Clock Tree Optimization for Power Supply Noise Reduction

Yaakov Kaplan and Shmuel Wimer

Jan. 12, 2015

CE Tech Report # 001

Based on Bar Ilan MSc. Thesis of Y. Kaplan, supervised by S. Wimer

Faculty of Engineering
BAR ILAN UNIVERSITY, Israel
Clock-Tree Optimization for Power Supply Noise Reduction
Yaakov Kaplan\textsuperscript{1,2} and Shmuel Wimer\textsuperscript{1}

Abstract
The voltage drop incurred by the power supply in today’s VLSI chips is a major concern, known as power supply noise. In sub-1 volt supply voltage, noise of very few hundred millivolts causes circuit malfunction. The reason for power supply noise is the fast and simultaneous voltage switching. While the logic signal switching is spread across the entire clock cycle, the switching of the clock-tree and the sequential circuits is occurring simultaneously, causing high current peaks. The latter is a primary contributor to the power supply noise.

This work proposes to spread the switching of clock-tree drivers in an attempt to reduce the peak current, while maintaining the clock signal quality and low skew at the far end tree’s leaves, where the sequential circuits are connected. A methodology of driver switching characterization has been developed for fast computation of peak current and other signal parameters, integrated into a two-phase optimization algorithm. It first initializes the clock-tree in a top-down traversal, employing a mix of high-threshold voltage (HVT) and low-threshold voltage (LVT) clock-drivers tree branches. A bottom-up delay correction phase then takes place, aiming at clock skew nullifying. The algorithm was implemented in 40 nanometers process technology, achieving a reduction of 50\% of the clock-tree peak current. The proposed method can be easily combined with other existing methods to further reduce the peak current.

1. Introduction
The voltage drop incurred by the power supply in today’s VLSI chips is a major concern known as power noise \cite{1}. With the increase of design complexity, moving from Application Specific Integrated Circuits (ASIC) to System on a Chip (SoC), and due to the sub-1 volt supply voltage, noise of very few hundreds millivolts causes circuit malfunction \cite{2}.

The fluctuations occurring in the high power supply $V_{\text{dd}}$ and the low power supply $V_{\text{gnd}}$ voltages are escalated by the process technology scaling. There, the underlying resistance, capacitance and peak current are increasing, and the current switching becomes faster, namely $dI/dt$ grows up, thus increasing the power noise. A typical model of power delivery network is illustrated in Fig. 1 \cite{1}.

The network consists of a power supply located on the board, power consumers (system’s logic circuits), and power and ground supply network (PDN). The power supply is assumed to behave as an ideal voltage source providing nominal $V_{\text{dd}}$ and $V_{\text{gnd}}$. The power consumers are modeled

\textsuperscript{1} Bar-Ilan University, Engineering Faculty, Ramat-Gan, Israel
\textsuperscript{2} Apache Corp.
as time-dependent current sources, denoted by \( I(t) \). \( V_{dd} \) and \( V_{gnd} \) lines connecting the supply with the power consumers are not ideal, having non-zero parasitic resistance \( R_p \) and \( R_g \), and inductance \( L_p \) and \( L_g \), respectively. The resistive voltage drops is \( I \cdot (R_p + R_g) \), while the inductive voltage drops is \( (L_p + L_g) \cdot dI/dt \). Shown in Fig. 1, the high and low voltages are therefore

\[
V_{dd} - I \cdot R_p - L_p \cdot dI/dt, \tag{1}
\]

\[
V_{gnd} + I \cdot R_g + L_g \cdot dI/dt. \tag{2}
\]

![Non ideal power delivery network](image)

Figure 1: Non ideal power delivery network [1].

Ordinary logic and sequential circuits are designed to work in nominal power supply voltage. Unfortunately maintaining constant voltage during operation is practically impossible. It all comes to the simple Ohm low of multiplying peak current by the power network impedance. The power network is an \( RLC \) circuit and high current peaks will thus cause various voltage drops at various points of the network. The noise can therefore be reduced by lowering the impedance, or avoiding high current peaks. Constructing proper power supply network having low-resistance and inductance, and high capacitance has been treated by many research papers [3-5], and its discussion is beyond the scope of this work, focusing on reducing the peak current.

Power noise can be controlled by reducing the PDN parasitic resistance \( R \) and inductance \( L \), and by lowering the current \( I \) and its density \( dI/dt \). Both techniques were extensively studied in the literature. In [4] PDN parasitic resistance and inductance reduction were studied. In [6] it was proposed to employ decoupling capacitors, hence reducing the effective \( R \) and \( L \) (and also the resonance factors). The reduction of \( dI/dt \) is obtained by improving the current sources. That can also be achieved by decoupling capacitors (though the reduction mechanism is different).

Peak current reduction achieves the following goals.
1. Reduction of the on-die \( IR \) drop, where the resistance dominates the impedance.
2. Reduction of the \( L(dI/dt) \) term occurring at the package level, where the inductance factor dominates the impedance.
3. Reduction of the clock jitter which is directly affected by $IR$ drop [7].
4. Improving the utilization of the de-coupling capacitors by increasing the effective distance of the capacitors, that is inversely proportional to $dI/dt$ [6].

The clock related voltage switching is the primary contributor to power supply noise [8]. While the logic signal switching is spread across the entire clock cycle, the switching of the clock-tree and the sequential circuits is occurring simultaneously, causing high current peaks. The clock network is therefore a natural candidate to treat for reducing peak current. A well-structured clock-tree should deliver high quality clock signal to the underlying sequential circuits connected at the tree’s leaves. To obtain proper and robust sequencing of the logic, the clock skew must stay within prescribed limits, usually not exceeding 5% of the clock cycle [9]. To ensure fast switching of the logic, the slope of the signal at tree’s leaves must also be sufficiently small.

Reducing the power supply peak current by clock-tree treatment is therefore a delicate task that must be handled carefully to ensure clock signal integrity. For that, three approaches have been proposed in the literature.

The first aims at reducing the power network impedance. Though it does not change the peak current, the voltage drop does reduce. This approach comprises well established techniques such as reducing the resistance of the power network by widening the power rails, increasing their density, and extensive usage of vias. Another technique is using decoupling capacitors, to effectively shorten the distance of the current sinks from their sources. Such techniques are being used since the early days of VLSI design. An excellent review of those methods is found in [6].

