Fanout-tree Restructuring Algorithm for Post-placement Timing Optimization

T.Aoki, M.Murakata, T.Mitsuhashi and N.Goto†
Semiconductor DA & Test Engineering Center,
†Research and Development Center,
TOSHIBA
Kawasaki, JAPAN

Abstract

This paper proposes a *fanout-tree restructuring* algorithm for post-placement timing optimization to meet timing constraints. The proposed algorithm restructures a fanout-tree by finding a tree in a graph which represents a multi-terminal net, and inserts buffer cells and resizes cells based on an accurate interconnection RC delay without degrading routability. The algorithm has been implemented and applied to a number of layout data generated by timing driven placement. Application results show a 17% reduction in circuit delay on the average.

1 Introduction

With today's sub-micron process, the interconnection delay becomes a dominant factor in total circuit delay. Therefore, timing consideration in logic and layout design stages is indispensable for designing high performance LSIs.

A number of works on repower of cell size and buffer insertion have been proposed [5, 6, 7, 8, 9]. However, some traditional methods may arise timing violations after layout because the methods use inaccurate delay based on the number of fanouts.

To solve this problem, several synthesis/resynthesis methods that consider layout information in the synthesis process have been presented. Pedram et al. [1, 2] describe a logic synthesis method in which rough placement is done during logic synthesis process, and use placement information to estimate delay and routability. Ginetti et al. [3] and Kannan et al. [4] describe a gate sizing and fanout-buffering method, which uses a placement information to calculate interconnection delay. However, timing measures used in these methods are based on the estimated wire load capacitance. So, the estimated interconnection delays during the synthesis process are very different from the actual interconnection delays after placement. For today's sub-micron process, RC delay can amount to as much as 15% of the total interconnection delay. To obtain good results, or to obtain timing error free design, it is important to optimize circuits on interconnection RC delay based on the placement.

In this paper, we present a fanout-tree restructuring algorithm for post-placement timing optimization. The proposed algorithm restructures a fanout-tree by finding a tree in a graph that represents a multi-terminal net. Furthermore, the method inserts buffer cells and resize cells based on an accurate interconnection RC delay without degrading routability. This method is incorporated into a concurrent logic and layout design system [10] to reduce circuit delay on the layout stage. The algorithm has been implemented and applied to a number of layout data generated by timing driven placement. Application results show a 17% reduction in circuit delay on the average.

The paper is organized as follows: Section 2 introduces a basic idea for our approach. Some definitions and problem formulation are shown in Section 3. The detail of the fanout-tree restructuring algorithm is described in Section 4. The outline of the post-placement timing optimization is described in Section 5. Experimental results are shown in Section 6 and concluding remarks are given in Section 7.

2 Basic Idea

In this section, we describe a basic idea of the proposed *fanout-tree restructuring* algorithm. Fig. 1 is an example of tree structure of a multi-terminal net, v_0 is a signal source, $v_1, \ldots, v_3$ are signal sink (v_1 isn't connected with the tree yet) and v_6 is a Steiner point. In this example, an interconnection delay from v_0 to v_2 can be expressed as follows [11]:

$$\begin{aligned} d_a(v_0, v_2) &= d_a(v_0, v_6) + \beta R(v_6, v_2) C(v_6, v_2) \\ &\quad + \alpha R(v_6, v_2) C_i(v_2), \qquad (1) \\ d_a(\cdot, v_0) &= \alpha R(v_0) C_l(v_0), \end{aligned}$$

where $\alpha = 1.1$, $\beta = 0.7$, $C_i(v_i)$ is the input capacitance of terminal v_i, $C(v_i, v_j)$ is the wire segment capacitance between terminal v_i and v_j, and $R(v_i, v_j)$ is a wire segment resistance, $R(v_i)$ is a on-resistance of source terminal v_i and $C_l(v_i)$ is the load capacitance from vertex v_i toward leaf of a tree. $C(v_i, v_j)$ and $R(v_i, v_j)$ are expressed as follows:

$$\begin{aligned} C(v_i, v_j) &= cL(v_i, v_j), & (2)\\ R(v_i, v_j) &= rL(v_i, v_j), & (3)\\ L(v_i, v_j) &= |x(v_i) - x(v_j)| + |y(v_i) - y(v_j)|, \end{aligned}$$

where c is the wire capacitance per unit length, r is the wire resistance, $L(v_i, v_j)$ is a rectilinear distance between v_i and v_j, and $x(v_i)$ and $y(v_i)$ are x coordinate and y coordinate, respectively.

