Optimal Buffer Placement

Table of Contents

1. Introduction

Wire delays can no longer be neglected in modern day chip designs. Also, with increased miniaturization, wires become relatively large objects in comparison to transistors. Hence it is necessary to have a refined electrical model of a wire and also to acknowledge that large and long fanout trees benefit from additional buffers to amplify and reshape the signal from a source pin to its sink pins.

2. Problem statement

2.1. Brief description

Solve the optimal buffer placement problem in the context of Elmore delays for wire segments. Specifically, write a program that inputs a fanout tree and that calculates the latest required arrival time at the root (source) of the tree under the assumption of an optimal buffer placement. In case of multiple solutions with equal required arrival time, the solution with least capacitive load is to be preferred. You can take advantage of the partially given code.

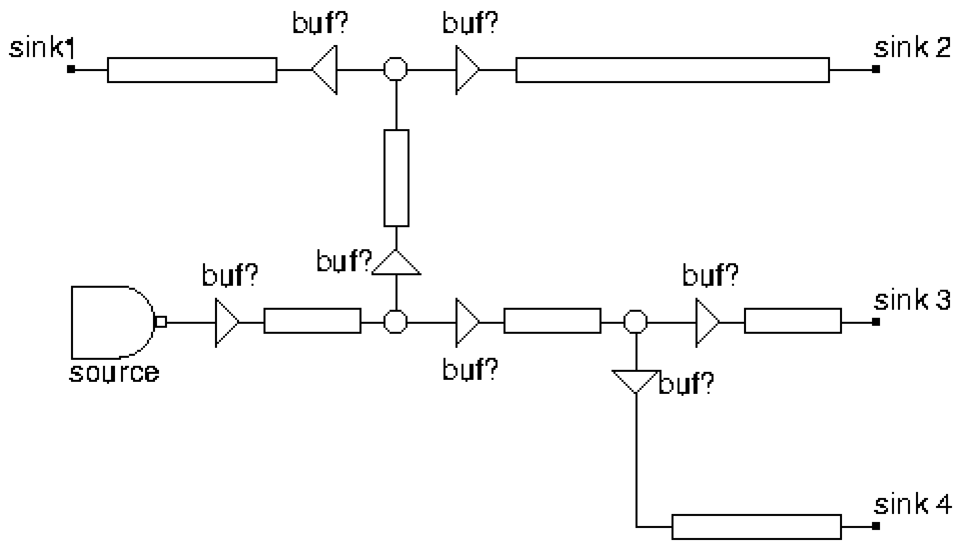
A sample circuit is shown in figure 1. We see a number of sinks driven by a NAND gate source. The small circles indicate internal nodes. Wire segments run from the source node to an internal node, between internal nodes, and from an internal node to a sink node. Note that the capacitive load attributed to a sink node is not shown. For uniqueness of the solution, we impose the restriction that a buffer (shown as a small triangle) can only be inserted at the start of each wire segment. In practice many other buffer locations are allowed.

2.2. More detailed description

Note that it is **not** required to actually find the optimum buffer distribution; it suffices to calculate the best required arrival time of signals at the source and the least capacitive load seen at the source. The necessary input data are the required arrival time and capacitive load for each sink node, the topology of the fanout tree network and the length of each wire segment. For simplicity, the per unit length resistance and capacitance values will be fixed, as will be the buffer characteristics.

You may assume that the problem can be modeled by a (pure) **binary tree**. The fanout tree may be recursively defined as follows:

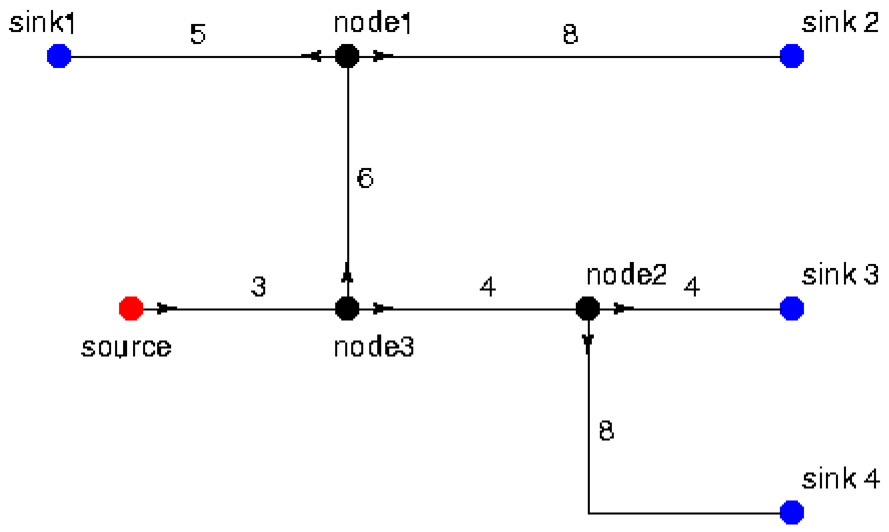
Figure 1. Illustration of buffer placement problem



