VLSI Detailed Routing

Implementation using Net Merge Channel Router Algorithm in C++

Rafal Piersiak
Stony Brook University

***Abstract*—Detailed Routing takes the channels and switchboxes created by the global routing and wires the nets to the terminals. It is a methodic approach, efficiently wiring each region one-by-one, ensuring that all previous routed regions are unaffected by the current routing. Detailed routers have the ability to increase the channel height if necessary to accommodate more tracks when necessary, but requires adjusting the floorplan. The algorithm proposed in this paper will discuss the Net Merge Channel Router, which uses a zone representation and a vertical constraint graph to merge nets to minimize the overall channel height. This paper will discuss the proposed solution, implementation issues, and experimental results.**

Keywords— VLSI; Net; Merge; Channel; Detailed Routing; Yoshimura; Kuh;

I. Introduction

Routing is a two-phase approach, first starting with global routing and then follows into detailed routing. Global routing defines a set of routing regions called channels and switchboxes, which contain numbered pins on opposite sides for channels, and on two adjacent sides for switchboxes. These channels and switchboxes are then fed into the detailed router, which performs the actual wiring inside each element.

Detailed routing is performed incrementally, one element at a time based on some predefined order. The order is determined on the critically of routing certain nets and the total amount if nets passing through a region [1]. Routing regions including channels and switchboxes are 2-D, meaning two metal layers are used. There are occasions where a switchbox may be 3-D; this increases the complexity of the routing considerably.

Channels are extendable in the Y-direction, meaning if the floorplanning underestimates the width of the channel, it can be extended. A left to right sweep can be performed after to alter the position of all channels affected by the expansion of the modified channel.

A floorplan may not have to be ripped up during these sweeps depending on whether the floorplan is slicing or non-slicing. Slicing floorplans can be easily expanded or contracted because there are no cyclic dependencies on channels, meaning the channel to the right of the origin channel does not go back past the origin channel. Non-slicing floorplans have channels that depend on channels to the left of themselves, which can cause cycles. Such channel may be converted to L-channels and 2-D switchboxes found in [1].

The routing problem is NP-complete [2]; there is no fast solution to find a solution. One metal layer or multi-layer routing problems are all considered NP-complete. Multi-layer routing utilizes vias to bind two layers together at a contact point. Generally, one layer only contains horizontal segments, while the other layer contains vertical segments. The intended connection of a horizontal and vertical wire is electrically connected with a via.

During routing several parameters define the routing strategy: the *number of terminals*, *net width*, *pin locations*, *via restrictions*, *boundary type*, *number of layers*, and *net types*. Many strategies have been developed to perform detailed routing; all ensuring no two wires from different nets are allowed to cross each other on the same layer. Channel routers also try to minimize the channel width as much as possible as well as ensure timing constraints are met for each net. Other secondary objective include, improving manufacturability, minimizing vias, average net length, and minimizing number of vias per net [1].

II. Proposed Solution and Implementation Issues

A. Channel Routing Problem

A channel is a routing region bounded by two parallel rows of terminals [1], usually portrayed horizontally. The top row specifies the top terminals, the bottom row specifies the bottom terminals. Each terminals contains a net name, usually a number. Any terminals with a net name of zero have no electrical connection.

The channel length is equivalent to the number of columns of terminals on the top/bottoms rows. The channel height specifies the minimum number of tracks required. A horizontal segment inside the channel is a trunk; vertical segment is called a branch. The trunk is placed into a track.

A netlist is parsed and each top/bottom terminal is assigned a net name, respectively. These two terminals lists and the channel length specify the channel routing problem.

These specifications are used to find the interconnection of all the nets in a way that minimizes the channel height, and ideally the minimum number of tracks required. Additional objectives include minimizing the number of vias and the length of any net [1]. The channel router attempts to complete the detailed routing within the predefined channel height set by the floorplanner. If this is compromised, the channel must be expanded. To ensure continuity, routing of channels is performed in a predefined order that accommodates expansion of channels.