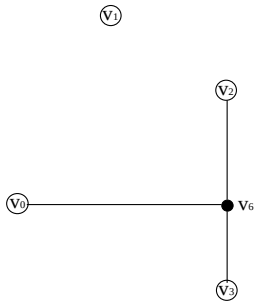

Figure 1: Example of routing tree. v_1 isn't connected with this tree yet.

Let's consider the case that the new signal sink terminal v_1 is connected to terminal v_2. The new delay from v_0 to v_1 can be calculated as follows:

$$\begin{aligned} d_a(v_0, v_1) &= d_a(v_0, v_2) + e(v_2, v_1), & (4)\\ e(v_2, v_1) &= C_\delta(\beta R(v_6, v_2) + \alpha R_m(v_2))\\ &\quad +\beta R(v_2, v_1)C(v_2, v_1) + \alpha R(v_2, v_1)C_i(v_1),\\ R_m(v_2) &= R(v_0) + R(v_0, v_6) + R(v_6, v_2),\\ C_\delta &= C(v_2, v_1) + C_i(v_1), \end{aligned}$$

where $d_a(v_0, v_2)$ is given by (1), $R_m(v_2)$ is a sum of wire segment resistance from terminal v_0 to v_2 and C_δ is the increase of load capacitance. In equation (4), $d_a(v_0, v_2)$ is the actual delay from v_0 to v_2 and $e(v_2, v_1)$ is a estimation delay from v_2 to v_1.

We use A*-Search algorithm to find buffer insertion points. The algorithm is known as the efficient algorithm to find the minimum/maximum cost paths from a given vertex s to other vertex t in a graph by the following estimation cost $\hat{f}(v)$.

$$\hat{f}(v) = \hat{a}(v) + \hat{e}(v), \quad (5)$$

where $\hat{a}(v)$ is a estimation cost of the optimal path from s to v. Also, $\hat{e}(v)$ is a estimation cost from v to t. In equation (4), the term $d_a(v_0, v_2)$ corresponds to $\hat{a}(v)$ and $e(v_2, v_1)$ corresponds to $\hat{e}(v)$. Then, the delay minimization problem can be solved by the A*-Search.

The *fanout-tree restructuring* algorithm is based on the above idea. It composed of the following basic two steps. In the first step, a *fanout graph* is generated from a multi-terminal net (as shown in Fig. 6). The detail of the *fanout graph* is described in Section 4.1. The *fanout graph* includes terminals, Steiner points of routing tree and buffer cells that will be inserted by *fanout-tree restructuring* process. In the second step, fanout-tree is generated by finding a path based on A*-Search. The generated fanout-tree suggests locations for the point of newly inserted buffer cells and its size. The detail of new estimation cost for A*-Search and path finding algorithm is described in Section 4.2.

3 Definitions

In this section, we introduce some definitions that are used for describing the algorithm.

Let c be a cell in a circuit C. Each cell has a set of signal sink terminals, or $\mathcal{I}(c)$, and a source terminal, or $v_0(c)$. The source terminal $v_0(c)$ has a connection to a set of sink terminals of other cells that is called *fanout set* of $v_0(c)$, or $\mathcal{F}(v_0(c))$. A terminal which drives $t \in \mathcal{I}(c)$ is denoted by $\mathcal{D}(t)$. Fig. 2 shows an example, $\mathcal{F}(v_0) = \{v_1, v_2, v_3\}$, $\mathcal{I}(c) = \{t_1^I, t_2^I\}$, $\mathcal{D}(t_1) = t_4^I$ and $\mathcal{D}(t_2) = t_5^I$.

