Non-Uniform Clock Mesh Optimization with Linear Programming Buffer Insertion

Matthew R. Guthaus
University of California Santa Cruz
mrg@soe.ucsc.edu

Gustavo Wilke
PGMicro - Universidade Federal do Rio Grande do Sul
Porto Alegre, Brazil
wilke@inf.ufrgs.br

Ricardo Reis
PGMicro/PPGC - Universidade Federal do Rio Grande do Sul
Porto Alegre, Brazil
reis@inf.ufrgs.br

ABSTRACT

Clock meshes are extremely effective at filtering clock skew from environmental and process variations. For this reason, clock meshes are used in most high performance designs. However, this robustness costs power. In this work, we present a mesh edge displacement algorithm that is able to reduce mesh wire length by 7.6% and overall power by 10.5% with a small mean skew improvement. We also present the first non-greedy buffer placement and sizing technique using linear programming (LP) and iterative buffer removal. We show that compared to prior methods, we can obtain 41% power reduction and an 27ps mean skew reduction on average when variation is considered compared to prior algorithms.

Categories and Subject Descriptors
J.6 [Computer-Aided Engineering]: Computer-Aided Design

General Terms
Algorithms, Design

Keywords
Clock mesh optimization, robust design

1. INTRODUCTION

Process and environmental variations are a great challenge to clock network designers. Variation effects on the clock network can not be predicted and, hence, they are difficult to directly consider in the design stage. Clock structures with redundancy such as the meshes [3,9,13], spines and cross-links [14] are an effective way to reduce variation effects. Many microprocessor designs use clock meshes to reduce the clock skew variation. These include a 1.2GHz Alpha processor [16], the Power4 [10], the Power6 [12], and a Dual-Core SPARC V9 [6]. Mesh-based clock distribution architectures are also supported by commercial tools in ASIC design flows [2].

A clock mesh is a grid of wires to which the clock sinks (flip-flops) are directly connected. In this paper, we use the terms grid and mesh interchangeably. A mesh is usually driven by a clock tree from a single clock source. Re-convergent paths created by the mesh structure are able to compensate the different clock signal arrival times at the mesh inputs. However, these re-convergent paths also produce short circuit currents between the mesh drivers. These short-circuit currents, in addition to the high capacitance associated with the mesh wire structure, are responsible for the higher power consumption in comparison to clock tree networks.

This work has two major contributions:

1. This is the first paper to consider non-uniform mesh placement by moving edge locations.
2. This is the first paper to present a non-greedy buffer insertion algorithm for clock meshes that considers mesh resistance in addition to capacitance.

By moving horizontal and vertical mesh wires towards the larger concentration of clock sinks the average stub length is reduced and the total clock mesh capacitance is reduced. Having less wire leads to improved robustness. Our buffer insertion algorithm uses linear programming (LP) and skew sensitivities to buffer size as a guide for buffer insertion. Our buffer algorithm performs sizing in a continuous domain and then rounds to a standard size library (or it can be implemented with a custom library). The power and clock skew achieved by our algorithms are significant improvements over [13] and [9].

The remainder of the paper is organized as follows: In Section 2, some important definitions and prior work are presented. In Section 3, the proposed algorithm for non-uniform mesh generation is presented. In Section 4, the buffer insertion algorithm is presented. Sections 5 and 6 present our experimental methodology and results, respectively. Finally, we offer conclusions in Section 7.