B. Horizontal Constraints

There exists a horizontal constraint between two nets when an overlap occurs during their simultaneous placement on an identical track [1]. A net N_i has an interval span I_i, defined by (r_i, l_i), where r_i is the right most net and l_i is the left most net [1]. The

Horizontal Constraint Graph (HCG) is an undirected graph $G_h = (V, E_h)$ where [1]:

$$V = \{v_i | v_i \text{ represents } I_i \text{ corresponding to } N_i\} \quad (1)$$

$$E_h = \{(v_i, v_j) | I_i \text{ and } I_j \text{ have non-empty intersection}\} \quad (2)$$

The HCG determines the lower bound of the channel height, depending on the maximum clique in the HCG. For a two layer net merge channel routing problem, summing the max number of nets in a zone determines the lower bound of the channel height.

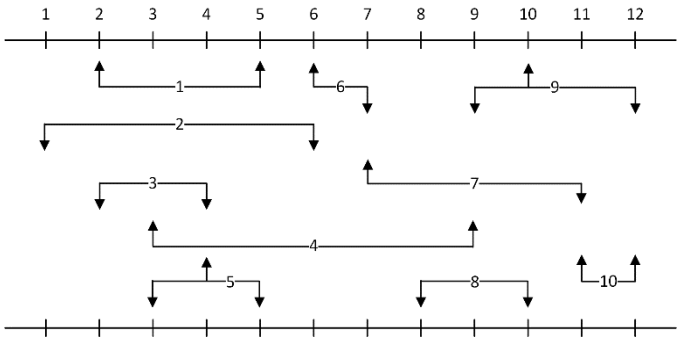

Figure 1: Netlist Channel Representation [2]

A channel netlist representation of a channel, found in Fig. 1, denotes the connections of the nets to the top and bottom terminals. An up arrow denotes a connection of the net to the top of the channel; the same holds true for a bottom arrow. Systematically going from left to right, placing each net with an up arrow in the first row, and placing each down arrow in the bottom row yields Fig. 2.

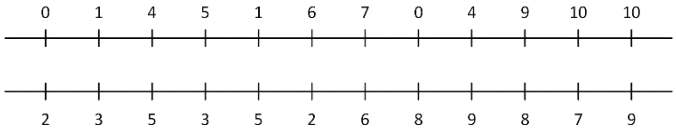

Figure 2: Typical Channel Routing Problem Representation

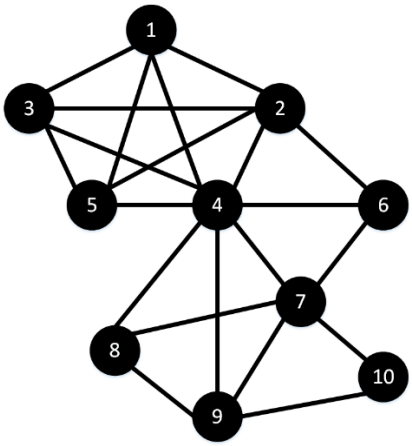

Figure 3: Horizontal Constraint (Interval) Graph

In this typical representation of a channel routing problem in Fig. 2, the undirected HCG is formed by connecting each extended net in the column to every other extended net in the column with an undirected edge. Multiple instances of the same edge are all considered one undirected edge. This process yields the HCG, also known as an interval graph in Fig 3.

The nets 1 – 5 form a complete graph – forming a maximum clique of five – specifying the lower bound of the minimum number of tracks.

C. Vertical Constraints

A net N_i has a vertical constraint with N_j if there exists a column such that the top terminal of the column belongs to N_i and the bottom terminal belongs to N_j and $i \neq j$ [1]. The Vertical Constraint Graph is a directed graph $G_v = (V, E_v)$ where [1]:

$$E_v = \{(v_i, v_j) | N_i \text{ has a vertical constraint with } N_j\} \quad (3)$$

In this typical representation of a channel routing problem in Fig. 2, the directed VCG is formed by checking if the top net is different from the bottom net for each column. If the nets are different, a directed edge is drawn from the top net to the bottom net. A terminal with a net of zero denotes that the net above or below the zero net is unconnected. Multiple instances of the same top/bottom net combination are all considered one directed edge. If a net A has a directed edge that bypasses a connected net B where both nets lead to the same final net C, the directed edge from net A to net C is broken, seen in Fig. 4. This ensures the longest paths are realized in the VCG. This entire process yields the VCG in Fig 5.