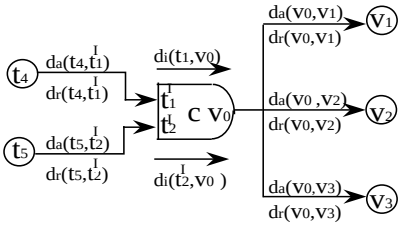

Figure 2: Calculating the actual and required delay.

A timing constraint $d_r(v_i)$ and an actual delay $d_a(v_i)$ for each terminal v_i belong to *fanout set* can be expressed as follows:

$$\begin{aligned} d_r(v_i) &= d_r(v_0(c), v_i) + MIN[d_i(t_j^I, v_0(c))\\ &\quad + d_r(\mathcal{D}(t_j^I), t_j^I) | t_j^I \in \mathcal{I}(c), v_i \in \mathcal{F}(v_0(c))], & (6)\\ d_a(v_i) &= d_a(v_0(c), v_i) + MAX[d_i(t_j^I, v_0(c))\\ &\quad + d_a(\mathcal{D}(t_j^I), t_j^I) | t_j^I \in \mathcal{I}(c), v_i \in \mathcal{F}(v_0(c))], & (7) \end{aligned}$$

where $d_i(t_j^I, v_0(c))$ is an intrinsic delay from t_j^I to $v_0(c)$,

$d_r(v_0(c), v_i)$ and $d_a(v_0(c), v_i)$ are timing requirement and actual interconnection delay from $v_0(c)$ to v_i, respectively. As equation (6) and (7), a slack for cell c, or $z(c)$ can be expressed as follows:

$$\begin{aligned} z(c) &= z_r(c) - z_a(c), \qquad (8) \\ z_r(c) &= MIN[d_r(v_i)|v_i \in \mathcal{F}(v_0(c))], \\ z_a(c) &= MAX[d_a(v_i)|v_i \in \mathcal{F}(v_0(c))]. \end{aligned}$$

The problem of fanout-tree restructuring is formulated as follows:

$$\begin{aligned} &minimize\ |z(c)|, c \in C \\ &subject\ to\ d_r(\mathcal{D}(t_i^I), t_i^I) \geq d_a(\mathcal{D}(t_i^I), t_i^I),\ t_i^I \in \mathcal{I}(()c). \end{aligned}$$

4 Fanout-Tree Restructuring Algorithm

In this section, we introduce the *fanout-tree restructuring* algorithm. This algorithm consists of two basic transformations, *repower* and *buffering* . The transformations are illustrated in Fig. 3 and Fig. 4. The *repower* chooses the best size of cell c and/or inserts buffer b (Fig. 3). The *buffering* inserts non-inverting buffers ($b1$) and remakes a topology of the multi-terminal net (Fig. 4).

The following pseudo-code is the *fanout-tree restructuring* algorithm. In the first step, the algorithm evaluates the slack $z(c)$ for the result of *repower* . If *repower* does not meet timing constraint ($z(c) < 0$), the algorithm applies *buffering* . The *buffering* process makes a *fanout graph* , or $G'(V', E')$ and finds the tree N_t in the G' by repeating a path search. The found tree expresses how to remake a topology of the multi-terminal net.

```
Fanout-Tree-Restructuring(c,ξ,τ,λ)
{
  z(c) = Repower(c,ξ,τ,λ);
  if(z(c) ≤ 0) return;
  G'(V', E') = Make-Fanout-Graph(c, ξ, λ);
  N_t = source vertex;
  Let SINK be a set of all the sink vertices of V'
       which is sorted by the slack.
  for each v* ∈ SINK {
       if(v* is included N_t) continue;
       PATH = A*-Search(N_t, v*, τ);
       N_t = N_t ∪ PATH;
       Update-Tree-Spec(N_t); }
  Insert-Buffer(N_t, ξ) ;
}
```