A second technique [10] proposed to reduce the peak current by minimizing its component caused by the flip-flops (FFs) switching. Its underlying idea is illustrated in Fig. 2. In (a) the clock signals of the FFs are aligned, causing a large and narrow supply current pulse, compared to (b) where the clock signals are displaced, and the resulting current pulse is smaller and spread over time. To account for the peak current, the signal switching [10] accounted only the first level of the combinational logic, assuming that the switching factor at deeper levels is far smaller than at the first level. Peak current reduction was achieved by clock scheduling procedure which utilized the allowable clock skew, a technique first proposed for timing optimization in a seminal paper [11]. Clock signal rescheduling at FFs’ clock inputs may considerably complicate the timing convergence of the underlying design. Peak current minimization by [10] could not be handled by the linear programming solution used in [11], and a heuristic based on genetic algorithm was proposed, yielding 30% of FFs peak current reduction, without sacrificing the clock frequency.

A refinement of [10] was presented in [8], which accounted all the switching of deeper logic levels. A more modest, but still significant peak current reduction of 12% was claimed, which lead to 19% reduction of the power supply voltage variation. A heuristic to enhance the quality of the genetic algorithm solution was presented in [12]. Paper [13] presents another skew spreading optimization technique by dividing the skew time intervals into slots and then the
Clock timing of each FF was allocated to some slot in an attempt to reduce the peak current. 17% peak current reduction was claimed.

Figure 2: Peak supply current reduction by misalignment of the clock signal.

All the above works used current profiling of the gates involved in the combinational logic. While [10] and [12] required full current characterization of the logic gates involved and thus employed extensive SPICE simulations, [8] used only the peak current of those gates, making the CAD solution simpler and easier to maintain on the account of accuracy. All those methods approximated the current waveforms by triangles, which were scheduled according to the switching time of their corresponding gates (see Fig. 2). The entire current profile drawn from the power supply was obtained by a superposition of the individual current profiles. Our work uses similar current profile methodology with full SPICE characterization. For our work SPICE accuracy is essential since unlike the above mentioned methods, we pursue zero-skew.

As in [8,10,12,13], our work is flattening the peak supply current by manipulating the timing of the clock signal. There is a major difference though, which avoids the timing side effects difficulties mentioned before. While the former methods shift the clock signal at the far-end of its distribution network, as shown in Fig. 2, our method does so only at the internal nodes of the clock-tree, while the skew at the its far-end nodes is fully controlled and maintained small. Furthermore, our method is systematic and does not require extra hardware, but rather mixes clock-drivers by of low-threshold (LVT) and high-threshold (HVT) types. The other skew-driven methods add delay elements at the far-end, a reason for further design complication and extra hardware overheads.
It should be noted that the main difference among former skew spreading methods is in their spreading algorithms and current profiling modeling, namely, in their design flow aspects or CAD approaches. Other than that, they all use the same paradigm of skew spreading. Though skew spreading techniques deliver the premise of peak supply current reduction, they impose significant burden on the design. That follows from the consumption of the skew margin for purposes other than solving delay violations, which may make the timing convergence very difficult. Techniques as time-borrowing [14] may be not applicable, as their main resource, namely, allowable clock-skew, is being consumed for another purpose. All in all, peak supply current reduction and time-borrowing conflict with each other.

![Diagram of buffer polarity assignment](image)

Figure 3: The idea of buffer polarity assignment. In (a), the buffer exhibits high $I_{dd}$ ($I_{ss}$) current at the rising (falling) edge of the clock signal. In (b), an opposite situation occurs for an inverter [16].

A third type of methods is reducing the peak supply current by mixing clock driver polarities within the clock-tree distribution networks, claiming for 50% peak supply current reduction [15]. The idea is illustrate in Fig. 3, where the power supply current is shown for a buffer in (a) and an inverter in (b). In Fig. 4(a) an ordinary network is using positive polarity drivers (buffers), presented with underlying FFs connected at the leaves. In Fig. 4(b), the polarities of the drivers are systematically mixed, causing a substantial reduction of the peak current. Still, local current peaks exist due to the uniformity of the clock-drivers in local regions (either inverters or buffers). To overcome that problem the work in [17] proposed to use the physical placement information of the clock-drivers so that for local regions about half of the driving elements are inverters and the other half are buffers.
A problem arising by the clock-driver polarity method is the uncontrolled skew occurring by different delays of inverters and buffers residing on root-to-leaf clock paths, as shown in Fig. 4(b). To control the skew occurring by driver type varieties of root-to-leaf paths, the work in [18] sized the clock-drivers for compensation. Polarity assignment aware of clock skew was also proposed by [19-21] which still used single size clock-drivers. A comprehensive solution for reducing power supply noise, combining the above approaches, has been described in [22]. It presented a practically efficient optimal algorithm based on dynamic programming integrated in a systematic design flow. It was lately enhanced in [16] to support multiple power modes, by dynamically controlling the internal delay of the clock-drivers.

Though the clock-driver polarity mix and its combination with driver sizing, is elegant and considerably reduces peak supply current, it may impose huge difficulties on the design. The authors of [15-22] claim that timing is not heart, which is true only for logic paths connected between FFs of same polarity. Logic paths connected between different polarities unfortunately cannot be avoided. Such paths leave only half clock cycle for the logic to compute, imposing huge difficulties on the design methodology and timing analysis tools.

The rest of this work is organized as follows. Section 2 describes the structure of a clock-tree and how power noise builds-up. It also highlights the theme of the novel clock-tree design method proposed by this work. Section 3 presents a clock-driver characterization method that is a central component to achieve computational efficiency of the peak current reduction algorithm, overviewed in Section 4. Section 5 develops a simplified algebraic model of the problem. Section 6 gets into more details of the clock-tree structuring and elaborate on the computation involved in the current pulse waveform derivation. Section 7 describes the algorithm in details. Section 8 presents experiments where currents and skews are calculated by the models and the characterization, compared to results obtained by full SPICE simulations, showing very good correlation. Section 9 concludes the work and proposes direction for further research.
2. Clock-tree and power noise

The construction of clock networks as a part of a SoC design offers various topologies, such as spine, mesh, grid and trees. The choice of the network topologies depends on many parameters and an overview of those can be found in [9]. No matter what topology is used for the clock distribution at the chip-level and block-level, the lower levels of the clock network mostly use tree topology, comprising three to five levels of hierarchy. This work focuses on those trees, aiming at reducing the peak current drawn from the supply.

![H-Tree clock network](image)

Figure 5: H-Tree clock network, schematics physical structure (left) and actual layout used by IBM\Motorola PowerPC processor (right) [24].

A widely used clock-tree example called H-tree is shown in Fig. 5. It was used by the PowerPC processors family [23,24]. H-tree symmetry made it favorite for distributing robust clock signals. By its very structure, each sink (tree’s leaf) has similar path to root, comprising similar drivers and wire segments. Up to on-die variations, H-tree ensures same nominal source-to-sink latency, and hence very small nominal clock skew. The terms sinks and leaves are used interchangeably.