2. BACKGROUND

A clock mesh is composed by a set of buffer connected to a grid of vertical wires (column) and horizontal wires (row). The point where a vertical line crosses an horizontal line is referred to as a mesh node. The wires through which the clock elements are connected to the clock mesh are called stubs. The clock mesh size is defined by the number of rows and columns in the clock mesh, a clock mesh with m rows and n columns will be referred to as a \( m \times n \) clock mesh. For sake of simplicity, in the scope of this work, we will work only with square meshes (i.e. a clock mesh with the same number of rows and columns). The mesh size for a square clock mesh with \( m \) rows and \( m \) columns is \( m \). A uniform clock mesh is a clock mesh where rows and columns are evenly spaced over the sink area.
Clock meshes are a recent topic of research for the design automation community. There have been three major publications on automated clock mesh synthesis [3, 9, 13]. Desai et al [3] proposed a network flow method for clock mesh wire sizing and reduction. This method requires an initial clock mesh and buffer insertion that meets skew constraints and minimizes power given the skew bound. More recently, Venkataraman et al [13] presented fast algorithms for initial mesh design using greedy buffer insertion and mesh reduction through network survivability. In [9], an initial mesh size selection algorithm is presented along with improvements for both the mesh buffer placement and mesh reduction algorithms presented in [13].

3. NON-UNIFORM MESHES

For all previous designs in which clock meshes are used, mesh wires are routed uniformly over the clock sink region. Although the uniform mesh is the most common way to route a mesh it is not necessarily the best method in many cases. If, for example, sinks are not uniformly distributed, it is better to have more mesh wires near the denser regions and fewer wires near the sparser regions.

Previous methods have addressed this issue by creating relatively dense meshes to start and then removing unnecessary edges later. This, however, has the disadvantage that edge removal in the mesh is highly dependent on the buffer locations. Instead, we propose to change the mesh structure from the very beginning which will result in lower mesh wire length and less skew. We define a non-uniform clock mesh as a clock mesh where rows and columns can be unevenly spaced over the mesh area. For this work, however, we still assume that the number of rows and columns is the same, but this is not a limitation and is a possible future extension.

There are two components to the wire length in a mesh: the stub wire length and the grid wire length. Stub wire mismatch directly contributes to skew. To reduce this in uniform meshes, a dense mesh is desirable. This, however, assumes that the sink locations are a random variable when, in practice, they are not. In many designs, sinks are clustered in certain areas due to the many bits of a register or there may be few sinks in another location due to a large memory array. In the case of late design cycle ECOs, it is possible to place the new clock sinks so that they are located near a mesh wire in both the uniform and non-uniform cases.

We propose to minimize the average stub wire length by shifting the vertical and horizontal grid lines. As an example, consider Figure 1 which shows two moves on a vertical and horizontal edge, respectively. Many sinks, such as A and B will connect themselves to the newly located edge rather than a previous edge that is further away. In addition, the overall grid dimensions can be reduced when outer mesh edges are relocated inward toward the internal sinks. This can create some sinks and stubs external to the mesh, but this is easily considered during optimization. In general, this makes the mesh dimensions smaller and the average stub length less. While the difference between the maximum and minimum stub length determines the local skew of a mesh, we find that once the buffering and variation is considered, this component is rather small.

During mesh generation, we define two sets of grid wire locations \( X = \{x_0, \ldots, x_n\} \) for vertical wires and \( Y = \{y_0, \ldots, y_n\} \) for horizontal wires. Our goal is to optimize the values in these sets, i.e. the location of each mesh edge, such that the overall stub length is minimized. This problem is similar, but slightly different, than the k-means clustering problem. The k-means clustering problem is to generate k clusters from a set of points to minimize the total distance of each point to the cluster center. This is a well known NP-hard problem in the Euclidean space, but efficient heuristics are used in other fields to identify natural data clusters [7]. In our approach, however, we want to minimize each clock sinks proximity to the nearest horizontal or vertical grid wire rather than an arbitrary mean location. During the evolution of this paper, we tried several heuristics to optimize the edge location including: iterative method of means and medians (MMM) [1], simulated annealing, independent and simultaneous k-means in the x and y dimensions. We found the simultaneous k-means optimization in the x and y dimensions is able to give very good results with relatively low run-time. It is important that the vertical and horizontal grid wire locations be optimized simultaneously and that each sink be paired with only one horizontal or vertical wire. If this is not done, a significant portion of the flexibility is lost.