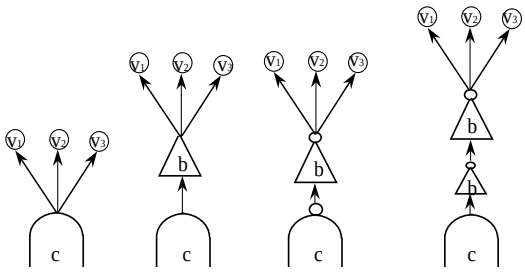

Figure 3: Repower transformations

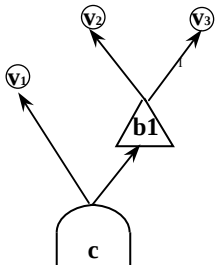

Figure 4: Buffering transformations

4.1 Fanout Graph

In this subsection, we define the *fanout graph* which is used in *buffering* to generate a fanout-tree.

A multi-terminal net can be represented as a directed planar graph (DPG), or $G(V, E)$. Where V is a set of vertices which correspond to a signal source terminal v_0 (*source*), sink terminals $\mathcal{F}(v_0)$ (*sink*) and Steiner points W (*spoint*), that is $V = \{v_0\} \cup \mathcal{F}(v_0) \cup W$. Also, E is a set of directed edge. If v_i and v_j can be connected by straight line and v_j isn't *source* , v_i has a outgoing edge from v_i to v_j, or $(v_i, v_j) \in E$. In DPG, there is a *source* at most one and it has only outgoing edges. Fig. 5 shows an example of DPG, where v_0 is a *source* , v_1,...,v_3 are *sink* and $v_4, \ldots, v_6$ are *spoint* . The symbol '$\rightarrow$' are uni-directed edges and '$\leftrightarrow$' are bi-directed edges.

Let $G'(V', E')$ be a *fanout graph* that is made by inserting new vertices and edges into a DPG $G(V, E)$, that is $V \subset V'$ and $E \subset E'$. If there is an edge $(v_i, v_j) \in E$, we insert a new vertex g which corresponds to a buffer cell and insert new two edges (v_i, g) and (g, v_j).

The following pseudo-code is the algorithm to make *fanout graph* .

```
Make-Fanout-Graph(c, ξ, λ)
{
  E' = ∅, V' = V;
  G(V, E) = Make-DPG(c, ξ, λ);
  for each (v_i, v_j) ∈ E {
       [x(g), y(g)] = SearchLoc(gξ);
```

```
        if(g can't place) continue;
        V' = V' ∪ {g}, E' = E' ∪ {(v_i, g), (g, v_j)};}
    return G'(V', E');
}
```

The *SearchLoc* step aims to search a rectangular area $([x(v_i), y(v_i)], [x(v_j), y(v_j)])$ for a placement location of g. Fig. 6 depicts the *fanout graph* G' that is re-made from DPG in Fig. 5, where g, g_1, g_2 are inserted *buffer* .

4.2 Path Search

In order to finds the path which meet the timing constraint, we use the A*-Search technique [13]. This approach is based on the performance driven global router described in [14]. In the first step of A*-Search, start vertices are pushed into a heap list H, that is maintained during path search. The A*-Search repeats the following process until the target vertex v^* is popped out from H.

1. Pop out a vertex u from H.
2. Calculate new cost for vertex v which adjacent to u.
3. If vertex v does not have trace-back-point, give cost and push v into H.
4. If v has trace-back-point, compare the new cost and old cost. If new cost are better than old cost, change new trace-back-point of v to u, give new cost to v and push v into H.