The elegance of the H-tree structure is also a source of considerable power-noise. Due to its symmetry, all the drivers at a given level of the tree will nominally switch simultaneously (see Figs. 2(a) and 4(a)). This results in a progressive sequence of current peaks, cumulating to a current pulse whose amplitude is increasing along the progression from source (tree’s root) down to sinks (tree’s leaves), as shown in Fig. 6 by the red waveform. Flattening it will reduce the chip-level and the package-level noise, by reducing both $IR$ drop $L(dI/dt)$, as described in the introduction. The green waveform in Fig. 6 illustrates the current resulted by the clock-tree modifications proposed by this work. Notice the 40% peak current reduction.

To ensure robust signal with small slope, the clock-tree is traditionally using low (LVT) or nominal (NVT) threshold voltage transistors. Though suffering of high leakage, their short transition time ensures small slope of the clock signal at the sinks, where flip-flops (FFs) are connected. The uniformity of the clock-tree structure, where each level comprises identical
drivers, ensures the uniform propagation delay of the clock signal to sink, yielding small skew at the tree’s sinks. The robustness of the clock signal at tree’s internal nodes does not stand for itself; it helps to accomplish the integrity requirements at the sinks. An important question therefore is whether the driver uniformity and symmetry of the clock-tree is necessary to ensure the integrity at leaves? Or maybe the integrity at tree’s leaves can be differently achieved.

Here is the theme of our proposal, breaking the clock-tree uniformity paradigm.
1. Use as many as possible HVT drivers rather than LVT ones.
2. Spread and smooth the current waveform by mixing HVT and LVT driver in the same level of the tree, thus introducing some “disorder” in the commutations of the peak current.
3. Mix driver sizes at the same level of the tree, providing another degree of “disorder”.
4. Maintain acceptable clock signal slope and skew at sinks.

![Figure 6: Peak current waveform flattening (SPICE simulation). The ordinary tree in shown in red, while the green is the outcome of the tree proposed in this work.](image)

### 3. Characterization of clock-drivers
The algorithm that minimizes the peak current is two-phase, traversing the clock-tree \( T \) top-down first and then bottom-up. A node visited in the top-down phase determines the driver types of its sons, while the interconnect lengths are set to ensure equal delay at its sons. That requires iterative equation solving, involving extensive delay and slope calculations. Evaluating those parameters by invocation of SPICE simulation at each node and iteration within the node is unacceptably time consuming. Rather, one could characterize beforehand each type of clock-driver and then use composition of characteristics data. This is far computationally efficient,
whereas accuracy is hardly degraded, as shown later in the experimental results, comparing the algorithm computations with SPICE simulation.

![Diagram](image)

**Figure 7**: Setup for the clock-driver characterization by SPICE simulation.

![Graphs](image)

**Figure 8**: Driver’s characteristic data. (a) $S_{out}(s,c)$, (b) $T_{pd}(s,c)$ and (c) $I_{peak}(s,c)$.

Building the clock-drive characteristics takes place offline. The repertoire of library drivers to be used by the clock-tree is first decided. Each driver is then characterized by running extensive SPICE transient simulations in the setup shown in Fig. 7. The simulation results are tabulated for further usage within the top-down and bottom-up algorithm computations. This work used 40 nanometers process technology. The divers library is characterized at a $PVT=TTT$ corner where $P=\text{Typical}$ (typical process), $V_{dd}=1.1V$ (Typical supply voltage) and $T=25^\circ \text{C}$. Implications of other corners and process variations are also studied.

Fig. 8 illustrates the various driver parameters used by the algorithm. The following notations are in order. $S_{in}$ denotes the set of input slopes, ranges from 10pSec to 200pSec in steps of 8pSec. $C_{load}$ denotes the set of capacitive loads, ranges from 5fF to 250fF in steps of 50fF. Each slope-load pair $(s,c)\in S_{in} \times C_{load}$ is simulated with SPICE to obtain the characteristics of a driver.
Each \((s, c)\) point comprises the following characteristics, illustrated in Figs. 8 and 9.

1. \(S_{\text{out}}(s, c)\): driver’s output slope, measured from 10% to 90% of \(V_{dd}\).
2. \(T_{pd}(s, c)\): 50% to 50% driver’s propagation delay.
3. \(I_{\text{peak}}(s, c)\): driver’s peak current in milliamps.
4. \(T_{\text{peak}}(s, c)\): driver’s elapsed time, from 50% input switching till \(I_{\text{peak}}\) time.
5. \(T_{10\% \text{ rise}}(s, c)\): driver’s elapsed time, from 50% input switching till 10% \(I_{\text{peak}}\) time, for rising edge of the output current.
6. \(T_{10\% \text{ fall}}(s, c)\): driver’s elapsed time, from 50% input switching until 10% \(I_{\text{peak}}\) time, for falling edge of the output current.

The above characterization enables clock-tree construction and optimization involving accurate delay and current computations, without any in-line SPICE invocations as a part of the algorithm computations. The SPICE results for LVT and HVT types of a certain driver size, tabulated and stored in-memory by appropriate data structure, are shown in Fig. 8. The data is used afterwards for current superposition (accumulation), to obtain the current waveforms along the switching time period.

Clock network comprises not only drivers, but interconnects as well. Those strongly affect the current waveform and the clock skew. The interconnect setting includes their sizing, and where
the drivers are located along those. To avoid in-line SPICE simulations during optimization, the impact of the interconnecting wires is modeled by a closed-form expression.

Figure 10: Clock-tree topology and modifications.

Consider three successive levels of a clock-tree illustrated in Fig. 10(a) and its corresponding graph representation in Fig. 10(b). A driver at level \( n \) of the tree is denoted by \( v_n \). Fig 10(c) shows a modification of the physical location of \( v_n \) within the tree. While some wires are shortened, others become longer. To model the delay changes occurring by re-positioning drivers, the model in Fig. 11 is used. Considering three successive drivers \( v_{n-1}, v_n \) and \( v_{n+1} \), let the location of \( v_{n-1} \) and \( v_{n+1} \) be fixed, while \( v_n \) is allowed to be arbitrarily located in between.