The pseudo-code for our non-uniform k-means mesh optimization is shown in Algorithm 1. It is an extension of Lloyd’s well-known iterative algorithm [7]. We start with a uniform grid as in traditional clock meshes and then we iteratively refine the grid locations \( x_i \in X \) and \( y_j \in Y \). Each clock sink, \( s_i \in S \), is assigned to the nearest horizontal or vertical grid wire (\( x_{\text{nearest}} \) or \( y_{\text{nearest}} \)) to form a grid wire cluster. Then, the grid wire (\( x_i \) or \( y_j \)) is moved to the mean location of the cluster. If no sinks are attached to the grid wire, we leave it alone to allow sinks to connect to it in the future. The iterative process terminates when there are no changes in the grid wire locations. Typically this is within a few iterations. While not optimal, the Lloyd-like algorithm is very fast and converges on

![Figure 1: Mesh Uniformity Optimization.](image-url)
4. LP MESH BUFFERING

Both [13] and [9] use a similar greedy set covering heuristic for buffer insertion. In [13], each buffer covers a region that is limited by the output capacitance limit of the buffer. A larger buffer has a bigger region than a smaller buffer. The region itself is determined by the capacitance limit of the buffer and total capacitance of the mesh and sinks in the region. Their heuristic measures the buffer effectiveness (the buffer size over the number of sinks covered) of each buffer at each location. It then iteratively selects the most effective buffer until all sinks are covered. In [9], the authors extended this heuristic to consider spatial locality of the covered sinks and RC attenuation effect of the mesh by encouraging frequent small buffers rather than few large buffers. Both methods, however, are still heuristic and greedy.

The problem with the previous approaches are several fold. First, they do not directly consider resistance. In our implementation of [13], we added a small enhancement that expands the coverage radius in the direction of least resistance rather than purely Euclidean distance. In a non-uniform mesh, this is critical, but for a uniform mesh, it does not matter much since each direction has a uniform resistance. However, the coverage region concept still ignores resistive shielding. Second, neither heuristic includes actual delay. Even in a simple model, the delay depends on the resistance and capacitance from the buffers to the sinks. Ignoring this can result in either large skew, large short-circuit power, and ineffective buffer usage. Last, they do not consider skew (directly) in their formulation. If buffer capacitance limits are properly chosen the results generally meet skew constraints, but the quality of the buffering is highly sensitive to the sizes of the buffer library, the buffer output capacitance load, and the sink capacitance. For example, with a coarse mesh and dense sinks in a region near a grid node, it is possible that putting the largest buffer in each nearby node may not be adequate to cover the region. To find a feasible solution, the library would need to be augmented, the density of sinks decreased or multiple buffers per grid node required.

In this work, we formulate the buffer insertion algorithm based on iterative sizing and deletion of the mesh buffers. Initially, we start with a full coverage of minimum size buffers on all grid nodes and perform continuous sizing. After sizing, this offers the best possible solution for skew, but with very poor power. Using the size information, however, we select a set of small, unnecessary buffers and remove them. Since these buffers are small, the sizing algorithm determined that they were ineffective at improving the skew. We repeat this and terminate the optimization when either the desired number of buffers remain or the skew exceeds a user-specified bound.

4.1 LP Buffer Sizing

The core of our algorithm uses a sensitivity-based linear programming buffer sizing formulation. This is similar to prior works in clock tree synthesis such as [4, 5, 15]. We formulate the LP problem as:

\[
\begin{align*}
\text{min} \quad & \Delta s_{\text{max}} - \Delta s_{\text{min}} \\
\text{s.t.} \quad & s_i - s_{\text{min}} \geq 0, \quad \forall i \in \text{Sinks} \\
& s_{\text{max}} - s_i \geq 0, \quad \forall i \in \text{Sinks} \\
& D + G\Delta = S \\
& P_{\text{cur}} + \beta\Delta \leq P_{\text{max}}
\end{align*}
\]