To find the path which meet the timing constraint, we define following cost function from equation (5).

$$
\begin{aligned}
f(v) &= d_r(v^*) - (d_a(v) + e(v)), \\
d_a(v) &= d_a(u) + \alpha R_m(u)(C(u,v) + C_i(v) \\
&\quad + \beta C(u,v) R(u,v) + \alpha R(u,v) C_i(v), \\
e(v) &= \alpha R_m(u) C_{buf} + \alpha R_{buf}[C(v,v^*) + C_i(v^*)] \\
&\quad + \beta C(v,v^*) C_i(v^*) + \alpha C(v,v^*) C_i(v^*), \\
R_m(v) &= \begin{cases} \textit{on-resistance} & \text{if } v \text{ is } \textit{buffer} \\ R_m(u) + R(u,v) & \text{otherwise} \end{cases} \\
C_m(v) &= MIN[C_m(u) - C(u,v) - C_i(v), \\
&\quad (d_r(v) - d_a(u,v))/R_m(u)], \\
(u,v) &\in E',
\end{aligned}
$$

where C_{buf} and R_{buf} are input capacitance and on-resistance of *buffer* , respectively. $C(u,v)$ and $R(u,v)$ are given by equation (2) and (3). $R_m(v)$ is a sum of interconnection resistance from *source* $v_0(c)$ to v. $d_r(v^*)$ is a required delay for a target vertex $v^* \in \mathcal{F}(v_0(c))$. $e(v)$ is a estimated delay from v to target v^*. $d_a(v)$ is an actual delay from *source* $v_0(c)$ to v when passing through u. $C_m(v)$ is an upper bound of add-able capacitance at v to meet timing constraints of already found paths.

In the step 4, if the value of $d_a(v)$ is bigger than required delay $d_r(v)$, the A*-Search does not push v into H because the condition $d_a(v) > d_r(v)$ means that the delay of this path does not meet the timing constraint for v. Also, the value of $C_m(v)$ is a negative, the A*-Search does not push v into Hbecause the negative value of $C_m(v)$ means that we can not add any capacitance to v.

In the step1, the target vertex v^* is popped out from H, the A*-Search traces back from the target v^* to start vertices to fix the path.

Let us consider a case of path search with an example of Fig. 6, Fig. 7 and Fig. 8. We assume that Fig. 6 is a fanout graph and Fig. 7 is a tree that was found so far. In this case, the set of start vertex is $N_t = \{v_0, v_1, v_3, v_4, v_5, v_6, g_1\}$, the target vertex is $v^* = v_2$. The final step traces back the path as $(v_2 \rightarrow g_2 \rightarrow v_6)$. As a result, the path $PATH = \{v_2, g_2, v_6\}$ is found. In this example, the net topology is depicted in Fig. 9.

4.3 Repower

In this subsection, we describe the algorithm of *repower* . The *repower* has done following transformations:

- Substitute other cell that has the same boolean function.
- Substitute other cell that has a negative boolean function and insert inverting buffer.
- Insert a non-inverting buffer.
- Insert two inverting buffers.

These transformations are depicted in Fig. 3. The *repower* algorithm is simple. The first step of *repower* makes a candidate list of transformation. The transformation candidate for a cell c is a combination of logically equivalent/negative cells and inverting/non-inverting buffers. In the second step, *repower* has done the following processes for each candidate of transformation:

- Place the substituted cell and inserted buffers.
- Construct a *trunk Steiner tree* [12] for calculating objective function $z(c)$.
- Choose the best transformation that has the minimum $z(c)$.

5 Post-placement Net-list Optimization

In this section, we describe the inputs, outputs and an outline of the post-placement net-list optimization (*PNO*). The inputs are a net-list of a circuit C, after placement layout data ξ, timing constraints τ and library λ. The input library includes multiple transistor size cells which have the same boolean function and the timing information of cell consists of an intrinsic delay, input capacitance and on-resistance. The input timing constraint consists required time for each interconnection. The following pseudo-code is an outline of *PNO*.

```
postPlacementNetlistOptimizer(C,ξ,τ,λ)
{
  CELL_LIST = ∅;
  for echo c ∈ C {
      z(c) = Calculate-Slack(c, τ, ξ);
      CELL_LIST ← [c, z(c)]; }
  for each c ∈ CELL_LIST
      Fanout-Tree-Restructuring(c,ξ,τ,λ);
}
```