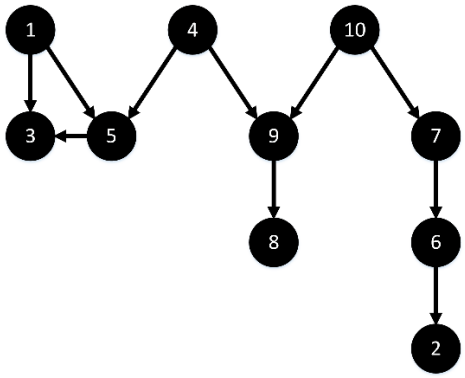

Figure 4: Bypassing Net

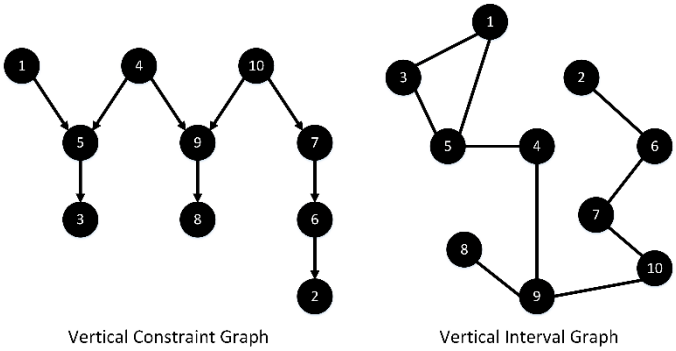

Figure 5: Vertical Constraint Graph

The VCG makes it aware that a vertical constraint implies a horizontal constraint, while the converse is not true. This is confirmed by the ability to stack the vertical graph on top of the horizontal graph, where the directed edges on the VCG are considered undirected. Therefore the VCG is a subset of the HCG. The HCG is not a subset of the VCG.

$$VCG \subseteq HCG \quad (4)$$

$$HCG \not\subset VCG \quad (5)$$

D. Net Merger Channel Routing Algorithm

1) Zone Representation of Horizontal Segments: Zones are formed by the maximal cliques in the HCG in Fig. 3. The maximal cliques are '1,2,3,4,5' '2,4,6', '4,6,7', '7,8,9', '7,9,10'. The channel height (track density) is the maximal clique with the most amount of nets.

To determine the zones for the Net Merge Channel Routing algorithm, we define S(*i*) to be a set of nets whose horizontal segments intersect column *i* [2]. The variable *zones* is assigned S(*i*), where S(*i*) is maximal.

This assignment of *zones* is carried out by the following pseudocode:

```
algorithm makeZones
begin
    load( S(i) );
    for I = (1 to channel length) do
        if ( S(i) - S(i - 1) != empty )
            if (S(i)  ⊄ S(i – 1) )
                new zone(z) = max( |S(i)|, z-1 to z )
end.
```

TABLE I. SET S(I) REPRESENTATION

Column	*S(i)*	*Zone*
1	2	1
2	1 2 3	1
3	1 2 3 4 5	1
4	1 2 3 4 5	1
5	1 2 4 5	1
6	2 4 6	2
7	4 6 7	3
8	4 7 8	4
9	4 7 8 9	4
10	7 8 9	4
11	7 9 10	5
12	9 10	5

The function load() cycles through each column looking for the same net N_j from the top and bottom rows in Fig. 2. It records the starting column and the absolute ending column. S(*i*) has N_j pushed onto each S(*i*), for the interval *i*, (starting to ending). Table 1 displays the elements in each subset of S(*i*).