where \(X\) is a vector of buffer sizes, \(s_i\); \(\Delta\) is a vector of buffer size changes, \(\delta_i\); \(D\) is a vector of present sink delays, \(d_i\); \(G\) is the matrix of the first-order sensitivity, \(\frac{\partial d_i}{\partial x_j}\), of each sink delay, \(d_i\), with respect to each buffer size, \(x_j\); \(S\) is a vector of the new sink delays, \(s_i\); \(s_{\text{max}} (s_{\text{min}})\) is an auxiliary variable for the new slow (fast) sink delay; \(P_{\text{max}}\) is the power limit; \(P_{\text{cur}}\) is the present power consumption; and \(\beta\) is a vector of power sensitivities per buffer size change.

The gradient approximates the sensitivity of each sink delay to a buffer size via a single sided finite difference

\[
\frac{\partial d_i}{\partial x_j} \approx \frac{d_i(x_j + \epsilon) - d_i(x_j)}{\epsilon}
\]

where \(\epsilon\) is the amount of the perturbation and \(d_i(x_j)\) is the sink delay with the given buffer size, \(x_j\).

Given each sensitivity, a linear constraint of the form

\[
s_i = d_i + \sum_{j} \frac{\partial d_i}{\partial x_j} \delta_j,
\]

is created for each sink \(i\) and buffer \(j\). These constraints in clock grids are fairly dense as opposed to clock trees where they are very sparse. All such constraints can be written in matrix form giving us the gradient \(G\), the vector of size changes \(\Delta\), and the present and new sink delays \(D\) and \(S\), respectively in (4).

In general, the linearity of the skew with respect to the gradient does not hold over a wide range especially when many buffers have an impact on the skew as in a grid. In this case, we iteratively apply the LP algorithm while limiting \(\delta\), to an absolute 10% of the minimum buffer size per iteration. Typically, convergence is seen within a few iterations of LP.

4.2 Mesh Buffering through Iterative Deletion

Buffer insertion in an RC mesh is a challenging problem because the buffer locations directly affect the resistive behavior of the mesh. Using modified nodal analysis to calculate the Elmore Delay of a mesh such as in [3], it is not possible to compute the resistance matrix \(R = G^{-1}\) from the admittance matrix \(G\) without at least one buffer because the matrix is linear dependent. Incrementally adding buffers in a greedy method is very sensitive to the first buffer that is chosen. To account for this, we therefore propose an iterative deletion approach to buffer insertion.

The detailed pseudo-code of our algorithm is shown in Algorithm 2. Our algorithm starts by inserting the smallest buffer at each grid node in the mesh. Given these buffers, we perform a minimum skew LP buffer sizing. After this, invariably some buffers remain minimum sized since they have little to no impact on the overall mesh skew. We remove a small percentage of these buffers and repeat until a desired number of buffers exist in the mesh. It is also possible to remove buffers until a skew bound is violated, but we leave that for future work.

We do not want to immediately remove all minimum size buffers, however, because they may be important once other minimum size buffers are removed. In addition, during the initial iterations, there are many minimum sized buffers so we randomly select a subset of these to prevent too many buffers from being removed in a given area. If this happened, skew would be introduced and the algorithm would have poor performance.

It is also important to note that during iterative buffer removal, we only do one iteration of LP instead of many for convergence. This is because we are not concerned with the absolute value of the buffer sizes, but rather the relative sizes to select buffers for removal. If we are concerned with bounded skew, however, we must perform many iterations of LP to find the optimal buffer sizes. After buffer removal is finished, we perform a single accurate sizing with a large iteration limit of the buffers for the final sizes.
Algorithm 2 Mesh Buffering through Iterative Deletion