Figure 11: Delay modeling of driver re-positioning.
The resulting delay obtained by SPICE is shown in Fig. 12. It linearly depends on the location of $v_n$. Four clock-drivers, two LVT and two HVT, of different strength were simulated. Linear behavior is shown across wide range of wire lengths and clock-driver types. The linear behavior adheres the following equation

$$\Delta_{\text{delay}} = T_1(v_n) - T_2(v_n) = K_{\text{delay}} \cdot \Delta x, \hspace{1cm} (3)$$

where $\Delta x$ is wire length change in microns and $K_{\text{delay}}$ is a factor measured in picoseconds per micron, obtained by SPICE simulations. By using (3) to account for the delay changes occurring by displacing drivers, the clock-tree construction and optimization algorithm avoids the expensive simulations. It is important to note that $K_{\text{delay}}$ is another characteristics data of a driver. Similarly, the output slope $S_{\text{out}}$ of $v_n$ is changed due to repositioning $v_n$, and a similar relation exists as follows

$$\Delta_{\text{slope}} = S_1(v_n) - S_2(v_n) = K_{\text{slope}} \cdot \Delta x. \hspace{1cm} (4)$$

The parameter $K_{\text{slope}}$ is another characteristic data of a driver, obtained by SPICE simulation shown in Fig. 13 to linearly depend on the location of $v_n$. 

Figure 12: The delay dependency on driver position.
A key feature of the peak current minimization algorithm (elaborated in Section 7) is the maintenance of accurate values of $T_{pd}(s,c)$, $I_{peak}(s,c)$ and $S_{out}(s,c)$. For each node traversal of Fig. 10(c), the algorithm employs delays and slopes fast calculations by a superposition of the values calculated for the parent with those of the new driver being traversed.

**4. Peak current reduction by clock-driver mixing**

The key idea of peak current reduction is in mixing HVT and weak LVT drivers, rather than solely LVT drivers commonly used in clock-trees. We use the notation LVT/2 to denote weak (half size) LVT driver. The idea is illustrated in Fig. 14. On the top there is a fork of an ordinary clock-tree, comprising only LVT drivers. The tree proposed by this work is using one of the two possible structures shown at the bottom. This is how the red waveform shown in Fig. 6 is being spread and replaced by the green waveform, having considerably smaller peak value. The peak current reduction is achieved by deteriorating the uniformity of the clock-tree, which misaligns the drivers’ switching time. For a tree with $n = 2^N$ sinks, the structure proposed in Fig. 14 obtains $n$ distinct root-to-sink paths, comprising a mix of HVT and LVT/2 drives. The leftmost root-to-sink path is purely LVT/2, whereas the rightmost is purely HVT. An example of an eight leaves tree is shown in Fig. 15.
Let $T(V,E)$ be the clock-tree (see Fig. 10(b)), which nodes $V$ are the clock drives. The terms nodes and drivers are used interchangeably. The edges $E$ are the wires connecting parents to sons. The set $Q \subseteq V$ denotes $T$ ’s leaves, where the FFs are connected. $T$ is assumed to be a binary balanced tree (see Fig. 5). This is a common assumption in the development of algorithms, which does not limit the generality of the discussion. Each node $v \in V$ is associated with the following parameters, accumulated along the tree traversal, using the characteristic data of driver types, shown in Fig. 8.

1. $c(v)$: capacitive load driven by $v$.
2. $T_{pd}(u,v)$: propagation delay from the output of the parent $u$ to the output of its son $v$.
3. $T_{pd_{tot}}(v)$: propagation delay from $T$ ’s root to the output of $v$.
4. $s(v)$: voltage slope at $v$ ’s output.
While the peak current is reduced, the clock skew at the sinks, defined by the difference between the maximum and minimum root-to-sink latency, should be maintained within a prescribed limit. This could be achieved if the propagation delays from the root of the two branches in Fig. 14 to their far end will stay equal, thus avoiding skew accumulation. Fig. 16 shows the propagation delays of the two branches obtained by SPICE simulation. The magnified crossing point of the two voltage responses shows that the 50% to 50% delay difference between the two branches is 3.6 picoseconds, which is less than 0.5% of 1GHz clock frequency. Similar behavior was observed for all the clock-drivers in 40 manometers technology. Another important advantage is the leakage current reduction due to the HVT and weaker LVT/2 drivers.

Figure 16: Propagation delays equalities of HVT and LVT/2 branches.

The simulation in Fig. 16 used equal loads at the far ends of the two branches. This however is not sufficient to guarantee that the root-to-leaf delays of the entire tree will stay within acceptable skew. The peak current reduction algorithm by its very nature displaces the location of drivers in its top-down phase. This in turn changes the loads seen by the branches, and their propagation delays are therefore not equal any more. To this end a bottom-up phase maintains the global skew within a prescribed limit by further driver displacement. Though local delay inequalities at branches occur, those do not matter, as after all what counts is the global root-to-leaf delays and if those are made equal, the goal of small skew is achieved.

We shall use the property of delay linearity shown in (3) and Fig. 12 to obtain zero local skew at the top-down traversal phase. Though not guaranteeing global zero skew (root-to-leaf), it is a first step towards skew convergence. Skew corrections are performed by a bottom-up traversal, where a fine tuning of the drivers’ location (wire stretching) fixes the skew occurred due to the blindness of the top-down traversal to the global delay.
Fig. 17 shows how the peak current is reduced by the mix of HVT and LVT/2 drivers. The left waveform is of a nominally sized LVT driver, used by common clock-trees. The center waveform is of two successively connected LVT/2 drivers and an HVT one, while the left waveforms shows the entire current pulse drawn from the supply. Comparing the original and the modified waveforms, a reduction of 43% in the peak is observed. Notice that at the root of the tree, the fork of 2xLVT/2 and HVT shown in Fig. 16 may extend along tens to hundreds of microns, while close to the leaves it is only few microns.

5. Analytic solution of peak current reduction
Before addressing in details the numerical algorithm in Section 6, it is shown that with some simplifications the peak current reduction subject to skew zeroing problem can be presented in algebraic form. The peak current is primarily reduced by the tree modifications shown in Fig. 14. The exact locations of the drives have a small influence on the peak current. Drives positions along the branches however have a primary impact on the skew. To this end we subsequently propose a simple, yet effective, model of tree branches adjustment, yielding nearly zero skew.
Consider the $RC$-delay model of the fork illustrated in Fig. 18(a), where the root is either HVT or LVT/2 driver, with internal resistance $R_{tr}$. The model comprises two types of driver-to-driver interconnects. Fig. 18(b) models the delay from the root driver to the HVT driver and the first LVT/2 one, while Fig. 18(c) models the delay from the first LVT/2 driver to the second one. $C_{g(H)}$ and $C_{g(L/2)}$ denote the input gate capacitance of HVT and LVT/2 delivers, respectively. We use the notation $R_{int(H)} = r_{int} x_H$ and $C_{int(H)} = c_{int} x_H$ for resistance and capacitance, respectively, of the wire connecting the root to the HVT driver. $r_{int}$ and $c_{int}$ are per-micron resistance and capacitance coefficients, respectively, and $x_H$ is a wire length variable. Similar conventions are used for LVT/2 wire connection.