In the first step of the *PNO*, required and actual delay for each cell c is calculated. The $CELL\_LIST$ is sorted by slack $z(c)$, so that the first cell of the list has the smallest slack. In the second step, the *fanout-tree restructuring* , which resizes a cell and inserts buffers, is applied to each cell in $CELL\_LIST$ to meet timing constraints. The outputs are net-list and layout data. The output net-list is optimized by the *fanout-tree restructuring* . Also, the output layout data has new location of resized cells and inserted buffers.

6 Experimental Result

The above method has been implemented. The program runs on SUN/SparcStation-10 and applied to SOG/EA type LSIs as shown in Table 1. Experiments have been performed as the following procedure: (1) Timing driven placement with an initial synthesized net-list. (2) Net-list optimization by our method(*PNO*) for the timing driven placement result. Set the timing constraints to critical paths based on the precisely estimated interconnection delays for given cell placement results [10].

The experimental results are shown in Table 2. The column labeled *Delay* represents the critical path delays. Notice that *pre* is a before net-list optimization and *post* is an after optimization. On the average, delay was reduced by 17%.

Table 1: Chips characteristics

	CHIP1	CHIP2	CHIP3	CHIP4
Technology	$0.5\mu m$	$0.8\mu m$	$0.8\mu m$	$0.8\mu m$
#Nets/#Cells	60k/50k	23k/20k	54k/43k	23k/18k
	CHIP5	CHIP6	CHIP7	CHIP7
Technology	$0.8\mu m$	$0.5\mu m$	$0.5\mu m$	$0.5\mu m$
#Nets/#Cells	14k/11k	12k/12k	19k/17k	30k/26k

Table 2: Experimental results

	Delay(ns)		delay	cpu
	pre	post	reduction(%)	time(min:sec)
CHIP1	23.00	18.98	17.5	
	17.49	11.14	36.3	
	15.74	11.14	29.1	
	7.95	7.39	7.7	
CHIP2	50.09	41.99	16.2	
	47.28	39.60	16.2	
	44.61	35.75	19.9	
	43.72	36.29	17.0	
CHIP3	52.44	44.77	14.8	
	27.78	24.33	12.4	
CHIP4	42.87	39.88	7.0	
	32.13	18.44	42.6	
	22.53	16.52	26.7	
	18.14	14.72	18.7	
	17.81	12.99	27.1	
	5.26	5.14	2.3	
CHIP5	10.88	10.81	0.6	
	41.88	39.00	6.9	
CHIP6	14.87	13.68	8.1	15:19
CHIP7	15.65	12.44	20.5	28:31
CHIP7	13.73	11.54	15.9	17:29

7 Conclusions

We have presented a *fanout-tree restructuring* algorithm for post-placement timing optimization. The algorithm performs the restructuring to meet the timing constraints by buffer insertion and gate sizing. By the A*-Search on fanout graph, the optimal buffer locations are found without degrading routability that may occur by wire crossing caused by the restructuring. Experimental results show 17% reduction in circuit timing for circuits laid out by timing driven placement program.

References

[1] M.Pedram and N.Bhat, *28th DAC*, pp.99-105,1991.

[2] M.Pedram and N.Bhat, *ICCAD-91*, pp.134-137,1991.

[3] A.Ginetti and D.Brasen, *CICC-93*, pp.9.2.1-9.2.4,1993.

[4] L.N.Kannan,P.R.Suaris and H.Fang, *31st DAC*, pp.327-332,1994.

[5] M.A.Cirit. *24th DAC*, pp.121-123,1987.

[6] P.K.Chan. *27th DAC*, pp.353-356,1990.

[7] K.Yoshikawa et al., *28th DAC*, pp.112-117,1991.

