. Pinname
      2. All the rects with their metal layer
   4. Position and orientation of the net component on the grid
   5. Dimensions of the net component (width and height)
3. Vector of routes (r)
   1. Metal layer of route(wire)
   2. Coordinates of points that the wire connects
4. **Unitd : This is unit used for conversion in the def file**

***Now, we will explore the data structure that contains the info that generates the SPEF file.***



**For ever net there is a vector of struct wires. Each wire contains:**

A-A start node (this could be one of the net components with their associated pin or an internal node that is created)

B- C\_cap this represents the capacitance of the wire and the start node is the same as the wire ( the endNode is always the ground)

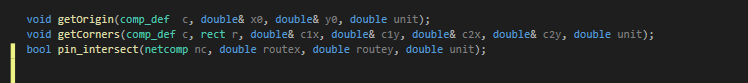
C- C\_res this contains the endNode as well because resistance connects between two nodes ad the value of the resistance.

D- Length of the wire

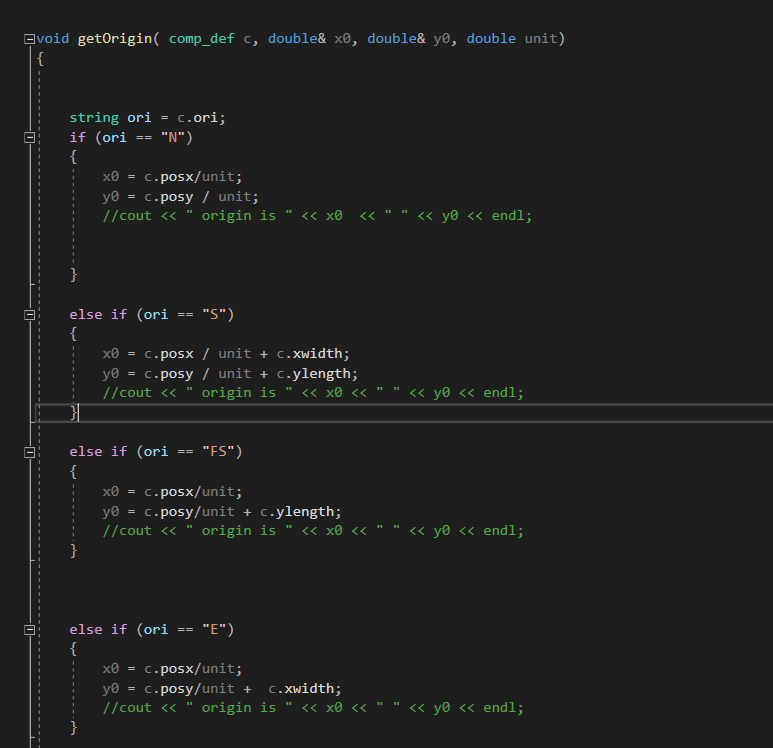
**For each net, there is a vector of wires according to the number of routes.**

**Main Functions**

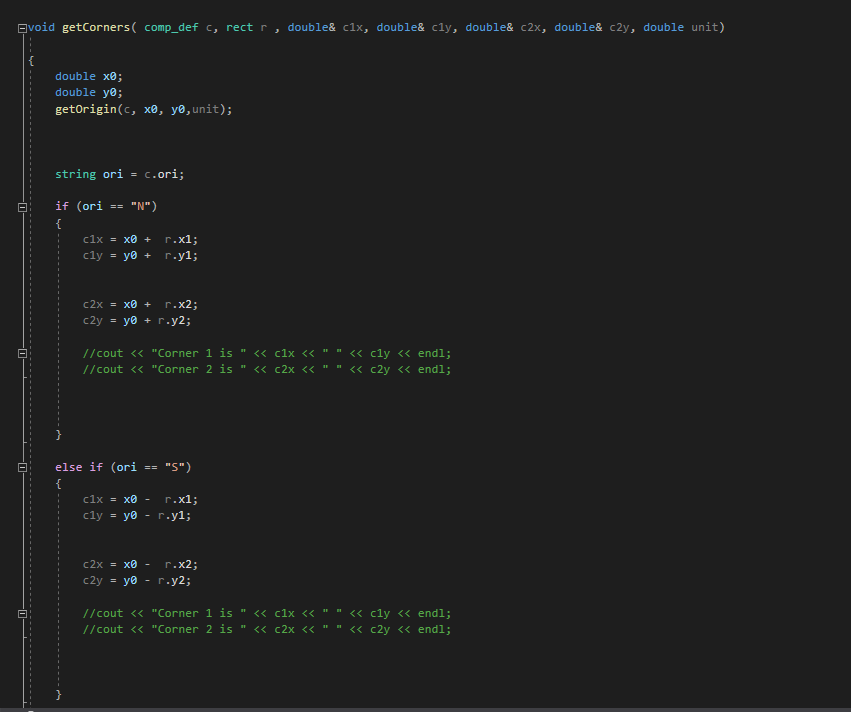
**1) These functions are used to find the intersection between the coordinates of the routes and the net components taking into consideration the different orientations.**