Using Elmore delay model [25 Ch. 5], the delay $\tau_H$ from the root to the HVT driver is

$$\tau_H = R_{tr} \left( c_{int} x_{L/2} + C_{g(L/2)} \right) + \left( R_{tr} + \frac{r_{int} x_H}{2} \right) \left( c_{int} x_H \right) + \left( R_{tr} + r_{int} x_H \right) C_{g(L/2)} =$$

$$\frac{r_{int} c_{int}}{2} x_H^2 + R_{tr} c_{int} (x_H + x_{L/2}) + \left( r_{int} C_{g(H)} + R_{tr} c_{int} \right) x_H + R_{tr} \left( C_{g(H)} + C_{g(L/2)} \right) \quad (5)$$

The delay $\tau_{L/2}'$ from the root to the first LVT/2 gate is the following

---

**Figure 18:** Delay model of a clock-driver fork.
\[
\tau'_{L/2} = R_{tr} \left( c_{int} x_H + C_{g(H)} \right) + \left( R_{tr} + \frac{r_{int}^2 x_{L/2}}{2} \right) \left( c_{int} x_{L/2} \right) + \left( R_{tr} + r_{int} x_{L/2} \right) C_{g(L/2)} =
\]
\[
\frac{r_{int} c_{int}}{2} x_{L/2}^2 + R_{tr} c_{int} \left( x_H + x_{L/2} \right) + \left( r_{int} C_{g(L/2)} + R_{tr} c_{int} \right) x_{L/2} + R_{tr} \left( C_{g(H)} + C_{g(L/2)} \right)
\]

Fig. 18(c) shows the delay model from the first to the second LVT/2 drivers. The delay \( \tau''_{L/2} \) is given by

\[
\tau''_{L/2} = \left( R_{tr(L/2)} + \frac{r_{int} x_{L/2}}{2} \right) c_{int} x_{L/2} + \left( R_{tr(L/2)} + r_{int} x_{L/2} \right) C_{g(L/2)} =
\]
\[
\frac{r_{int} c_{int}}{2} x_{L/2}^2 + \left( R_{tr(L/2)} c_{int} + r_{int} C_{g(L/2)} \right) x_{L/2} + R_{tr(L/2)} C_{g(L/2)}
\]

Figure 19: Indexing of the clock-tree wire length variables and constraints.

Having the expressions \( \tau_H \), \( \tau'_{L/2} \) and \( \tau''_{L/2} \) in (5)-(7) involved in a fork, the root-to-leaf path delay of every sink can be calculated by summing the appropriate branch delays. Consider for instance the clock-tree of Fig. 19, comprising two successive levels. Fig. 19(a) shows an ordinary clock-
tree with a fork length parameter $D$, and a level-to-level length reduction by factor two (see the H-tree in Fig. 5). Fig. 19(b) is the modified tree, where wire lengths variables $x_{x_{x_{x}}}$ are introduced to tune the root-to-leaf delays for skew nullifying. The solution (not necessarily unique) finds appropriate locations of drivers along their branches. Driver-to-driver interconnects lengths are variables, where their partial sums must satisfy branch lengths constraints, dictated by the methodology of level-to-level wire length reduction by factor two.

Let the clock-tree have $n$ leaves. To keep track of the wire-length variables, we adopt the triplet indexing notation $\langle m, k, l \rangle$ of the drivers, where, $1 \leq m \leq \log_2 n$ is the tree level, $1 \leq k \leq 2^m$ is the driver’s index within the level, and $l \in \{1, 2\}$ is an index, used to distinguish between the two cascaded LVT/2 drivers along their branch (see Fig. 18(b)). For the sake of uniformity an HVT driver is always indexed by 1. Driver’s incoming wires inherit the driver’s $\langle m, k, l \rangle$ index. In the above convention the wires connected to the output of the leaf drivers in Fig 19(b) are assumed at level 3.

To zero the skew, all the root-to-leaf latencies must be the same, say $T_{\text{latency}}$ (another degree of freedom). One can derive from (5)-(7) $n$ root-to-leaf equations, whose matrix notation is

$$A\bar{X}^2 + B\bar{X} + C - T_{\text{latency}} I = 0.$$  \hspace{1cm} (8)

$\bar{X}$ in (8) is the vector of wire-length variables, whose length is $4n-3$, $A$ and $B$ are matrices of size $n \times (4n-3)$ and $C$ is a vector of length $n$. Their entries are derived from (5)-(7).

The solution of (8) must satisfy the geometric constrained imposed by the pre-defined tree length, dictated in Fig. 19(a). Those constraints are of two types. The first type comprises $n$ constraints accounting for the tree’s total length, given by $\sum_{m=0}^{\log_2 n} D/2^m = D(2n-1)/n$. The second type constraints are aimed at maintaining the relative length of the forks according to their level. Recalling that a fork length is inherited from its parent fork by the factor half, it is only the first driver in the LVT/2 branch that has the freedom to be displaced within the fork, as shown in Fig. 19(c). That yields other $n-1$ constraints. All in all there are $2n-1$ wire length constraints, presented by

$$F\bar{X} = \bar{L},$$  \hspace{1cm} (9)

where $F$ is a $(2n-1) \times (4n-3)$ matrix and $\bar{L}$ is the constraints vector of length $2n-1$. If the system of $n$ quadratic equations in (8) and $2n-1$ linear equations in (9), of the $4n-2$ variables ($T_{\text{latency}}$ is a variable too) has a real solution, zero skew is achieved.

6. Algorithm for clock-tree construction
We subsequently present an algorithm to reduce the peak current and maintain minimum skew, by the tree modifications proposed in Fig. 14 and wire lengths adjustments illustrated in Section 5. The algorithm comprises two phases. The first is a top-down traversal. Its primary goal is to cut the peak supply current. It also maintains small skew in an ad-hoc manner, ensuring it does not escape at the tree’s leaves. That is a good starting point to a second phase of bottom-up traversal, aiming at skew nullifying by fine adjustments of the clock-drivers positions in the tree branches. The second phase has a very small impact on the peak current, which has already been reduced in the first phase.

Starting at the root, for each visited node, the top-down traversal does the following:
1. Substitution of a fork mixing HVT and LVT/2 drivers as illustrated in Fig. 14.
2. For both the HVT and LVT/2 fork’s branches, the cumulative delay from the root, and the slope at the branch driver’s output are calculated.
3. The position of the HVT driver is adjusted to ensure equal delay from the root to the far end HVT and LVT/2 drivers. Delay equalization is done by numerical, binary search iterations, using the driver’s characteristics presented in Section 3.