Then each S(*i*) is scanned for a dropped element from S(*i*) - S(*i - 1*). If an element was dropped, check if S(*i*) is a subset of S(*i - 1*). If it is, make a new zone and include S(*i*), where S(*i*) is maximal from the new zone to all S(*i*) before the previous zone.

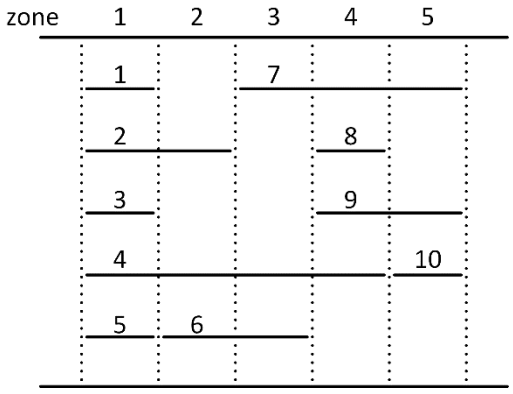

Figure 6: Zone Representation

The channel routing problem is completely characterized by the zone representation, Fig. 6, and the vertical constraint graph, Fig. 5 [1] [2].

2) Merging of Nets: The merging of nets is defined by to two conditions:

1. There is no edge between v_i and v_j in HCG.
2. There is no directed path between v_i and v_j in VCG.

If these conditions are satisfied, net N_i and net N_j can be merged to form a new fused set [1].

The merging of nets is defined by Yoshimura and Kuh in [2], referred as algorithm#1 found below:

```
    procedure Algorithm#1 (z_s, z_t)
    begin;
a1:     L = { };
a2:     for z = z_s to z_t do;
        begin:
a3:     L = L + {nets which terminate at zone z};
a4:     R = {nets which being at zone z+1};
a5:     merge L and R so as to minimize the increase of the
        longest path length in the vertical constraint graph;
a6:     L = L - {n_1, n_2, ...}, where n_j is a net merged at step a5;
        end;
    end;
```

Lines a3 and a4 performs the more detailed operations:

$$L = L + \{z_i - (z_i \cap z_{i+1})\}$$

$$R = \{z_{i+1} - (z_i \cap z_{i+1})\}$$

Line a5 creates the set L' by returning a set of merged nets using the Merge function:

$$L' = \text{Merge}(L, R)$$

Line a6 finds the difference between set L and set L' and sets it to the set L:

$$L = L - L'$$

Each merge operation has the two merged nets stores in another set for access later during placement of nets on tracks. For the channel routing problem specified in Fig. 2, five zones have been apportioned, represented in Table 1. A series of (|zones| -1) iterations takes place to merge the nets. A representative example of the merging process found in algorithm#1 will displayed in the following subsection.

a) Zone 1 to Zone 2: In the first iteration, L = {1, 3, 5} and R = {6}. Each L element is merged with each R element. The merging with the shortest VCG critical path is carried out to fruition.

The process begins by attempting to merge $N_{1.6}$, $N_{3.6}$, and $N_{5.6}$. Using the initial VCG in Fig. 5, merging N_1 and N_6 yields a critical path of 5, N_3 to N_6 equals 4, while N_5 to N_6 also produces 4. Since both $N_{3.6}$ and $N_{5.6}$ have equal length critical paths, either can be selected. For continuity, $N_{5.6}$ will be selected to correspond with [2].

The process for selecting the nets with the smallest critical path is more involved than has been specified here. Yoshimura and Kuh provided a simple and complete explanation of the heuristics performed during the Merge operation in Section IV. C [2]. They also mention a more advanced algorithm (algorithm#2) that implements merging based on matching. This algorithm was not implemented in this paper and will not be discussed in further detail, other than that subsequent mergings are not blocked by previous merges and it is more flexible.

The VCG's for each net merger is found in Fig. 7, labeled corresponding to the order of net mergers taken into account in this process description.