**Require:** Mesh with set of potential locations \( N \); buffer library \( B \); and desired number of buffers \( k \)

**Ensure:** Buffer Map Set (BM) of \( (i, j) \) for \( i \in N \) to \( j \in B \) (or \( \emptyset \))

1. Set each node to smallest size buffer.
2. For all \( i \in N \) do
   - \( BM(i) \leftarrow 1 \)
3. While \( |BM| > k \) do
   - Perform LP-based buffer sizing iteration
   - Find all minimum size buffers.
   - For all \( i \in \{1..|BM|\} \) do
     - If \( BM(i) == 1 \) then
       - \( M \leftarrow i \)
     - End if
   - End for
   - Add more if not 25% of remaining target in set.
   - While \( |M| < 0.25 \times (|BM| - k) \) do
     - \( M \leftarrow \text{FindNextSmallest}(BM) \)
   - End while
   - Remove first \( 0.25 \times (|BM| - k) \) elements of \( M \) from \( BM \)
   - Perform SLP-based buffer sizing (until convergence)
   - Increase all sizes proportionally until slew target is met.

At this point, it is possible that the skew is good, but slew values are violated. We found that simply scaling the solution while preserving relative buffer sizes preserves the skew value and meets the slew bound with a good power budget. After this, we discretize the buffers to the next highest size in the buffer library. We found that skew in a mesh is especially insensitive to discretization errors.

5. EXPERIMENTAL METHODOLOGY

We implemented our clock mesh synthesis tool in C++ and ran all experiments on CentOS Linux 5.3 system with a 2.6GHz AMD Opteron processor and 8GB of memory. Our delay model uses direct calls to GNU ngspice [8] exclusively for utmost accuracy. All benchmarks are able to complete in at most a few hours.

In our experiments, we use the technology data from the 2009 ISPD Clock Synthesis Contest [11], but we adapted the benchmarks used in [9, 13] to the 45nm technology node in a similar manner as the ISPD benchmarks. Clock sinks are assumed to be minimum size buffers in the technology with a 4.2fF capacitance. We require that clock sinks have maximum slew rates of 100ps without exception. We use 12 buffer sizes ranging from \( 2 - 24 \times \) with load limits based on a buffer electrical effort of 4. All results use this buffer library and not continuous sizes.

A direct comparison with [9] was not possible since we were unable to obtain clarifications on the methodology and algorithms from the authors. However, our implementation obtains very similar results when using the hand-sized meshes. We found, however, that we do not need such fine grids (> \( 12 \times 12 \)) as they used to achieve reasonable skews even with our baseline implementation of their algorithm. We instead opt to use the minimum wire length uniform grid to determine the size. All algorithms use the same grid sizes in all cases.

To verify the robustness of the solutions, we ran Monte Carlo simulations in ngspice [8] with ±50ps 1-sigma variation of buffer input delays and and ±5% 1-sigma variation of wire widths, buffer gate lengths, and sink load capacitance. This is similar to the parameters used in both [13] and [9].

Rather than approximate power using the total buffer and mesh capacitance, we use the integral of current over a full clock period to calculate the total charge consumed. This includes short-circuit power and leakage in addition to dynamic power in our power measurements.

6. EXPERIMENTAL DATA

6.1 Grid Optimization

In our first set of experiments, we compare the uniform and non-uniform meshes using the baseline set covering buffer insertion as done in [13]. An example mesh on benchmark s3r1 from the ISPD contest is shown in Figure 2. The mesh wires have shifted from the upper right area without any clock sinks to denser areas to reduce many stub lengths. The overall dimensions of the grid are significantly less due to the shifting of the outer mesh wires as well. In Figure 3, we plot the total wire length, grid wire length, and stub wire length for various mesh sizes on the benchmark using uniform and non-uniform meshes. It is apparent that for coarse meshes, the non-uniform approach does much better, but as the mesh gets finer they asymptotically approach the same result. This is because the non-uniform meshes can only reduce the stub lengths. In the very fine meshes, the stub wire length reduction becomes insignificant compared to the total mesh wire length. Therefore, shifting mesh edges does not have an impact on the total wire length. In the region of the minimum capacitance mesh size, we frequently observe 15% or more wire length reduction. This is often enough power reduction to increase the mesh size by one or two values in each dimension to improve robustness. However, for our experi-
ments, we will fix the mesh size to be equal.