Starting at the leaves, for each visited node, the bottom-up traversal does the following:
1. It calculates the delay deviations which have been resulted at the top-down phase. This deviation occurs due to the blindness of the top-down calculations to the loads of the drivers in the HVT and LVT/2 branches, as such loads are affected by the position adjustments of the HVT driver that has not been decided yet.
2. Nullifying the skew at the node, by fine adjustments of the locations of the LVT/2 and the HVT drivers.

6.1 Current waveform construction
Once the bottom-up traversal completes, the entire clock-tree is determined. The current waveform at the tree’s root can be accurately computed by cumulating the currents drawn by the individual drivers. The skew supposed to be small, though not yet meeting the prescribed limit.
Figure 20: Current waveform build-up, approximation vs. accurate.

The current waveform construction by a superposition of the currents at each driver of the clock-tree is presented, similar to [8, 10]. Fig. 20 presents the strength of the method, the accuracy of the driver characterization, and the robustness of the RC model used by this work. The figure illustrates the waveform of a purely LVT 5-level tree (32 drivers at leaves). The green waveform is obtained by SPICE simulation, whereas the red one is the approximation, utilizing driver’s characteristic data and the RC delay model. Very good fit is observed. The experimental results section shows good fitness across a wide range of trees.

Figure 21: Current waveform construction.

Figure 22: Approximating the current pulse at a driver.
Fig. 21 shows how a current waveform is constructed for the modified tree of 8 sinks (leaves). Such a tree has a total of \((8-1) \times 3 = 21\) HVT and LVT/2 divers. The current drawn at each drivers and its relative time alignment is progressively computed from top-to-bottom. The slope and capacitive loads at each node are calculated, and with the aid of the \((s,c)\) slope-load driver characteristics (see Section 3, Figs. 5(a)-(c)), the currents and their relative time alignments are derived. Based on \(I_{\text{peak}}, T_{\text{peak}}, T_{10\% \text{ rise}}\) and \(T_{10\% \text{ fall}}\) defined in Section 3, a triangular approximating the driver’s current waveform is defined. That triangle is defined by three time parameters \(t_s, t_p\) and \(t_e\), shown in Fig. 22. Notice their relation to \(T_{\text{peak}}, T_{10\% \text{ rise}}\) and \(T_{10\% \text{ fall}}\) in Fig. 9.

7. Algorithm details
We subsequently describe in details steps algorithm, with emphasis on the computational aspects. The top-down peak current reduction is described first, followed by the bottom-up drivers’ position adjustment for skew nullifying.

7.1 Top-down peak current reduction
The primary goal of the top-down phase is to reduce the peak current. It also maintains the skew within reasonably small value, which can afterwards be nullified by further tuning. Fig. 23 illustrates the fork substitution with respect to an ordinary clock-tree, where the various points are numbered for reference. We denote by \(l_{i,j}\) the length between two points \(i\) and \(j\), and by \(\delta_{i,j}\) the 50% to 50% propagation delay along a wire connecting \(i\) with \(j\). \(\delta_H\) and \(\delta_{L/2}\) denote the intrinsic delay of HVT and LVT/2 clock-drivers, respectively. Recall that \(\delta_H\) and \(\delta_{L/2}\) are functions of their output load and input slope, and can be obtained by a two-dimensional linear interpolation of the characteristic data described in Section 3.

![Figure 23: Wire length budgeting in fork substitution.](image-url)
Fig. 23(a) shows the relations between the lengths of an ordinary tree’s branches, adhering the level-to-level length reduction by factor two (see the H-tree in Fig. 5). There is

\[ D_n = \frac{D_{n-1}}{2}, \]  

where \( n \) denotes the level of the tree (see Fig. 10). The substituted tree in Fig. 19(b) maintains wire length invariance by imposing the following 3D/2 total length on two successive tree levels

\[ \frac{3D}{5} + \frac{3D}{5} + \frac{3D}{10} = \frac{3D}{2}, \quad \text{and} \quad \left( \frac{3D}{2} - x \right) + x = \frac{3D}{2}, \]

where \( x \) is a variable set at the top-down phase to ensure delay equality of the two fork’s branches. There is some blindness of the far end capacitive loads since fork substitution in the successive level of the tree will change the length of the far end wires. The temporary top-down delay equalization ensures the global skew measured at the tree’s leaves is not escaping, such that the bottom-up wire lengths fine tuning can recover the skew to be nearly zero.

Referring to the numbers in Fig. 23(b), the delay equalization is obtained by the substitution

\[ \delta_{1,2} + \delta_{H(2,3)} = \delta_{1,6} + \delta_{L/2(6,7)} + \delta_{7,8} + \delta_{L/2(8,9)}. \]  

(11)

Notice that the drivers’ intrinsic delays \( \delta_{H(2,3)} \), \( \delta_{L/2(6,7)} \) and \( \delta_{L/2(8,9)} \) in (11) are associated with the position in the tree’s branch, since those delays depend on their input slope and output capacitive load, varying along the branch. For the wire delays of (11) there is

\[ \delta_{1,6} = \delta_{7,8} = \left( \frac{3D}{5} \right)^2 \frac{r_{\text{int}} C_{\text{int}}}{2} + \left( \frac{3D}{5} \right) r_{\text{int}} C_g, \]  

(12)

\[ \delta_{1,2} = x^2 \frac{r_{\text{int}} C_{\text{int}}}{2} + x r_{\text{int}} C_g, \]  

(13)

where \( r_{\text{int}} \) is the wire’s resistance per micron and \( C_{\text{int}} \) is its capacitance per micron, both taken from TSMC for 40 nanometer technology. \( C_g \) is the gate capacitance of the TSMC drivers which has been used for this study. For (12) and (13) we used the interconnect delay model in [25, Ch. 5].

The intrinsic delays of the drivers are obtained from the characteristic data. Shown in Fig. 8, those are discrete functions of the input slope and the output load. To find \( \delta_{H(2,3)} \), \( \delta_{L/2(6,7)} \) and \( \delta_{L/2(8,9)} \) in (11), the slope at point 1 in Fig. 23 is known from the p fork calculation, and the wire loads 1-2 and 1-6 can be calculated from \( r_{\text{int}} \), \( C_{\text{int}} \) and \( C_g \). Those are used to propagate the slope from point 1 to 2 and to 6. The loads driven by the drivers at points 2 and 6, comprising the wires and gate capacitance of the drivers connected at their far ends, can also be calculated. Shown in
Fig. 24, by knowing the input slopes $S_{in}$ and the output loads $C_{load}$, a two dimensional interpolation finds the intrinsic delays of the drivers connected at points 2 and 6.