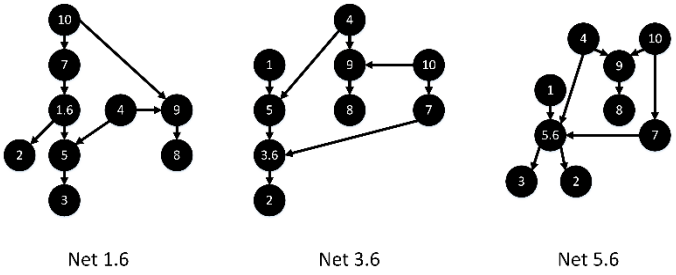

Figure 7: Summary of Merged Nets for each Iteration

Algorithm#1 cycles until the final iteration is completed. Table 2 summarizes the merging process for each iteration.

TABLE II. NET MERGING

Iteration	*Merged*
1	5.6
2	1.7
3	5.6.7, 3.8
4	4.8

Fig. 8 shows the alteration of the VCG for each iteration. The first iteration produces a merging of $N_{5.6}$, found in Fig. 8a. Then follows the merging of $N_{1.7}$ in Fig. 8b. The third iteration produces two merges, $N_{5.6.9}$ and $N_{3.8}$, both in Fig. 8c. The fourth and final iteration completes the merging process with the merger of $N_{4.10}$ in Fig. 8d.

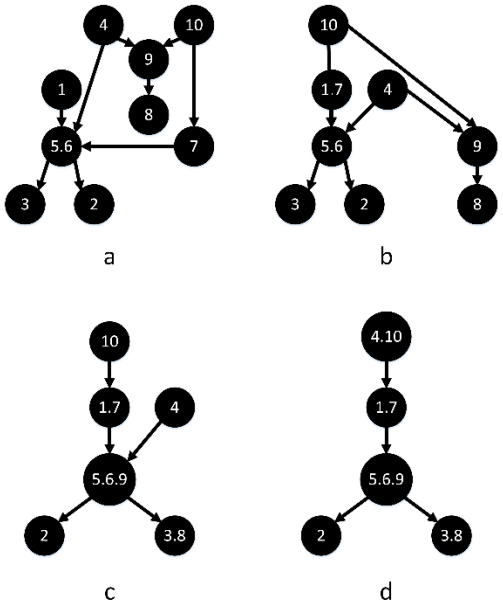

Figure 8: Summary of Merged Nets for each Iteration

3) Merged Zone Representation and Detailed Routing: The zone representation is updated during each merging of nets. Fig. 9 updates the zone representation in Fig. 6 to include all merged nodes.

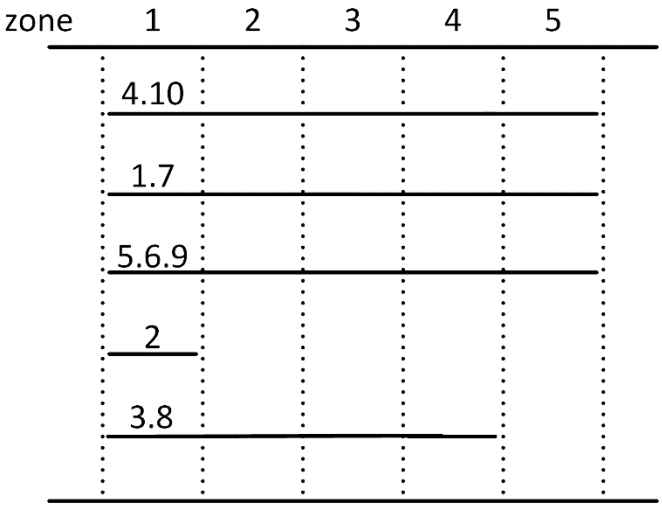

Figure 9: Updated Zone Representation

Based on the output of Fig. 8d, a detailed routing of the channel routing problem specified in Fig. 2 can be implemented. Track 1 is assigned to the top most net(s), $N_{4.10}$. Then follow the nets connected directly by the previous net(s), $N_{1.7}$. Track 3 contains

nets $N_{5.6.9}$, Track 4 is assigned N_2 and Track 5 is assigned $N_{3.8}$. The final detailed routing is presented in Fig 10.