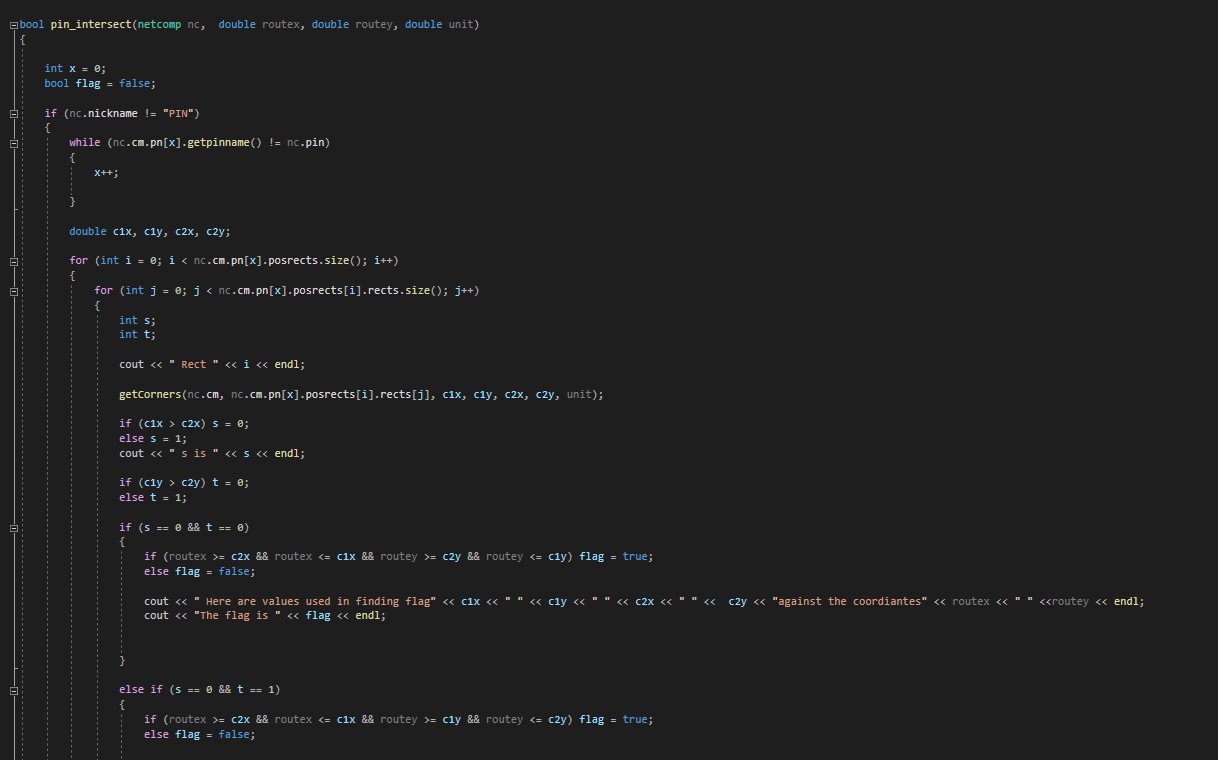
**A- getOrigin gets the new origin of the component on the grid based on the position of the component and orientation in the def file ( there are 8 cases)**



**B- getCorners gets the coordinates of the two corners of the rects associated with the pins based on the orientation and the new origin calculated above.**