By using four adjacent grid points of Fig. 24, indexed (1,1), (1,2), (2,1) and (2,2), the following expression interpolates the propagation delay $T_{pd}$ at points 3 and 7 in Fig. 23.

$$T_{pd}(S_{in}, C_{load}) = \left[ \frac{T_{pd}^{1,1}(S_{in}^{2,1} - S_{in}^{1,2})(C_{load}^{1,2} - C_{load}) + T_{pd}^{2,1}(S_{in}^{1,1} - S_{in}^{1,1})(C_{load}^{2,2} - C_{load})}{(S_{in}^{2,1} - S_{in}^{1,1})(C_{load}^{2,2} - C_{load})} \right]. \quad (14)$$

Similarly, a two-dimensional interpolation for the slopes $S_{out}$ at points 3 and 7 in Fig. 23 is

$$S_{out}(S_{in}, C_{load}) = \left[ \frac{S_{out}^{1,1}(S_{in}^{2,1} - S_{in}^{1,2})(C_{load}^{1,2} - C_{load}) + S_{out}^{2,1}(S_{in}^{1,1} - S_{in}^{1,1})(C_{load}^{2,2} - C_{load})}{(S_{in}^{2,1} - S_{in}^{1,1})(C_{load}^{2,2} - C_{load})} \right]. \quad (15)$$

To complete the fork calculations, it is required to further propagate the delay and slope from point 7 to 8, and then from 8 to 9. Those are obtained by similar techniques and equations (12)-(15).

Fig. 23(b) shows that the fork calculations depend on $x$, which supposed to solve (11). The solution can be obtained by any search method, where iterations require re-calculation of the fork’s right branch delays, using drivers’ characteristics for computational efficiency’ avoiding run-time explosion if SPICE simulation have been used. We used binary search to solve (11).

Fork substitution takes place at each tree’s node. Once done, substitution of its left and right sons takes place. We subsequently detail the calculations of the left fork at point 9, shown in Fig. 25.
The first step is to define the fork’s total length budget. For point 1 that was $D + D/2 = 3D/2$. For the left fork at point 9 that would be $3D/10 + D/4 = 11D/20$. This is shown by the distance between points 9 and 11, supposed to consume $2/5$ of that budget, yielding altogether $2/5 \times 11D/20$.

In the right branch the length $z$ between points 9 and 10 (denoted by $x$ in the parent) is sought such that delays equality as in (11) is solved for the fork emanating at point 9. By similar considerations, the length budget of the right son fork is $(3D/2 - x) + D/4 = 7D/4 - x$. Then the branch length $y$ between points 3 and 4 is sought to solve (11).

![Diagram of wire length solution](image)

Figure 25: Wire length solution to ensure delay equalities at sibling son drivers.

To solve (11) for the entire tree, a top-down tree traversal can be implemented by a recursive call to an appropriate software function. There is a delicate point though. The solution of (11) for a given fork (tree’s node) is blind to the later solution that will take place in its son forks, while the capacitive loads assume the default positions setting in Fig. 23(b). Those are adjusted by subsequent invocations of the recurrence, so the delays and slopes calculated at points 3 and 9 in Fig. 23(b) are inaccurate. The slope is corrected simultaneously with the son fork solution by applying the linearization in (4). For practical reasons and computational efficiency, the delay correction in (3) takes place at the bottom-up phase, as it anyway re-adjusts the length connecting point 7 to 8 for global skew zeroing, as subsequently described.

### 7.2 Bottom-up delay correction

The blindness of the top-down fork substitutions to the downstream loads resulted delay inaccuracies. This in turn may lead to non-negligible clock skew at the tree’s leaves. While the slopes at points 3 and 9 have already been precisely computed at the top-down traversal, it is the role of the bottom-up phase to resolve delay discrepancies to yield nearly zero skew. To this end, position adjustments of the LVT/2 driver residing between points 6 and 7 in Fig. 25 takes place. The wire length between points 1 and 9, dictated by the top-down traversal, remains unchanged.
When the bottom-up phase is invoked, the lengths $x$, $y$ and $z$ shown in Fig. 25, are already determined for the entire tree.

Comparing Fig. 25 to 23(b), the delay discrepancy at point 3 occurs due to the difference between the length $3D/2 - x$ which has been used for the top-down delay calculation from point 3 to points 4 and 5, while the substitution of the son fork changed those to $y$ and to $2/5(7D/4 - x)$, respectively. The accurate delay at 3 can be found from equation (3) by accounting for the differences $y - (3D/2 - x)$ and $2/5(7D/4 - x) - (3D/2 - x)$ at the right branch left branches, respectively. The delay correction at point 3 is therefore

$$
\Delta_{\text{delay}}(\delta_{H(2,3)}) = K_{\text{delay}}(\delta_{H(2,3)}) \left[ y + \frac{2}{5} \left( \frac{7D}{4} - x \right) - 2\left( \frac{3D}{2} - x \right) \right].
$$

(16)

The delay correction at point 9 is

$$
\Delta_{\text{delay}}(\delta_{6,9}) = K_{\text{delay}}(\delta_{6,9}) \left( z + 11D/50 - 2 \cdot 3D/10 \right),
$$

(17)

where $\delta_{6,9}$ is a compound linearization factor between points 6 and 9.

Fig. 26(a) illustrates a fork and its left and right emanating sub-trees. The equality of all the latencies in a sub-tree is the invariant of the bottom-up latency equalization process. Fig. 26(b) exemplifies the three lowermost levels of a sub-tree, where $T_{pd1}$, $T_{pd2}$, $T_{pd3}$ and $T_{pd4}$ are the root-to-leaf latencies. Notice that each of the four lowest forks at points 4, 5, 6 and 7 satisfies the invariant property of left and right branch delay equalities by the to-down construction. That happens since the loads at the leaves are the FFs clock input loads, which are accurately known. The operation taken at point 2 in Fig. 26(b) aims at equating $T_{pd1}$ with $T_{pd2}$, and similarly at point 3 for $T_{pd3}$ with $T_{pd4}$, yielding the situation shown in Fig. 26(a).
The evaluation of (16) and (17) ensures that the latencies $T'_{pd}$ and $T''_{pd}$, shown in Fig. 26(a), are accurately known (though not equal yet). Once $T'_{pd}$ with $T''_{pd}$ are equalized, they will stay so regardless of any further bottom-up manipulation at higher levels. We subsequently describe how $T'_{pd} - T''_{pd}$ is nullified. In terms of Fig. 26 the solution of

$$
\delta_{1,2} + \delta_{H(2,3)} = \delta_{1,6} + \delta_{L/2(6,7)} + \delta_{7,8} + \delta_{L/2(8,9)} + \left[\left(T'_{pd} - \Delta_{\text{delay}}(\delta_{H(2,3)})\right) - \left(T''_{pd} - \Delta_{\text{delay}}(\delta_{6,9})\right)\right] \quad (18)
$$

is in order. Equation (18) is numerically solved by a binary search of the position of the LVT/2 driver connected between points 6 and 7 in Fig. 25, where the wire length between points 1 and 8 is not changed. The solution of (18) also uses driver characteristics. Another delicate point to consider is that the solution of (18) changes the load seen at point 1 in Fig. 25, which in turn changes the slope at the point. Those variations are accounted and maintained in the numerical solution with the aid of the linearization in (4).