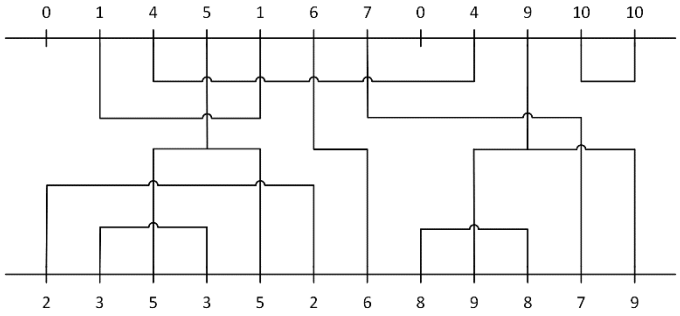

Figure 10: Detailed Routing

E. Program Overview

The implementation of the Net Merge Channel Router algorithm described in this paper is written in C++ using the C++11 coding standard. The IDE and compiler used for the construction and running of this program is Visual Studio Ultimate 2013. The program consists of a main file and four secondary files that contain the functions and variables called out in the main program. Each secondary file has its own header file, bringing the total file count for this program to nine files. This was done to aid readability and provide a formal structure to the program.

The initial iteration of the program can be described in the following manner:

1. Parse channel .txt file
   a) Place top nets into up vector
   b) Place bottom nets into down vector
2. Start clock
3. Run makeZones algorithm
   a) Load S(*i*) with nets for each column
   b) Define zone regions
   c) Move maximal S(*i*) into each zone
4. Run netMerger algorithm
   a) Create initial VCG
   b) Set up *L*, *R* sets
   c) Run Merge algorithm
      i. Create directed graph for each net pair
      ii. Perform heuristics table for each net pair
      iii. Find max(f(m)) for net *m* in z_i+1
      iv. Find min(g(n, m) for net *n* in z_i
      v. Return *n, m*
   d) Save merged nets
   e) Find *L* – *L'*
   f) Repeat 3b – 3e until $z = z_i - 1$
5. Stop clock
6. Output Results
   a) Print columns, nets
   b) Print channel routing problem
   c) Print S(*i*)
   d) Print zones with maximal S(*i*)
   e) Print merged nets in tracks

F. Input File Format

The channel input file was designed to be easily readable, but also simple enough to not require complex parsing algorithms to extract the net # and associated column positions. The first line *Columns:* specifies the channel length. The second line *Nets:* specifies the number of nets. Next is the information that specifies the channel routing problem in Fig. 2. The example channel input file found in Fig. 11 shows a portion of the entire file. The parsing algorithm first finds a net and saves it. It then proceeds to the line underneath the net and stores the net number into some column vector denoted *UP* or *DOWN*.

For Net 1, the number '2' specifies that the net #, '1' is placed into column vector *UP* (2), where 'U' denotes to place in the *UP* vector. The same applies for 'D'. Once the file is reached the channel routing problem in Fig. 2 is specified.

```
Columns: 12
Nets: 10
Net 1
2U 5U
Net 2
1D 6D
Net 3
2D 4D
Net 4
3U 9U
```

Figure 11: Example Channel Input File

G. Implementation Issues

The most difficult task was figuring out how to parse the channel routing problem into a directed VCG. After spending an entire night attempting various methods and techniques to find a bullet-proof solution to implementing a directed VCG, a method was developed utilizing an overlapping algorithm that checked for continuity between all top/bottom nets in each column. The algorithm is also able to find false paths, which skip a portion of the entire directed path. For instance, the algorithm was able to reason that 7, 6, 2 was a directed path, but it is later found that 10, 7, 6, 2 is a superset of 7, 6, 2. In order to avoid more heuristics to figure out whether a directed path is a superset of the other, once the bottom net of a top/bottom net pair matches with a net in the top column vector, the algorithm searches to find whether the top net of the top/bottom pair matches with a net in the bottom column vector. If both conditions are true, the top/bottom pair is a subset of a longer directed path, and is therefore discarded and the next subsequent top/bottom pair is analyzed. Fig. 12 provides an explanation of this special condition.