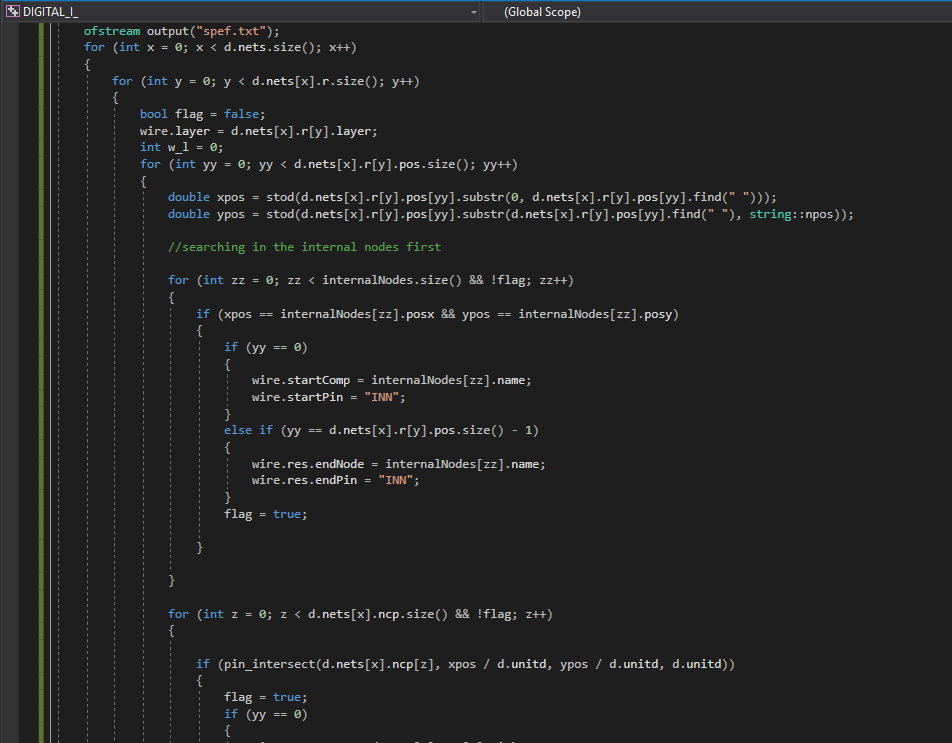


**C- pin\_interesect: a net component and a coordinate is passed to this function and the intersection is checked according to all the rects associated with the pin associated with the net component**



**D- The main itself**

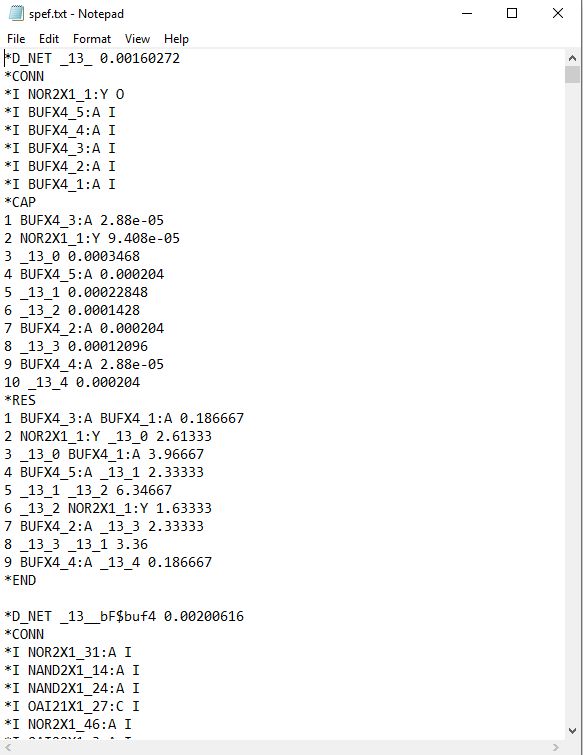
**Within the main, for every net we go through every route and check for intersections between coordinates and components/internal nodes and determine the start and end node of the wire. The length of the wire is calculated and the resistance and capacitance for each is calculated and stored in the wire. For every net, there is a vector of wires. Then the spef file is generated.**

****

**Test Cases:**

**Here are screenshots of the SPEF file generated using the def and lef files.**

**(timer.def and osu035.lef). Attached to the project file are the lef and def files and the SPEF generated.**



**Limitations**

1)Even though we outputted SPEF files, we did not test if they are correct as we were having issues with OpenSTA.

2) Since we created our own parsers, we are assuming that tall def files follow the same format as timer.def so any format that is different ( in terms of indentation and categorization) will not be supported.