[8] K.J.Singh and A.Sangiovanni-Vincentelli, *27th DAC*, pp.357-360,1990.

[9] S.Lin and M.Marek-Sadowska, *Proc. of the European Conference on Design Automation*, pp.539-544,1991.

[10] M.Murakata et al., *CICC-95*, pp.465-468,1995.

[11] H.B.Bakoglu, Circuits, Interconnections, and Packaging for VLSI, *Addison-Wesley*,1990.

[12] M.Igarashi et al., *SASIMI-93*,pp.253-244,1993.

[13] N.J.Nilsoon, Problem-Solving Methods in Artificial Intelligence, *McGraw-Hill*,1971.

[14] S.Prasitjutrakul and W.J.Kubitz, *IEEE Trans. on CAD*, vol.CAD-11,no.8,pp.1044-1051,1992.

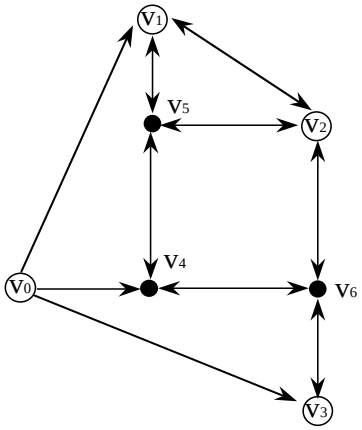

Figure 5: Example of directed planner graph. '$\rightarrow$' are uni-directed edges and '$\leftrightarrow$' are bi-directed edges.

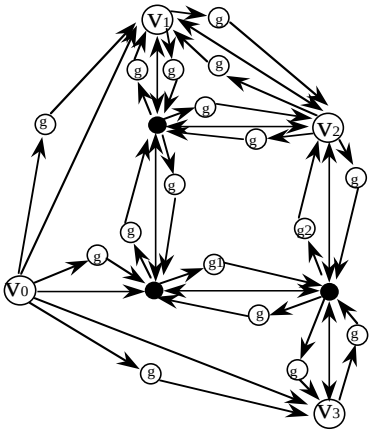

Figure 6: Example of fanout-graph. g, g_1, g_2 are inserted *buffer* .

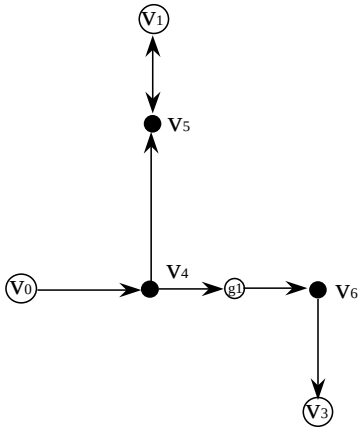

Figure 7: Example of start vertices

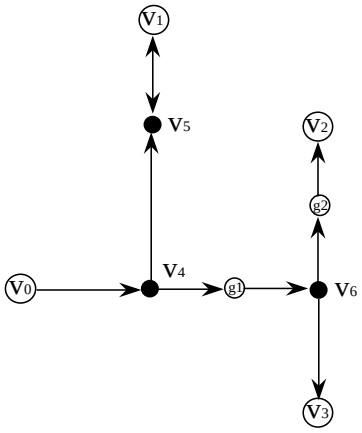

Figure 8: Example of maximized slack tree

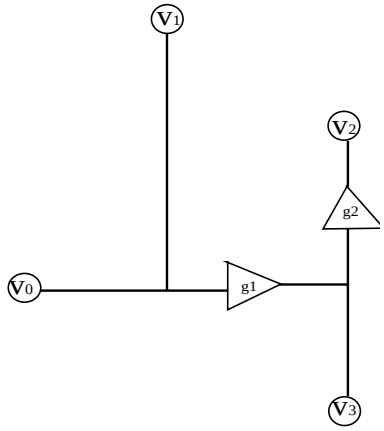

Figure 9: Example of fanout-tree configuration. g_1 and g_2 are inserted buffer cells.