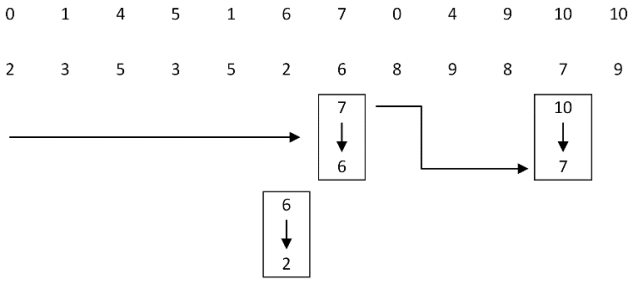

Figure 12: Special Subset Directed Path Condition

If the start of directed path is not a subset of a longer directed path, the algorithm proceeds to find all nets that connect to the current top/bottom net pair. Although the algorithm was relatively easy to implement, the methodology behind the algorithm was painstakingly difficult to understand and created a standstill in the implementation of the Net Merge Channel Routing algorithm.

The second most difficult task was to figure out how to define the zones. Yoshimura and Kuh provide some cryptic information on how S(*i*) relates to *zones*, but it was jargon to me. I spent some time playing with Table 1, trying to figure out a methodic way of figuring out why a zone stops and ends where it does in [2]. It was finally found that if a net drops from one S(*i*) to the S(*i+1*) and if S(*i+1*) is not a subset of S(*i*) then a new zone is made after S(*i*). If these conditions don't fall true in this order, a new zone is not made. This method was most likely what was presented by Yoshimura and Kuh in [2].

The parsing algorithm was straightforward after much practice with parsing files in the first and second projects. Utilizing C++ built-in set functions was extremely useful and made various functions of the algorithm straightforward to implement.

III. Experimental results

The output file provides a variety of useful information. The number of lines in the file, columns and nets read from the input file are pasted into the output file first. Then the channel routing problem is output exactly as seen in Fig. 2. Next, S(*i*) is printed for each column in Fig. 2. After that is the text description of the zone representation of the horizontal segments, with maximal S(*i*) for each zone. The track lower bound is calculated by finding the zone with the most elements. Next, the merged nets are shown in their respective tracks, where Track 1 denotes the top track.

The only experimental results available in this paper are circumstantial. The program was not implemented fully. The Merge function in the NetMerger file description is incomplete. In order to make the Merge function useable, the heuristics described in [2] would have to be implemented. Also, the directed graph function would have to be to take the nets inserted into the function and join them and create a new directed graph, which takes into account the merging of both nets and its effect on the other nets in the VCG.

With a few more hours, the heuristics would be completed quickly, primarily because it's just a bunch of mathematical operations that have a max and min value that reveal which nets to merge. The directed graph would need some tweaking to accommodate the introduction of two nets that must be merged in the VCG, which should not be difficult.

Also, some sorting would have to be done to make sure the merged nets are placed in the proper tracks. This can be easily realized with a comparison of the final VCG and extracting the net connections and placing them in separate tracks.

Overall, the program is ready to run and will even show incorrect merges repeatedly due to some hard coding of the nets that are returned by the Merge function. The output file is fully implemented.

IV. Conclusion

An attempt to reproduce the Net Merge Channel Router algorithm for this directed routing project was fully understood and mostly implemented, except for two functions, which were not fully implemented for the Merge function, due to time constraints.

Acknowledgment

This paper on the implementation of the Net Merge Channel Routing algorithm in C++ was submitted as a project paper for ESE 556 - VLSI Physical and Logic Design Automation, conducted by Professor Alex Doboli at Stony Brook University.

References

[1] Sherwani, N.A.; "Detailed Routing," in Algorithms for *VLSI Physical Design Automation*, 3rd ed. Norwell, K.A.P., 1999, ch.7

[2] Yoshimura, T.; Kuh, E.S., "Efficient Algorithms for Channel Routing," Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on , vol.1, no.1, pp.25,35, January 1982