- A tree either consists of a single root node and a single sink node, connected by a wire segment, or
- A tree is the combination of two subtrees connected at an internal node (identifying the two subtree roots and the new internal node) which is connected via a wire segment to the new root node.

Figure 2 shows an example of a well-defined fanout tree corresponding to the circuit of figure 1. Sink nodes are colored blue; internal nodes are two-way branching and are shown in black. The root or source node is colored red. Possible buffer locations correspond to the arrow points.

Figure 2. Abstract fanout tree



The edge labels are the respective wire lengths. It should be obvious that in an implementation, the wire length of a segment can be stored at the destination node. Hence, the source node need not explicitly be represented in the tree data structure (virtual root).

The necessary theory for the calculation and a possible algorithmic approach are detailed in the paper by Lukas van Ginneken. The pseudo code given in that paper is a bit obscure. The C code below emphasizes the operative steps of the required top-level routine.

```
Options bottom_up(Node k)
{
  Options Z;

  if (is_leaf(k))
    Z = options_sink(k);
  else {
    Options Z1, Z2;

    Z1 = bottom_up(subtree1(k));
    Z2 = bottom_up(subtree2(k));
    Z  = options_combine(Z1, Z2);
  }
  Z = options_add_wire(Z, wire_length(k));
  Z = options_add_buffer(Z);
  return Z;
}
```

2.3. Input/output specification

The input is specified by the Backus-Naur Form syntax description of the next section. The output solely consists of two pairs of numbers: the first pair presenting the data for the unbuffered case, the second pair presenting the data for the optimally buffered case. To minimize rounding errors, it is strongly advised to use the C type `double` for all floating point data types.

2.3.1. Input

The following table lists the given values for the wire and buffer characteristics. The wire values are specified per unit length. These values may be hard-coded into your program.

Table 1. Fixed values for wire and buffer

Element	Resistance	Capacitance	Delay time
wire	0.1	0.2	(according Elmore formula)
buffer	10.0	4.0	2.0

Here is the syntax of the input file.

```
        Input         : Tree .
        Tree          : Leaf
                      | "(" Id Wire_Length Tree Tree ")" .
        Leaf          : "<" Id Wire_Length Required_Time Load ">" .
        Wired_Length  : Float_Number .
        Required_Time : Float_Number .
        Load          : Float_Number .
        Id            : Identifier .
```

You may assume that the actual supplied input conforms to the format as specified above. There is no need to worry about ill-formatted input. Moreover, a parser for the input format written in C is available to you.

2.3.1.1. Example input

Here is a sample of an input data file with a topology that corresponds to the tree in figure 2:

```
# Example input file "test1"
( node3 3.0
  ( node1 6.0
    < sink1 5.0 34.5 64.0 >
    < sink2 8.0 24.0 30.5 >
  )
  ( node2 4.0
    < sink3 4.0 34.5  7.0 >
    < sink4 8.0 19.3 13.0 >
  )
)
```

Additional comments may be attached by using the "#" character.

2.3.2. Output

The program first prints the latest required arrival time at the (virtual) source node and the total capacitive load at the source for the case when no buffers are inserted. The pair of numbers must be enclosed in parentheses and separated by a comma and a space character. On the next line it prints the pair of numbers associated with the optimally buffered case. All numbers should have precisely 2 digits after the decimal point (Use the format `"%.2f")`.

2.3.2.1. Example output

```
         (-96.20, 122.10)
         (-95.06, 118.30)
```

3. Additional information

Take advantage of what is given to you. Really all you need to implement is how to combine solution sets for the two children of each node (the function options_combine() in the code).

4. References

- Buffer Placement in Distributed RC-tree Networks for Minimal Elmore Delay, Lukas P.P.P. van Ginneken, Proceedings IEEE International Symposium on Circuits and Systems, Vol. 2, May 1-3, 1990, pp. 865-868
- An O(nlogn) Time Algorithm for Optimal Buffer Insertion, Weiping Shi and Zhuo Li, Proc. Design Automation Conference, June 2-6, 2003, Anaheim, CA, pp. 580-585