Problem 2: Simultaneous Buffer Insertion and Steiner Tree Adjustment Considering Buffer Blockages

2005 ICCAD/SIGDA CADathlon

1. Introduction

Buffer insertion is a key technology for improving VLSI interconnect performance. Buffers can be inserted either in a long wire to regenerate the signal with improved strength or in a routing tree to shield capacitive load from timing critical paths.

Given a Steiner tree with required arrival time (RAT) at each sink node and a set of candidate buffer locations, a buffer insertion algorithm tries to find a buffer placement solution such that the minimum timing slack among all sink nodes is maximized. Most of existing buffer insertion algorithms follow the classic van Ginneken's algorithm. Van Ginneken's algorithm starts with generating candidate solutions at the sink nodes. Then, these solutions are propagated toward the source node along the given tree in a bottom−up manner. When a candidate buffer location is met, a new solution is generated by inserting a buffer there. Each solution at a node is characterized by the downstream capacitance and required arrival time at that node. A solution is inferior and pruned out if it has greater downstream capacitance and smaller required arrival time compared to any other solution. When these solutions reach the source node, the one with the maximum required arrival time (which is equal to the minimum slack among all sinks) is selected as the final solution.

In reality, some layout regions may be occupied by IP blocks or large memory arrays. These regions allow wires to pass through but have no room for buffer insertion. If a Steiner tree has large overlap with such buffer blockages, the effectiveness of van Ginneken's algorithm may be seriously degraded. This problem can be solved by performing tree adjustment simultaneously with buffer insertion. If a Steiner node overlaps with a blockage, one can generate new solutions by moving the Steiner node to a location outside the blockage and then inserting buffers. This technique is called RIATA (Repeater Insertion with Adaptive Tree Adjustment).

2. Problem statement

2.1. Brief description

Write a program to perform simultaneous buffer insertion and Steiner tree adjustment considering buffer blockages.

2.2. Input/output specification

The input is a file which has a section describing the Steiner tree and a section for the buffer blockages. The output file tells where the buffers are inserted and Steiner node location changes. The program should be invoked like this:

```
./riata tree_and_rect_filename
```

2.2.1. Input

The section for Steiner tree in the config file contains a node list, an edge list and a set of RC parameters. Each node is specified by its type, ID, x and y coordinates. For sink nodes, sink capacitance and required arrival time (RAT) are also provided. Only the source node and the Steiner nodes are candidate buffer locations. Each edge is specified by its upstream (closer to the source) node ID and its downstream (closer to sinks) node ID. The RC parameters include driver resistance, wire resistance per unit length, wire capacitance per unit length, buffer output resistance and buffer input capacitance.

The syntax of nodes and edges can be summarized as:

- nodeType nodeID x y sinkCapacitance sinkRAT
- edge upstreamNodeID downstreamNodeID

The section of blockages saves a set of rectangles. Each rectangle is specified by its minX, minY, maxX and maxY coordinates.

2.2.1.1. Example input

The following is a sample of Steiner tree section and parameters:

```
source  1   0   0
steiner 2   10  10
sink    3   20  10  10   2000
sink    4   10  20  1   100
edge 1 2
edge 2 3
edge 2 4
driverRes       1
wireRes         1
wireCap         1
bufRes          1
bufCap          1
```

The following is a sample of blockage list section:

```
rect 4  4   8  13
rect 11 4   20  13
rect 11 13  20  20
```

2.2.2. Output

The output is printed to stdout. It shows the timing slack of the buffer insertion solution. The slack is the maximized minimum slack among all sinks, or equivalently the maximized required arrival time at the source. A node list in the output tells where buffers are inserted. Since a node may have two child branches, it also tells which branch it drives. The modified Steiner node locations are also displayed.

2.2.2.1. Example output

The following is a sample output:

```
Slack  -874
BUF 2 drives child 6
```

```
BUF 3 drives child 4
BUF 3 drives child 5
BUF 2 drives child 3
STEINER_LOC_CHANGE 2 ( 58 50 )
```

3. Additional information

A framework of the program has been provided. It provides the definition of basic data structures, the parsing capability as well as the basic operations. You are expected to finish the implementation of the algorithm. In specific, you need to write a subroutine "findBufSolutions(const Node&, BufSolutionSet&)" which collects candidate solutions at a node.

There are two small differences from the original RIATA work published in ISPD 02. First, the alternative location of a Steiner node is restricted between its parent node and itself in the ISPD work. Here, the alternative Steiner node location is a point outside blockages and closest to its original location. Second, we restrict the candidate buffer locations to the source node and the given Steiner nodes although the locations of the Steiner nodes can be moved. In contrast, the ISPD work also considers the intersections between edges and blockage boundaries as candidate buffer locations.

This algorithm runs fast and normally the testcase is small. On an average platform, the runtime for each testcase should be less than one second. If the implementation takes more than 1 minutes to complete, the contestants will be graded fail even if the results are correct.

4. References

- Buffer Insertion With Adaptive Blockage Avoidance, J. Hu, C.J. Alpert, S.T. Quay and G. Gandham, IEEE Transactions On Computer−Aided Design Of Integrated Circuits And Systems, Vol. 22, No. 4, April 2003