The maximum and minimum stub lengths can contribute directly to skew. Given our non-uniform mesh, it is possible that the maximum stub length will increase, but we do not see this in practice. Even when the maximum stub length increases, the clock skew will be extremely dependent on the buffer locations in addition to the mesh itself. It should be noticed, as seen in Figure 2, the maximum stub length is decreased and therefore clock mesh skew decreases. Non-uniform meshes also reduce the maximum delays within the mesh. Regions where clock sinks are clustered present higher switching delays and, therefore, more skew. Since non-uniform meshes reduce stub wire lengths by moving edges to the center of the clusters and more effectively utilize mesh wires, the maximum delay to those regions decreases, reducing the total clock skew.

To certify that the non-uniform mesh presents a clock skew equivalent to an uniform mesh, we compared the results of both when mesh buffers are inserted according to the methodology in [13]. This experimental data is reported in Table 1. We observed an average of 7.6% total wire length reduction which translated to a 10.5% power reduction. This additional power reduction is primarily due to the different grid nodes in the non-uniform mesh compared to the uniform mesh. Since the grid nodes are potential buffer locations, they will typically be concentrated near higher density sink regions and allow more efficient buffering. We also observed an 11ps mean skew reduction when considering variation, so the robustness is increased on average.

6.2 Buffer Insertion

Even though our approach assumes a continuous buffer size during sizing, we discretize to the continuous library at the end by upsizing each buffer to the nearest library size. We also maintained the same 100ps slew constraints in all benchmarks.

Figure 4 shows a benchmark with, 4(b), and without, 4(a), our LP buffer insertion. For the smaller benchmarks such as s5378, we see that buffers are typically added to the perimeter and a few select center locations. We do not see buffers added to adjacent grid nodes very often. Whereas in the greedy method, there is no consideration of previous buffer locations since it only considers which sinks are covered during set covering. This allows buffers to be inserted adjacent which may be non-productive when variations are introduced.

We typically see buffer sizes of $4 \times 8$ in our result even after slew correction whereas [13] produced larger sizes and more of them. This is due to the restrictive capacitance limit and the discontinuity of the output slews have a broad range from about 75-100ps. Our result, however, has an electrical effort closer to 16, but with a tighter slew range of 90-100ps. Increasing the capacitance limit of the greedy method results in slew violations. In the 45nm PTM technology, we confirmed that an inverter with a 100ps input slew can drive a $16 \times$ load with an output slew of approximately 60ps. This confirms that the short-circuit currents in the greedy buffer method have a counterproductive effect even in the nominal case due to different buffer sizes, sink loads, and delays through the mesh.

6.3 CPU Time

Since we use ngspice for all delay models our run-time is relatively long (a few hours) compared to the set covering approach (less than a minute). Excluding calls to ngspice, however, the slowest benchmark (the non-uniform grid with LP sizing on s38584) takes approximately 11 minutes. For a high-quality design, our total run-time is perfectly acceptable, however. If desired, we can reduce our run-times by using a simplified Elmore Delay mesh analysis or integrating calls to circuit analysis to avoid file I/O.

7. CONCLUSIONS
In this paper, we presented two algorithms: one for non-uniform clock mesh generation and one for LP buffer insertion. Both methods show significant improvement over prior algorithms. Our non-uniform clock mesh can achieve a 10.5% power reduction due to decreased wire and indirect buffer savings. The non-uniform mesh was shown to be able to reduce clock mesh capacitance especially for coarse meshes. It should also be noted that this technique is complimentary to the mesh reduction techniques presented in [9, 13] that present a greater improvement for finer meshes. Our LP buffer insertion shows a 41% power reduction over the previous greedy set covering algorithm in addition to an 27ps mean skew reduction when variation is considered.

8. REFERENCES