### 7.3 Putting everything together in a code

Sections 7.1 and 7.2 described the top-down and bottom-up numerical node solution, respectively. Though those involve different computations, they can be captured in a single recursive traversal procedure. As the recursion dives (top-down) into a node, the fork substitution of Section 7.1 takes place. As the recursion returns from a sub-tree to its root node (bottom-up), the left and right delay equalization of Section 7.2 takes place. An appropriate pseudo code is shown in Fig. 27. It is called for every node $v$ where a fork is substituted, passing the following parameters:

1. $s(v)$: input slope of $v$, 

---

Figure 26: Invariants of the bottom-up latency equalization.
2. $T_{pd\_tot}(v)$: latency from tree’s root to $v$.
3. LenBudget: fork length budget substituted at $v$, passed from the parent node, and
4. $D$: additional budget of the fork length.

Each node of the tree (driver’s output), maintains an internal data structure, storing slope and root-to-node latency, length of the left and right outgoing wires, delay and slope linearization factors used in (3) and (4), respectively, and driver’s type. The code below refers to the notations of Fig. 23 and Fig. 25.

**DelayType:**

```c
DelayType( s(v), T_{pd\_tot}(v), LenBudget, D ) {
    LenBudget = LenBudget + D; // update fork’s wire length budget
    (x_{right}, s(v_{right}), s(v_{left}), T_{pd\_tot}(v_{right}), T_{pd\_tot}(v_{left}), K_{delay}, K_{slope}) =
        ForkCalc( s(v), T_{pd\_tot}(v), LenBudget ); // apply fork delay and slope calculations
    NodeUpdate( x_{right}, x_{left}, T_{pd\_tot}(v), “top-down”);
    SonUpdate( s(v_{right}), x_{right}, T_{pd\_tot}(v_{right}) ); // right son lengths, delay and slopes updates
    SonUpdate( s(v_{left}), x_{left}, T_{pd\_tot}(v_{left}) ); // left son lengths, delay and slopes updates
    // traverse right sub-tree
    Delta_{delay}(\delta_{H(2,3)}) =
        NodeTraverse( s(v_{right}), T_{pd\_tot}(v_{right}), LenBudget - x_{right}, D/2 );
    // traverse left sub-tree
    Delta_{delay}(\delta_{6,9}) =
        NodeTraverse( s(v_{left}), T_{pd\_tot}(v_{left}), 1/5\times LenBudget, D/2 );
    // fixes delay discrepancy by finding the appropriate location of LVT/2 driver
    if ( Delta_{delay}(\delta_{H(2,3)}) \neq 0 \text{ or } Delta_{delay}(\delta_{6,9}) \neq 0 ) {
        (x_{left}, Delta_{delay}(\delta)) = DelayCorrection( s(v), T_{pd\_tot}(v), x_{right}, LenBudget );
        // updates lengths, delay and slopes of node
        NodeUpdate( x_{left}, LenBudget, T_{pd\_tot}(v), “bottom-up”);
        return ( Delta_{delay}(\delta) );
    }
    else { return 0; }
} // NodeTraverse
```

Figure 27: Pseudo code of recursive node traversal.
The recursive function NodeTraverse is being called for the root of the clock-tree. Upon completion, the accurate root-to-node delay, slope and wire capacitive loads, are known for each node. Those, together with the aid of the characteristic data, enable the buildup of the current waveform by the triangle superposition described in Section 6.1.

8. Experimental results

Several clock-trees have been implemented, using the fork substitution algorithm. For each of those trees the current pulse waveform, the skew, and the sensitivity of the skew to input slope, have been measured in two manners. First, the computational engine of the algorithm together with the driver characteristics, have been used. The resulting tree was then simulated with SPICE. Very good correlation between the two across a wide range of clock-trees is shown.

8.1 Peak current reduction and computation accuracy

Fig. 28 shows from left to right the peak current achieved by the fork substitution algorithm for 3-level, 4-level and 5-level clock-trees, comprising 8, 16 and 32 sinks, respectively (green waveforms), compared to the ordinary clock-tree (red waveforms). Both the ordinary and modified trees have been simulated in SPICE.

![Figure 28: Peak current reduction of clock-trees obtained by fork substitution algorithm.](image)

Notice the clear local peaks in the ordinary tree, aligned in time according to the tree levels, where the highest peak occurs at the leaves, comprising half of the total drivers count. It can be observed that 30% to 45% peak current reduction was achieved. While the global clock network may be a mesh or spine, clock-trees are widely used for the local clock distribution [9]. Since such trees are more or less uniformly distributed across the entire silicon area, it ensures the effectiveness of the power noise reduction. Expectedly, the latency of the modified waveform is
longer than the original one in a reciprocal proportion of the peak current reduction, since the area under the waveform (charge) should be about the same.

![Figure 29: Accuracy of fork substitution algorithm compare to SPICE.](image)

### 8.2 Model accuracy

To validate its accuracy, the current waveforms obtained by the fork substitution algorithm (red waveforms) were compared to those obtained by SPICE simulation (green waveforms). Good fitness is shown in Fig. 29, where accuracies of 10% for 3-level and 4-level trees and 3% for 5-level tree were measured.

### 8.3 Small skew validation and slope variations

To validate that nearly zero skew has been achieved, the response of a clock pulse was measured at the leaves by SPICE simulations, shown in Fig. 30. The 50% $V_{dd}$ crossings are encircled. It is clearly seen that all responses are crossing 50% $V_{dd}$ almost simultaneously, showing that the small skew premise is achieved.
Another important aspect of the skew is its sensitivity to the slope of the clock signal driving the root of the tree. To this end the signal with slopes varying from 28pSec to 64pSec were simulated with SPICE. The results shown in Fig. 31, indicate that for a wide range of input slopes, the skew is maintained in the range of 1pSec to 8pSec. For clock cycle of 1GHz this is less than 1% of the clock cycle in the worst case. Furthermore, Fig. 31 shows well behaviour of the algorithm with respect to the tree size, maely the skew behaves similarly for the three trees.

References:


