Timing Closure in Chip Design

Dissertation
zur Erlangung des Doktorgrades
der Mathematisch-Naturwissenschaftlichen Fakultät
der Rheinischen Friedrich-Wilhelms-Universität Bonn

vorgelegt von

Stephan Held

aus Bad Harzburg

im Juni 2008

Erstgutachter:	Professor Dr. Bernhard Korte
Zweitgutachter:	Professor Dr. Jens Vygen

Tag der Promotion: 28.08.2008

Danksagung/Acknowledgments

I would like to express my gratitude to my supervisors Professor Dr. Bernhard Korte and Professor Dr. Jens Vygen. Without their ideas, help, guidance, and backing this thesis would not have been possible. Under their leading, the Research Institute for Discrete Mathematics at the University of Bonn provides outstanding working conditions.

I would further like to thank my past and present colleagues at the institute, especially Christoph Bartoschek, Jens Maßberg, Professor Dr. Dieter Rautenbach, Dr. Christian Szegedy, and Dr. Jürgen Werber. Working together in the various fields of timing optimization and clocktree design was very inspiring and joyful.

A special thanks goes to Alexander Kleff and Rüdiger Schmedding for their contributions to clock skew scheduling and time-cost tradeoff curve computations as well as to all other collaborating students in the VLSI team.

Sincere thanks go to Dr. Ulrich Brenner and Markus Struzyna for their efforts to modularize BonnPlace, providing the capability to use it in an integrated design flow.

I am very grateful to all people at IBM who shared their knowledge of VLSI design with me and helped to integrate, promote and support our design flow, especially Dr. William E. Dougherty, Günther Hutzl, Dr. Jürgen Koehl, Karsten Muuss, Dr. Matthias Ringe, and Alexander J. Suess.

Further thanks go to Koen Van Eijk and Pankaj Goswami from Magma Design Automation for their comprehensive experiments and helpful suggestions, while integrating the fast gate sizing code into the Magma design environment.

I am personally grateful to Maggie and David for reading parts of this thesis at short notice towards the end of completion.

My special thanks goes to my parents Gabriele and Wilhelm Held, and the rest of my family for all their support, exhortations, and repeated proofreading of this work.

But my biggest thanks goes to Sandra for her loving encouragements and patience during the whole time I was working towards the completion of this thesis.

Contents

1 Introduction

Chip design is one of the most fascinating areas of mathematical optimization and of combinatorial optimization in particular. A central characteristic of a computer chip is the speed at which it processes data, determined by the time it takes electrical signals to travel through the chip. A major challenge in the design of a chip is to achieve *timing closure*, that is to find a physical realization fulfilling the speed specifications. Due to the rapid development of technology with ever shrinking feature sizes, highly efficient and effective algorithms are essential to coping with deep submicron effects as well as with the rising complexity of computer chips. In this thesis, we develop several new algorithms for optimizing the performance of computer chips. These algorithms are combined into a common program flow to achieve timing closure in physical design.

In Chapter 2, we present the timing closure problem with a special focus on timing constraints. Apart from timing, further objectives such as power consumption and robustness also have to be considered.

One of the main subproblems is the construction of repeater trees that distribute electrical signals from a source to a set of sinks. In Chapter 3, we present a new algorithm for generating repeater tree topologies: First, we propose a new delay model for estimating the performance of such a topology. In contrast to known approaches, it accounts not only for the distance a signal has to cover but also for branchings in the topology that introduce extra delay. It turns out that the basic structures for the extreme optimization goals, (resource-unaware) performance and (performance-unaware) resource efficiency are optimum binary code trees with shortest path lengths and minimum Steiner trees. Our algorithm scales seamlessly between these two optimization goals. Moreover, its extreme speed is also very important, as several 10 million instances have to be solved within our timing closure flow.

Another indispensable optimization step is the circuit or transistor sizing. A chip is composed of millions of small circuits that implement elementary logic functions. A small set of predefined physical layouts is available for each circuit. The alternative layouts differ in the size of the underlying transistors, and thus in their speed and power consumption. Now, circuits have to be mapped to a layout of adequate size such that not only are timing constraints met, but total power consumption is also minimized. The problem is that discrete decisions are accompanied with nonlinear non-convex constraints. Instance sizes of several million circuits make exact optimization algorithms unusable. Former approaches usually work with simplified delay models, for which the problem can be solved optimally, but that can show substantial deviations from the actual delay rules.

In Chapter 4, we propose a new circuit sizing algorithm that is not restricted to certain delay models. Instead of hiding signal shapes, as most existing approaches do, they play a central role in our extremely fast algorithm. Furthermore, we develop a new method to compute lower bounds for the delay of the most critical path and we show the effectiveness of our algorithm by comparing the results with these bounds. On average, the critical path delays are within 2% of the lower delay bounds, and thus, are very close to the optimum.

Next, in Chapter 5, we consider the optimization of the clock skew schedule, which is the assignment of individual switching windows to each register such that the performance and robustness of the chip are optimized. We present the first strongly polynomial time algorithm for the cycle time minimization in the presence of multiple clock phases and multi-cycle phases. While existing approaches are based on binary search, we show that the problem reduces to a minimum ratio cycle problem and can be solved by an adaptation of Megiddo's algorithm.

Furthermore, we mathematically analyze the convergence of an iterative method that greedily schedules each register to its local optimum. It has recently been shown that this method maximizes the worst slack. We extend this result by the possibility of restricting the feasible schedule of a register to some time interval. Such constraints are used in practice to indirectly limit the power consumption of the clocktrees. The iterative method has many advantages over classical combinatorial algorithms, because it is very memory-efficient, fully incremental, and, at least on chip instances, surprisingly fast—although theoretically, it has a linear convergence rate with a convergence factor arbitrarily close to one. The iterative algorithm does not guarantee optimality for the slack distribution above the worst slack. We overcome this deficiency by introducing a hybrid model that selectively introduces global constraint edges to hide the most critical paths, which have already been optimized. The chapter closes with a description of how a clocktree can be constructed that realizes a given clock schedule.

Chapter 6 deals with the computation of linear time-cost tradeoff curves in graphs with variable edge delays and costs depending on the selected delays. Traditionally, this problem was restricted to acyclic graphs, where the time-cost tradeoff curve can be computed by successive maximum flow and longest path computations. We develop the first combinatorial algorithm for the general case with cycles in the graph. It alternates minimum cost flow computations, which determine a steepest descent direction, and minimum ratio cycle computations, which determine the maximum feasible step length. At the end of the chapter, we show how certain optimization problems, such as threshold voltage optimization or plane assignment, with a variable clock skew can be modeled as a discrete time-cost tradeoff problem. Here, linear relaxation serves as an effective heuristic for the discrete problem.

Finally, in Chapter 7, all the presented algorithms are combined into a timing closure flow. It alternates placement and timing optimization, where the placement is steered by netweights that penalize the length of critical nets. This phase is followed by a timing refinement and placement legalization step. In the end, the clock schedule is optimized for the last time, and clocktrees are inserted based on this schedule.

As part of this dissertation, we have implemented the proposed algorithms and the integration into a design flow with the support of colleagues and students. They are known as the Bonn Fast Timing Driven Loop and Bonn Clock Optimization and are part of the BonnTools, a collection of physical chip design programs, developed at the Research Institute for Discrete Mathematics at the University of Bonn, and part of an industrial cooperation with IBM and in recent years also with Magma Design Automation. As part of this cooperation, we have helped engineers all over the world employ these tools in the design of many of the most complex industrial chips, which can now be found in a number of devices, including network switches and mainframe servers.

2 Timing Closure

2.1 Integrated Circuit Design

Digital integrated circuits, also known as computer chips, represent finite state machines that process digital signals. Based on input signals and the current state, a new state as well as output signals are computed. The actual computations, which are the state transitions, are done in the *combinatorial logic*, while the states are stored in *registers*. Figure 2.1 shows a schematic of a computer chip. Starting

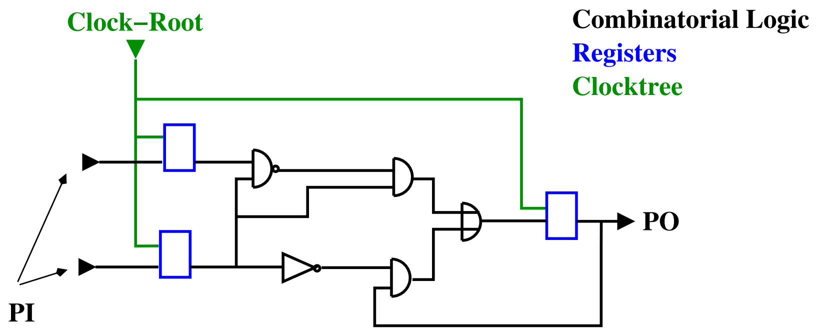


Figure 2.1: Schematic of a simplified chip.

from *primary inputs* (PI) and register outputs, Boolean variables, represented by electrical signals, propagate through the combinatorial logic until they reach *primary outputs* (PO), or register inputs, where they are stored until the next computation cycle starts. The registers are opened and closed once per cycle by a periodic *clock signal*. The clock signal is distributed by a *clock network*, which is often realized by a *clocktree*. A clocktree can be considered as a huge net with tree topology, into which repeaters (inverters and buffers) are inserted to refresh the signals, and that is constructed such that each source-sink path achieves a prescribed delay target. A higher clock frequency yields a faster chip, provided that the combinatorial logic is fixed, and not subdivided by registers into several *pipeline stages*. The frequency is limited by the delay through the combinatorial logic.

Computer chips consist of billions of electrical devices, mostly transistors. The process of creating chips with such a high device density is called very large scale integration—*VLSI*. Thus, today's computer chips are also called *VLSI chips*, and the design of their layout is called *VLSI design*.

VLSI chips are composed of a finite number of layers or planes, which are manufactured one by one in a bottom up fashion by etching and metalization. The lowest planes contain the transistors, while the upper planes are reserved for wires. The wire planes usually contain only wires in either x-, y-direction or vias between adjacent planes in x/y-direction.

2.1.1 VLSI Design Flow Overview

Due to the huge amount of interacting structures that must be layouted, VLSI design is a very complex task. Therefore, it is typically split into two main phases and several subphases as shown in Figure 2.2. First, in the *logic design phase* a correct

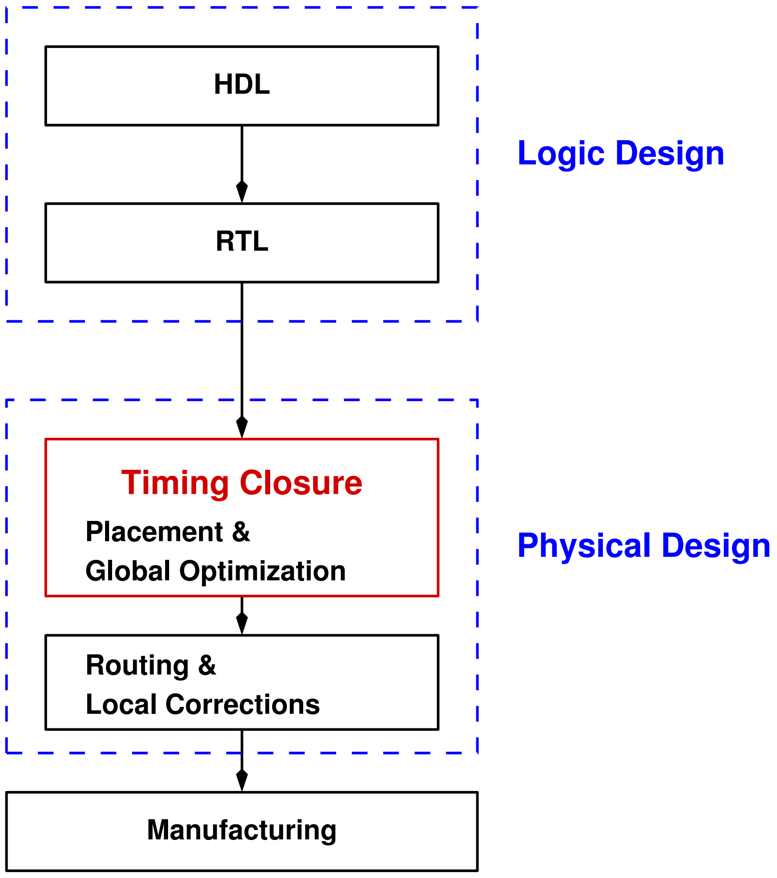


Figure 2.2: VLSI design flow

logical description of the final application has to be found. Such a description is usually made in a hardware description language (HDL), which is similar to computer programming languages, such as C/C++. Then the logical description is translated by a compiler into the *register transfer level* (RTL) description, where registers and the combinatorial logic in terms of elementary logic circuits such as *INVERTER*s or two bit *AND*s are determined. Most circuits represent a Boolean function with only a single output value. These circuits with a single output are also called *logic gates* or just *gates*.

Second, in the *physical design phase* the RTL description has to be implemented physically. The circuits have to be placed disjointly in the chip area, their sizes have to be chosen, interconnects are sped-up by repeaters, and routed disjointly within a small set of routing planes. These steps cannot be performed independently. On the one hand, useful circuits sizes and repeater trees can only be inserted once placement and some routing information is known. On the other hand, placement and routing must prioritize timing critical paths which are only known after sizing and buffering. Detailed routing is a very runtime-intensive task. Therefore, it is only possible to perform rather local corrections instead of global optimizations in interaction with detailed routing.

In practice chip design is not performed strictly according to that one-way diagram in Figure 2.2. There are all kind of feedback loops between those blocks. However, they will be iterated seldom compared to subprograms that are performed within in each box. In this thesis we consider the design steps before detailed routing with a focus on signal performance optimization. As this basically means to fulfill all timing constraints, this phase is called *timing closure*.

2.1.2 Decomposition of VLSI Designs

Physical laws and the lithographic production process of the hardware imply many constraints like shape and spacing rules for wires and transistors. An analysis of the lithographic realizability of a physical layout requires extensive computations. This holds also for the analysis of the electrical signal propagation through the chip. To reduce design complexity large circuits are composed of smaller circuits, which can be reused many times without being reinvented and checked every time.

In the flat design style, a chip is decomposed into multi-million circuits, which are mapped to *circuit definitions* or *books* from a predesigned *circuit library*. Most books represent elementary logic functions like an *INVERTER* or a two-bit *NAND* function. Other books can be large like memory blocks or even micro processors. Such elements are often called (*hard*) *macro circuits*.

In the hierarchical design style the chip is decomposed into a few sub-chips, called random logic modules (RLM). Each of them is designed as a separate chip instance. Figure 2.3 shows the placement of the chip David that is composed of more than 4 million circuits in the toplevel and another million in a hierarchy RLM in the bottom left (with orange background).

Hierarchy is rather a practical issue but does not change the theoretical aspects. Throughout this thesis, we consider only flat designs, which covers also the design of individual flat levels in an hierarchical design style.

2.2 Physical Design Input

The geometric information for an object x that occurs on a chip is given as a set $\mathcal{S}(x)$ of *shapes*. A shape is an axis-parallel rectangle, which is assigned to a

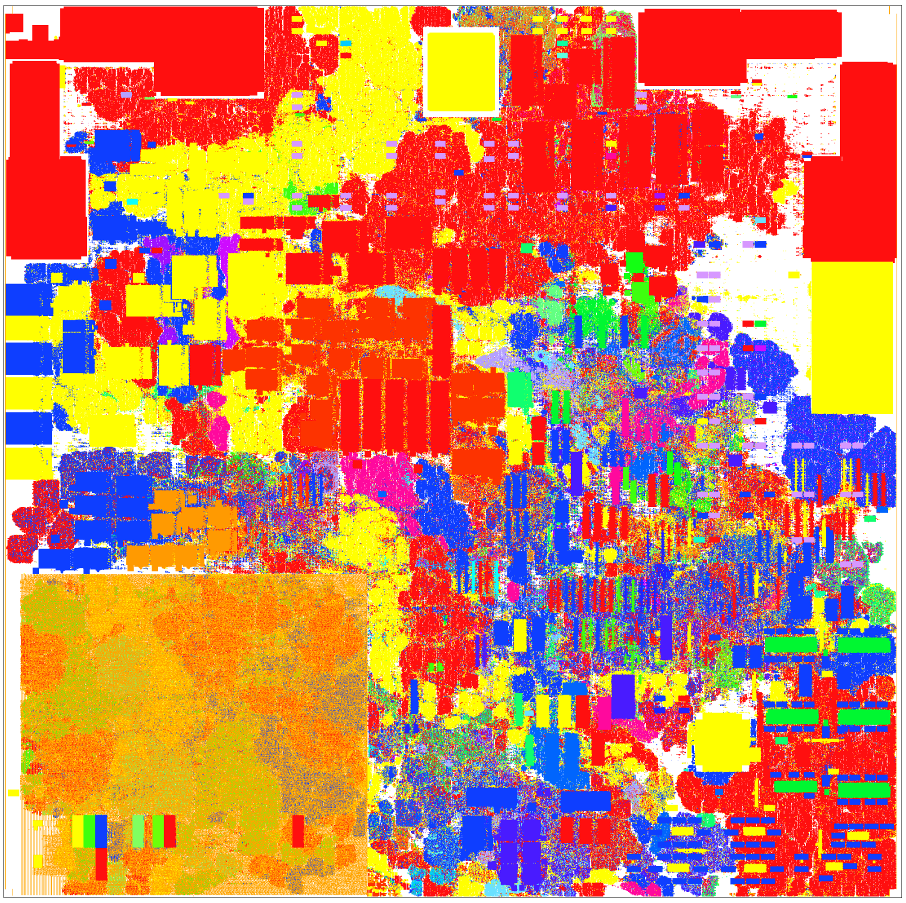


Figure 2.3: Placement of a chip

plane. Formally a shape is a set $[x_1, x_2] \times [y_1, y_2] \times \{z\}$, with rectangular coordinates $x_1, x_2, y_1, y_2, z \in \mathbb{N}$, $x_1 < x_2$, $y_1 < y_2$, and a plane z. A computer chip is constructed within a *chip image*. An image $\mathcal{I} = (\mathcal{S}(\mathcal{I}), \mathcal{D}(\mathcal{I}), P(\mathcal{I}))$ consists of a boundary shape set $\mathcal{S}(\mathcal{I})$ defined as

$$\mathcal{S}(\mathcal{I}) := \Big\{[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}] \times z \mid 0 \leq z \leq z_{max}, \Big\},$$

where a base rectangle $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$, $x_{\min}, x_{\max}, y_{\min}, y_{\max} \in \mathbb{N}$, $x_{\min} < x_{\max}$; $y_{\min} < y_{\max}$ is distributed over a given number $(z_{\max} + 1)$, $z_{\max} \in \mathbb{N}$ of planes. The planes with numbers $1, 2, \ldots, z_{\max}$ are used for wires whereas plane '0' is the placement plane. $\mathcal{D}(\mathcal{I})$ is a set of shapes representing blockages in either of the placement and routing planes. Furthermore, $\mathcal{I}$ contains a set $P(\mathcal{I})$ of *IO-ports* . The layout of each port $p \in P(\mathcal{I})$ is given by a shape set $\mathcal{S}(p) \subset \mathcal{S}(\mathcal{I})$.

A *circuit library* $\mathcal{B}$ defines a set of prelayouted circuits that can be instantiated on the chip. Each *book* $B \in \mathcal{B}$ has a physical description by a triple $(\mathcal{S}(B), \mathcal{P}_{in}(B), \mathcal{P}_{out}(B))$. $\mathcal{S}(B)$ is a *set of shapes* and $\mathcal{P}_{in}(B), \mathcal{P}_{out}(B)$ are the input and output pin definitions of B, which have their own *shape sets* $\mathcal{S}(\mathcal{P}_{in}(B))$ and

$\mathcal{S}(\mathcal{P}_{out}(B))$. Figure 4.1 on page 62 shows an example of three different books implementing an *INVERTER*.

The *netlist* of a chip consists of a quadruple $(\mathcal{C}, P, \gamma, \mathcal{N})$. $\mathcal{C}$ is a finite set of *circuits*. P is a finite set of pins. $\mathcal{N}$ is a finite set of *nets* that connect certain pins. Formally $\mathcal{N}$ is a partition of the set of pins. Pins are mapped to circuits or to the image in case of IO-ports by the mapping $\gamma : P \to \mathcal{C} \dot{\cup} \mathcal{I}$. A mapping $\beta : \mathcal{C} \to \mathcal{B}$ binds every circuit to a book from the circuit library.

The instance for physical design consists of a netlist $(\mathcal{C}, P, \gamma, \mathcal{N})$, an image $\mathcal{I}$, and a circuit library $\mathcal{B}$ with an initial binding β. By $P(o), o \in \mathcal{B} \cup \mathcal{C} \cup \mathcal{N} \cup \mathcal{I}$ we denote the pins associated with the object o. Furthermore $P_{in}(c)$ denotes the set of *input pins* and $P_{out}(c)$ denotes the set of *output pins* of a circuit or book $c \in \mathcal{C} \cup \mathcal{B}$. For a net $N \in \mathcal{N}$, $P_{in}(N)$ denotes the source pin, while $P_{out}(N)$ is the set of sinks. A source pin is either an output pin of a circuit of a primary input pin of the chip, while a sink is either an input pin of circuit or a primary output pin.

2.3 Design Constraints

The main focus in this thesis is to meet the timing constraints which are described in detail in Section 2.4. As further side constraints we summarize those that are most important for prerouting timing closure:

2.3.1 Boolean Equivalency

In general it is allowed to replace the input netlist by any Boolean equivalent netlist. But the Boolean equivalency of input and output has to be verified finally before the chip goes into production. The decision problem whether two netlists are Boolean equivalent is a *NP*-hard, as it contains the 3-SAT problem. Therefore, we consider only modifications of limited complexity, whose correctness is verifiable sufficiently fast.

Complex modifications such as latch insertion or re-timing, which is the swapping of registers with logic books to balance the length of data paths, are considered only at an early design stage (but not in this thesis). They require an extensive simulation of many execution cycles.

In this thesis the main focus is given to the replacement of the physical realization $\beta(c)$ of a circuit $c \in \mathcal{C}$ by a book from the class $[\beta(c)]$ of logically equivalent books, and the replacement of repeater trees by equivalent trees. In special applications the cloning of circuits, the merging of equivalent circuits, or the swapping of equivalent input pins of a circuit will be applied too.

2.3.2 Placement Constraints

The circuits $c \in \mathcal{C}$ and thereby transistors must be placed disjointly in the chip area. Formally, we have to find a location $\mathrm{Pl} : \mathcal{C} \to \mathbb{R}^2$ such that

$$(\mathcal{S}(\beta(c)) + \mathrm{Pl}(c)) \subset [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$$

and $(\mathcal{S}(\beta(c))+\mathrm{Pl}(c)) \cap (\mathcal{S}(\beta(c'))+\mathrm{Pl}(c'))$ is at most one-dimensional for two different circuits $c, c' \in \mathcal{C}$. Here, the sum of a shape S and a location $(x', y') \in \mathbb{R}^2$ denotes the translation $S + (x', y') := \{(x + x', y + y', z) \mid (x, y, z) \in S\}$. Furthermore, some part of the chip area might be blocked to reserve space for later circuit insertion or to model non-rectangular placement areas. We assume that blockages are modeled as dummy circuits of fixed size and location.

2.3.3 Routing Constraints

As we consider only prerouting optimization, we will not address routing constraints in detail. However, we have to ensure the routability of our output, which is measured by global routing or routing estimation algorithms, see Brenner and Rohe [2003], Vygen [2004], and Müller [2006]. Such an estimation is performed in a coarsened global routing graph $G^{\mathcal{GR}}$. Here, the chip is partitioned into regions by an axis-parallel grid. For each region and wiring plane there is a vertex in $V(G^{\mathcal{GR}})$. Adjacent vertices are joined by an edge, with a capacity value indicating how many wires of unit width can join the two regions. For each net we summarize its pins within each region. If a pin consists of several shapes located in different regions, the pin is assigned to one of those regions arbitrarily. Now, instead of connecting the individual pins, these regions respectively vertices of have to be connected by a Steiner tree in the global routing graph. A chip is considered routable if a packing of the Steiner trees of all nets that respects the edge capacities in the global routing graph can be found.

2.4 Timing Constraints

Timing analysis is the analysis of signals through the chip. A comprehensive introduction into VLSI timing analysis is given by Sapatnekar [2004]. The static timing analysis which we apply here was described first by Hitchcock et al. [1982]. We now summarize the basic concepts.

The logical state of a point on a chip is given by the electrical voltage. High voltage V_{dd} represents the **true** or **1** value, low voltage V_0 represents the **false** or **0** value. A *signal* is defined as the voltage change over time (Figure 2.4). We distinguish *data signals* and periodic *clock-signals.* Data signals represent the bits of a logical computation. They start at primary inputs or register outputs, propagate through the combinatorial logic and enter primary outputs and register inputs again. Clock signals control the storage elements on the chip. They are generated by analog oscillator devices combined with a *phase-locked loop* (PLL) and distributed from a *clock root* to the registers by a *clock distribution network* which is often realized by a *clocktree* (see Figure 2.1 on page 5).

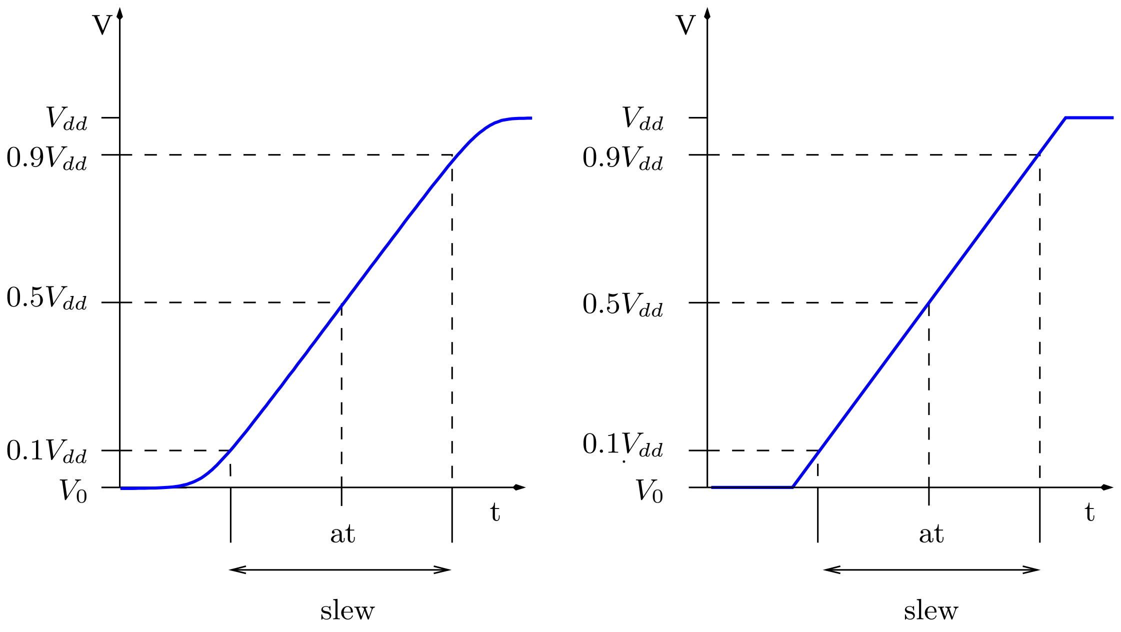


Figure 2.4: A (rising) signal on the left and its approximation on the right

Because of manufacturing uncertainties the signal propagation through an electrical device or wire segment can only be estimated. Even assuming certainty the delay-computation through a series of transistors requires the solution of non-linear differential equations. For our purpose, and in practice, for each book a delay-function is given as a black box function that approximates the result of the underlying differential equations.

In addition, an exact analysis of the dynamic switching behavior requires simulations with an exponential number of input patterns. Therefore, in *static timing analysis*, worst case assumptions for the switching patterns at every single circuit are made. For instance, for a two-bit *NAND* with input pins A and B, the delay from A to the output depends also on the present voltage in B. The delay from an input pin to an output pin of a circuit is estimated assuming a worst case scenario of input signals at the other input pins.

2.4.1 Static Timing

In static timing analysis signal occurrences are computed at certain *measurement points*, which are usually the pins in the netlist. Some measurement points can also be internal pins of circuits, which are not part of the netlist.

Signals are estimated by two linear functions that represent the earliest possible and latest possible occurrence of the signal. This defines two *timing modes* which are named *early* and *late*. Each linear function is given by two values: the *arrival time* (at) and the *slew* (slew). Usually the arrival time is defined as the 50% voltage change and the slew is defined as the 10% and 90% V_{dd} transition time for a rising signal and vice versa for a falling signal (see Figure 2.4). In general the slew is defined by the range in which the real signal is almost linear. In industry other ranges like 20%–80% or even 40%–60% are used rarely.

2.4.2 Circuit Delays

For every book $B \in \mathcal{B}$ a *model timing graph* $G_{\mathcal{T}_B}$ represents possible signal propagations from input toward output pins. The vertex set $V(G_{\mathcal{T}_B})$ contains at least one vertex for every pin from $\mathcal{P}_{in}(B)$. Complex books have internal vertices, that need not correspond to an existing pin. Directed edges in $E(G_{\mathcal{T}_B})$ represent signal relations between the vertices in $V(G_{\mathcal{T}_B})$. They are called *propagation segments*. All paths in $G_{\mathcal{T}_B}$ are directed from input to outputs. Every propagation segment is labeled by a triple $(\eta, \zeta, \zeta') \in \{early, late\} \times \{rise, fall\} \times \{rise, fall\}$. The label determines for some timing mode $\eta \in \{early, late\}$ a possible signal transition based on the underlying logic function. The signal is inverted if $\zeta' \neq \zeta$ and non-inverted otherwise. For instance, the propagation segments through an *INVERTER* or *NAND* are inverting, while segments through an *AND* or *OR* are non-inverting, and an *XOR* has both types of segments. A special case are registers where the output data signals are triggered by the opening clock input signal. Therefore, there are so called *triggering segments* of type $(\eta, \zeta, \text{rise})$ and $(\eta, \zeta, \text{fall})$, where ζ is the opening edge of the register. For elementary books $G_{\mathcal{T}_B}$ is a complete bipartite graph on $V(G_{\mathcal{T}_B}) = \mathcal{P}_{in}(B) \,\dot\cup\, \mathcal{P}_{out}(B)$, directed from input pins to output pins. For more complex books—like registers or large macros—$V(G_{\mathcal{T}_b})$ contains internal pins and edges, mostly for the purpose of graph size reduction. Figure 2.5 shows an example for a latch model graph of a simple latch. Propagation segments are colored blue. The red edge between the data input d and the clock input c refers to arrival time constraints that will be introduced later in Section 2.4.6. Parallel edges are drawn once.

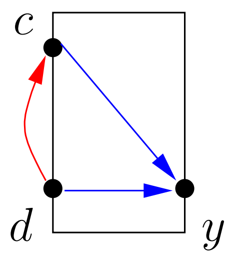


Figure 2.5: A simple latch and its model graph.

The delay and slew transformation through a propagation segment $(v, w) \in E(G_{\mathcal{T}_b})$ depend on the *input slew* of the signal at the tail v and the *downstream capacitance* $\text{downcap}(w, \eta)$, which is also called *load capacitance*, at the head w (see Figure 2.6). The downstream capacitance at w is the sum of the wire capacitance $\text{wirecap}(N, \eta)$

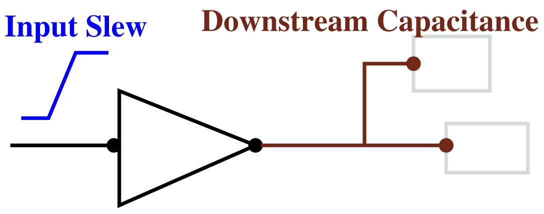


Figure 2.6: Circuit delay and slew function parameters.

and the sum of sink pin capacitances $\sum_{u \in N \setminus \{w\}} \text{pincap}(u, \eta)$ of the net $N \in \mathcal{N}$ containing w.

The capacitance values depend on the timing mode η to account for on-chip variations, such as varying metal thickness, small temperature or voltage changes. Variations across different chips and different operating conditions are usually much higher. Enhanced timing analysis methods, taking into account the full spectrum of variation, are applied only after routing. As they are accompanied with large running times they allow only a small number of physical design changes and are not suitable for (prerouting) timing closure.

A delay function $\tilde{\vartheta}_e : \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} \to \mathbb{R}$ and a slew function $\tilde{\lambda}_e : \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ provide all information for the signal propagation through a propagation segment. The first parameter is the downstream capacitance and the second the input slew. If w is an internal node the load capacitance can already defined by the internal structure of the book B. In such cases $\tilde{\vartheta}_e$ and $\tilde{\lambda}_e$ can be considered as constant in the load capacitance parameter.

The rules return reliable information only if the input slew and load capacitances are within some intervals $[0, \text{slewlim}(v, \zeta)]$ and $[0, \text{caplim}(w)]$ with $\text{slewlim}(v, \zeta)$, $\text{caplim}(w) \in \mathbb{R}_+$. We assume that the functions are extrapolated continuously and monotonically on $\mathbb{R}_{\geq 0}$. Besides the upper *slew limit* $\text{slewlim}(v, \zeta)$ and the upper *capacitance limit* $\text{caplim}(w)$ the timing rules are often only valid if slews and capacitances are greater than a lower limit, which is a very small positive number. The lower limits usually cannot be violated unless an output pin is not connected to a net or a physically impossible input slew is asserted by the designer. Therefore, lower limits are ignored throughout this thesis.

The timing functions are monotonically increasing Lipschitz continuous functions. Figure 2.7 shows an example of a delay function. Slew functions have similar shapes as delay functions.

2.4.3 Wire Delays

In contrast to circuit delays, where we are given precharacterized delay and slew functions for all $B \in \mathcal{B}$, wire delays have to be computed on arbitrary topologies. As it is usually modeled as the delay through a discrete electrical network consisting of resistance and capacitance elements, the wire delay is often called *RC-delay*.

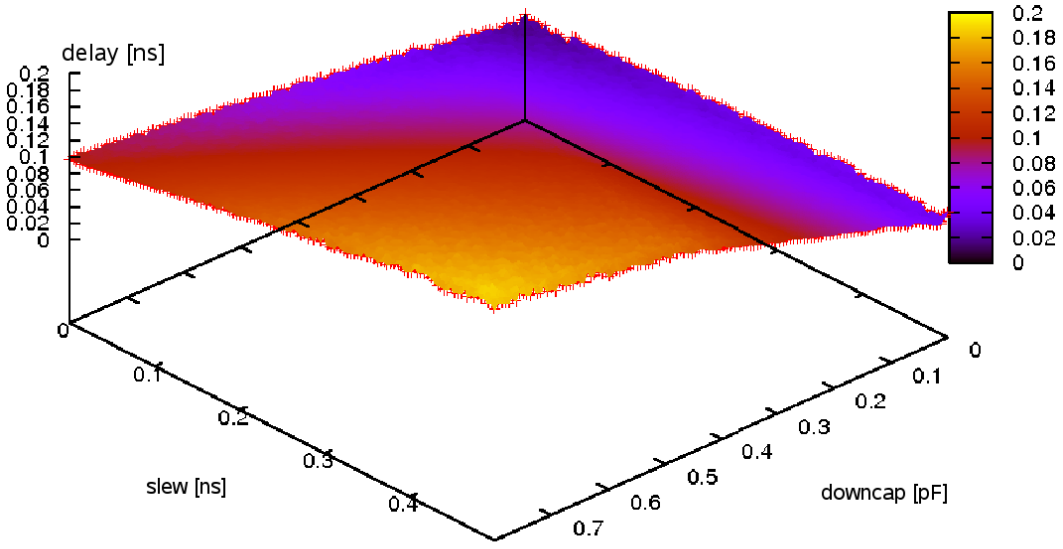


Figure 2.7: Example of a delay function. On the left the input parameters slew and downstream capacitance are shown. The graph on the right side shows a typical delay function. The x/y-axis are labeled by the input parameters slew and capacitance, while the z-axis shows the resulting delay.

Fortunately wire delays are easier to compute than circuit delays because the underlying differential equations are linear. For the purpose of timing closure we mostly apply an even simpler approximation proposed by Elmore [1948]. Given a Steiner tree Y, let $Y_{[pq]}$ be the edge set of the unique oriented path from a source p to a sink q. The (preliminary) RC-delay $rc_{\text{Elmore}}(p, q, \eta)$ from p to q in a timing mode $\eta \in \{early, late\}$ is defined as

$$rc_{\text{Elmore}}(p, q, \eta) = \sum_{e=(x,y)\in Y_{[p,q]}} res(e, \eta)\left(\frac{cap(e, \eta)}{2} + \text{downcap}(y, \eta)\right) \tag{2.1}$$

were $res(e, \eta)$ and $cap(e, \eta)$ are the wire resistance and capacitance estimates of the wire segment e, while downcap(y, η) is the total metal capacitance of all wire segments and input pins, which are reachable from y (assuming the tree being oriented from a root to the leaves with root p). The resistances and capacitances depend on wire thickness, width, layer (distance from ground) and neighboring wires.

Delay and slew functions on the p-q-connection are defined based on the RC-delay and the input slew s at p:

$$\vartheta_{\text{Elmore}}(p, q, s, \eta) = \vartheta_{\text{Elmore}}(s) \cdot rc_{\text{Elmore}}(p, q, \eta) \tag{2.2}$$
$$\lambda_{\text{Elmore}}(p, q, s, \eta) = s + \lambda_{\text{Elmore}}(s) \cdot rc_{\text{Elmore}}(p, q, \eta) \tag{2.3}$$

The final net topologies and neighboring wires are not known throughout the timing closure phase. For the majority of nets we make the pessimistic assumption that all its wire segments are affected by a maximum possible coupling capacitance. Later some of the algorithms will reserve free neighboring channels for timing critical nets to obtain faster and more predictable delays. Note that the Elmore delay is an upper bound on the wire delay with the nice property that it is invariant under subdividing a Steiner tree segment.

Especially those algorithms that are designed for fine-tuning do also work with more accurate delay models such as SPICE (Nagel and Pederson [1973]), or RICE (Ratzlaff et al. [1991]).

As the wire topologies are determined only after the timing closure phase, we estimate the final topology by short Steiner trees neglecting disjointness between the nets. The rectilinear minimum Steiner tree problem is known to be *NP*-hard (Garey and Johnson [1977]). For small trees with up to nine terminals exact solutions can be computed efficiently via table look-up (Chu [2004]). For more terminals diverse heuristics are applied (Hanan [1966], Takahashi and Matsuyama [1980]).

The *timing propagation graph* $G_p^{\mathcal{T}} = (V^{\mathcal{T}}, E_p^{\mathcal{T}})$ for the full chip is composed as follows. For each primary input or output pin $p \in P(\mathcal{I})$ a node is added to $V^{\mathcal{T}}$. For each circuit $c \in \mathcal{C}$ a copy of the model timing graph $G_{\mathcal{I}_\beta(c)}$ is added to $G_p^{\mathcal{T}}$. For each source sink pair (p, q) of a net four edges are added from p to q labeled (η, ζ, ζ) for $\eta \in \{early, late\}$ and $\zeta \in \{rise, fall\}$. They refer to all possible non-inverting transitions. We make the convention that functions that are defined on the model graph can be applied directly on $G_p^{\mathcal{T}}$. By slewlim(v), $v \in V^{\mathcal{T}}$, we refer to the slew limit of the input pin in the corresponding model graph. Figure 2.8 shows an example of a timing propagation graph, which consists of all nodes and the black edges. Red edges refer to timing constraints which will be introduced later in Section 2.4.6.

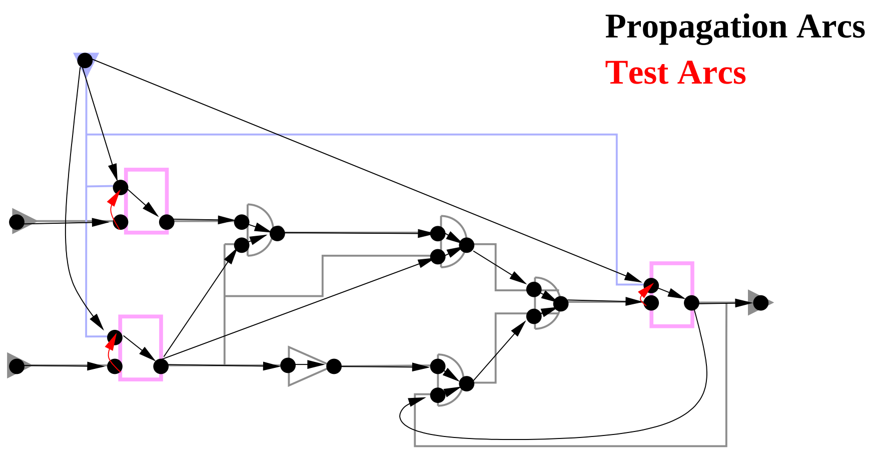


Figure 2.8: Example of a timing graph with parallel edges being collapsed

2.4.4 Signal Propagation

Signals are initialized on a selected set $V^{\mathcal{T}^{start}} \subset V^{\mathcal{T}}$ of *start nodes*. Typically $V^{\mathcal{T}^{start}}$ contains the primary inputs pins. For every $p \in V^{\mathcal{T}^{start}}$ a set $\mathfrak{S}(p)$ of four possible signals is asserted

$$\mathfrak{S}(p) := \{(p, early, rise), (p, early, fall), (p, late, rise), (p, late, fall)\} \tag{2.4}$$

Each signal $\sigma \in \mathfrak{S}(p)$ is characterized by its arrival time $\mathrm{at}(p, \sigma)$ and slew $\mathrm{slew}(p, \sigma)$. We will denote the timing mode of σ by $\eta(\sigma) \in \{early, late\}$ the transition by $\zeta(\sigma) \in \{\mathrm{rise}, \mathrm{fall}\}$. By η^{-1} we denote the inverse timing mode, that is, $\mathrm{early}^{-1} = \mathrm{late}$ and vice versa. Analogously, we write ζ^{-1} for the inverse transition of ζ, that is, $\mathrm{rise}^{-1} = \mathrm{fall}$ and vice versa.

Let us assume here that $G^{\mathcal{T}}_p$ is acyclic. We will discuss cycles in the propagation graph later in Section 5, Remark 5.9. Static timing analysis is a variant of the critical path method (CPM), which was invented jointly by the DuPont company and the Remington Rand Corporation in the 1950's for analyzing the duration of chemical processes. Kelley Jr. [1961] and Fulkerson [1961] are early references to the critical path method.

Starting at the signal sources, signals are propagated according to a topological ordering of $G^{\mathcal{T}}$. Let $q \in V^{\mathcal{T}}$ be a vertex with nonzero in-degree with all predecessors being processed already, that is, for all $(p, q) \in E^{\mathcal{T}}_p$ the set $\mathfrak{S}(p)$ of signals being propagated to the predecessor $p \in V^{\mathcal{T}}$ is already determined. Then for each $e \in \delta^-(q)$ with some label (η, ζ, ζ') we define the set $\mathfrak{S}(e)$ of signals propagated through e by adding $\sigma_q := (x, \eta, \zeta')$ to $\mathfrak{S}(e)$ for each $\sigma_p = (x, \eta, \zeta) \in \mathfrak{S}(p)$. The arrival times and slews propagated over the edge e are set by

$$\begin{aligned} \mathrm{at}(e, \sigma_q) \quad &:= \mathrm{at}(p, \sigma_p) + \vartheta_e(\mathrm{slew}(p, \sigma_p)) + \mathrm{adj}_e(\sigma_p, \sigma_q) \\ \mathrm{slew}(e, \sigma_q) \quad &:= \mathrm{slew}(p, \sigma_p) + \lambda_e(\mathrm{slew}(p, \sigma_p)) \end{aligned} \tag{2.5}$$

where ϑ_e/λ_e are either circuit delay/slew functions for a fixed downstream capacitance or Elmore delay/slew functions for a fixed wire topology, and $\mathrm{adj}_e(\sigma_p, \sigma_q)$ is some value that can be user defined or computed, and which is usually zero. Later we will see, how the adjust value is needed to describe the propagation of signals that flush through transparent latches.

The signal origin might be manipulated by a *phase rename*. That is, a signal label $\sigma_q := (x, \eta, \zeta) \in \mathfrak{S}(e)$ might be replaced by another label $\sigma'_q := (x', \eta, \zeta)$ while keeping the arrival time $\mathrm{at}(e, \sigma'_q) = \mathrm{at}(e, \sigma_q)$ and slew $\mathrm{slew}(e, \sigma'_q) = \mathrm{slew}(e, \sigma_q)$. The final signal set of e then becomes $(\mathfrak{S}(e) \setminus \{\sigma_q\}) \cup \{\sigma'_q\}$. Instead of removing the old signal, it could also be kept among the new signals. Furthermore several renames could be applied iteratively on an edge. Phase renames can be user defined, or automatically applied, for instance in connection with transparent latch timing.

To reduce the computational complexity in practice, the start nodes are often partitioned into a few groups with equal signal sets. For this purpose, a phase rename can be used to unify the signal sets within each group.

At q the signals propagated over all incoming edges are merged:

$$\mathfrak{S}(q) := \bigcup_{e \in \delta^-(q)} \mathfrak{S}(e).$$

Arrival times and slews in q for $\sigma \in \mathfrak{S}(q)$ are now given by

$$\text{at}(q,\sigma) := \max\left\{\text{at}(e,\sigma) : e \in \delta^-(q), \sigma \in \mathfrak{S}(e)\right\}, \tag{2.6}$$

$$\begin{aligned}\text{slew}(q,\sigma) := \max\{&\text{slew}(e,\sigma) + \nu \cdot (\text{at}(e,\sigma) - \text{at}(q,\sigma)) : \\ &e \in \delta^-(q), \sigma \in \mathfrak{S}(e)\},\end{aligned} \tag{2.7}$$

if $\eta(\sigma) = \textit{late}$, and

$$\text{at}(q,\sigma) := \min\left\{\text{at}(e,\sigma) : e \in \delta^-(q), \sigma \in \mathfrak{S}(e)\right\}, \tag{2.8}$$

$$\begin{aligned}\text{slew}(q,\sigma) := \min\{&\text{slew}(e,\sigma) + \nu \cdot (\text{at}(e,\sigma) - \text{at}(q,\sigma)) : \\ &e \in \delta^-(q), \sigma \in \mathfrak{S}(e)\},\end{aligned} \tag{2.9}$$

if $\eta(\sigma) = \textit{early}$ assuming $0 \cdot \infty = 0$.

The parameter ν was traditionally set to ∞, which implies that the slew in q is the induced slew of the latest incoming signal, where $\text{at}(q,\sigma) = \text{at}(e,\sigma)$. This can result in too optimistic values. Vygen [2006] showed how to choose ν individually for each pin definition, in a non-optimistic and least pessimistic way. However, many industrial timing engines provide only the global settings $\nu \in \{0, 2, \infty\}$. The value $\nu = 2$ corresponds to combining the latest arrival time with the latest (earliest) possible saturation, which is the 90% transition of a rising signal and 10% transition for a falling signal. According to Blaauw et al. [2000] this is a sufficiently pessimistic global choice.

So far, we have described the signal propagation through the timing graph. Now, we introduce constraints on the signals.

2.4.5 Electrical Correctness Constraints

To compute valid arrival times and slews, all capacitance and slew limits must be obeyed. Otherwise the timing rules are called outside their domain. In addition to the capacitance and slew limits given by the timing rules, there are usually capacitance limits for primary input pins as well as slew limits for primary output pins.

A source pin $p \in P_{in}(N)$ of a net $N \in \mathcal{N}$ is *electrically correct* if the capacitance limit is met:

$$\text{downcap}(p, \text{late}) \le \text{caplim}(p). \tag{2.10}$$

A sink pin $p \in P_{out}(N)$ of a net $N \in \mathcal{N}$ is *electrically correct* if the slew limit is met:

$$\text{slew}(p,\sigma) \le \text{slewlim}(p, \zeta(\sigma)) \text{ for all } \sigma \in \mathfrak{S}(p). \tag{2.11}$$

A net $N \in \mathcal{N}$ or a circuit $c \in \mathcal{C}$ is *electrically correct* if all its pins are.

A chip is electrically correct if all its nets are electrically correct. This implies also the electrical correctness of all circuits $c \in \mathcal{C}$ and all IO-ports. To compute valid slews, capacitance limits must be met. Therefore, capacitance violations are usually considered as more severe.

2.4.6 Arrival Time Constraints

If the electrical correctness of a chip is given, all computed arrival times and slews are reliable, apart from uncertainties in the timing rules. The arrival time constraints introduced in this section ensure that the chip will work correctly at the intended speed.

Let us first consider a simple (transparent) latch with a data input $d \in V^{\mathcal{T}}$ as a *tested vertex* and a clock input $c \in V^{\mathcal{T}}$ as a *testing vertex* (see Figure 2.5 on page 12). A periodic clock signal arriving in c opens and closes the latch once per cycle.

Setup Test

In the *conservative* setup test, a late (tested) signal $\sigma_d \in \mathfrak{S}(d)$ must arrive before the register opens and releases the data for the next cycle. This constraint is represented by the red arc in Figure 2.5. More precisely, the voltage state at the data input d must have become stable some time before the register opens. This, so called *setup time* $\text{setup}\,(\text{slew}(d, \sigma_d), \text{slew}(c, \sigma_c))$, depends on the slews at both test ends. While the conservative setup test holds also for flip-flops and most other register type, we will give a slightly relaxed “non-conservative” definition of the setup test at transparent latches in Remark 2.1.

Both the data and the clock arrival times usually refer to the same cycle. Therefore, $\text{at}(c, \sigma_c)$ must be adjusted to a later (usually the next) cycle. This adjust value is called *cycle adjust*. But the signals σ_d and σ_c can be of different frequency, or multiple cycles can be allowed for σ_d to enter d.

In general a *cycle adjust* $\text{adj}(\sigma_d, \sigma_c)$ defines an adjustment for σ_c, that must be applied to the computed arrival time before comparing with σ_d. The *late mode* constraints resulting at a latch can now be formulated as

$$\text{at}(d, \sigma_d) + \text{setup}\,(\text{slew}(d, \sigma_d), \text{slew}(c, \sigma_c)) \leq \text{at}(c, \sigma_c) + \text{adj}(\sigma_d, \sigma_c). \tag{2.12}$$

Now we give a short description of the cycle adjust calculation. The ideal oscillation of a clock signal is defined by three numbers $0 \leq t^{lead} < t^{trail} \leq T$, where T is the cycle time, $t^{lead} + kT$ is the ideal occurrence of the so called *leading clock edge* in the k-th computation cycle and $t^{trail} + kT$ is the ideal occurrence the *trailing clock edge* in the k-th computation cycle, with $k \in \mathbb{N}_0$. Usually, the leading edge corresponds to the rising clock signal at an oscillator or PLL output, and also to the opening edge of the registers behind, while the (inverse) trailing edge closes the registers. When propagating through a chain of inverters, a clock edge is alternatingly of type rise and fall.

Let $t_d^{lead}, t_d^{trail}, T_d$ be the clock definitions of the clock signal that triggers σ_d (at some register), and $t_c^{lead}, t_c^{trail}, T_c$ be the clock definition of σ_c. Furthermore, let $t_d \in \{t_d^{lead}, t_d^{trail}\}$ be the time of the particular reference edge that triggers σ_d, and let $t_c \in \{t_c^{lead}, t_c^{trail}\}$ be the time of the particular reference edge to which σ_c refers. Then the minimum positive difference

$$\begin{aligned}&\operatorname{mpd}(\sigma_d, \sigma_c) := \\ &\quad \min\Big\{t_c + iT_c - (t_d + jT_d) \,\Big|\, i, j \in \mathbb{N}_0; t_c + iT_c - (t_d + jT_d) > 0\Big\}\end{aligned} \tag{2.13}$$

is the most pessimistic travel time limit for signal σ_d on the data path, which is the path from the register which triggers σ_d to the current register where σ_d is captured by σ_c. The adjust value that must be added to the clock arrival time $\operatorname{at}(\sigma_c)$ is then given by

$$\operatorname{adj}(\sigma_d, \sigma_c) := t_d - t_c + \operatorname{mpd}(\sigma_d, \sigma_c) + (\operatorname{mc}_{\sigma_d,\sigma_c} - 1)T_d, \tag{2.14}$$

where the parameter $\operatorname{mc}_{\sigma_d,\sigma_c} \in \mathbb{N} \cup \{\infty\}$ is the maximum number of cycles the data signal is allowed to travel. It is usually called *multi-cycle adjust* and is mostly set to $\operatorname{mc}_{\sigma_d,\sigma_c} = 1$. If $\operatorname{mc}_{\sigma_d,\sigma_c} = \infty$, there will be no effective test, that is, σ_d and σ_c are asynchronous.

Remark 2.1. *(Transparent Latch Timing)*
A transparent latch contains a propagation arc from the data input to the data output. On such an arc, a phase rename and an adjust on a signal propagation arc are applied. The outgoing data signal is controlled and thus should have the same origin label as the incoming opening clock signal σ_c. This results in a phase rename from σ'_d to σ'_c on the data input to output edge, where σ'_c is a signal propagated from the clock input and σ'_d the preliminary signal propagated from the data input. In order to shift the output signal back to the cycle of the clock signal, a so called flush-adjust $\operatorname{adj}_e(\sigma_d, \sigma'_d) = -\operatorname{adj}(\sigma_d, \sigma_c)$ must be applied on the propagation arc e.

The setup test (2.12) represents a conservative restriction for a transparent latch that decouples the incoming data signal from outgoing data signals. If the data input signals are required to arrive before the latch opening time, the data output arrival times depend only on the opening time of the latch. Thus, the signal propagation graph can be considered as acyclic. However, the latch would still work correctly if the incoming data signal arrives during the open phase of the next cycle, which would then result in a later output arrival time. In this scenario σ_c represents the trailing clock edge, while the flush-adjust on the data input to output edge would be computed with respect to the opening edge.

Hold Test

Besides late mode constraints, signals must not arrive too early at d, because the voltage state at the output must be stable while the latch is open. Let now σ_d be an early signal in d. Similar to the setup time for late mode tests, a *hold time* $\operatorname{hold}(\operatorname{slew}(d, \sigma_d), \operatorname{slew}(c, \sigma_c))$ specifies how long the input must be stable after the latch closes, where σ_c is now the latest closing signal at the latch. The arrival time of σ_c is again adjusted by some number $\mathrm{adj}(\sigma_d, \sigma_c)$. The *early mode* constraints read as follows:

$$\mathrm{at}(d, \sigma_d) + \mathrm{hold}\Big(\mathrm{slew}(d, \sigma_d), \mathrm{slew}(c, \sigma_c)\Big) \geq \mathrm{at}(c, \sigma_c) + \mathrm{adj}(\sigma_d, \sigma_c) \tag{2.15}$$

The hold adjust is computed differently from the setup adjust. Let t_d, T_d, t_c, and T_c be defined as before. For the setup test t_c refered to the opening edge. Now σ_c and therefore t_c refer to the closing edge. The minimum non-negative difference

$$\begin{aligned}&\mathrm{mnnd}(\sigma_d, \sigma_c) := \\ &\quad \min\Big\{t_d + iT_d - (t_c + jT_c) \,\Big|\, i, j \in \mathbb{N}_0; t_d + iT_d - (t_c + jT_c) \geq 0\Big\}\end{aligned}$$

is the most pessimistic lower travel time limit for signal σ_d on the data path. The adjust value is then given by

$$\mathrm{adj}(\sigma_d, \sigma_c) := t_d - t_c + \mathrm{mnnd}(\sigma_d, \sigma_c) + (\mathrm{mc}_{\sigma_d, \sigma_c} - 1)T_d, \tag{2.16}$$

Here $\mathrm{mc}_{\sigma_d, \sigma_c} \in \mathbb{N}_0 \cup \{-\infty\}$ determines the number of cycles the data signal must at least propagate before arriving at d, and $\mathrm{mc}_{\sigma_d, \sigma_c} = -\infty$ is used for asynchronous signals.

In the relevant “synchronous” case the signals σ_d and σ_c emerge from a common clock source with reference cycle time T^{ref}. However, σ_d and σ_c can still be derived from different clock definitions with different frequencies. Certain circuits such as PLLs can create a signal with cycle time qT^{ref}, $q \in \mathbb{Q}_+$.

Note that for both, setup constraints (2.12) and hold constraints (2.15), the difference $t_d - t_c$, which is part of $\mathrm{adj}(\sigma_d, \sigma_c)$ cancels out in the sum

$$\mathrm{at}(c, \sigma_c) - \mathrm{at}(d, \sigma_d) + \mathrm{adj}(\sigma_d, \sigma_c) \tag{2.17}$$

as the values t_d and t_c are implicitly contained in $\mathrm{at}(c, \sigma_c)$ and $\mathrm{at}(d, \sigma_d)$. Therefore, (2.17) will increase if the underlying reference cycle time T^{ref} increases by some $\Delta T^{ref} \in \mathbb{R}_+$. Consequently, the setup constraints (2.12) are relaxed by

$$\mathrm{mpd}(\sigma_d, \sigma_c) \cdot \Delta T^{ref}, \tag{2.18}$$

and the hold constraints (2.15) are invariant or tightened by

$$\mathrm{mnnd}(\sigma_d, \sigma_c) \cdot \Delta T^{ref} \tag{2.19}$$

when increasing the reference cycle time by ΔT^{ref}.

More complex registers such as flip-flops, SRAMs, or non-sequential clock gates have similar constraints that are representable as inequalities between data input and clock input arrival times. All constraints between two signals have in common that they express a race-condition between two signals which originate from a common source, for example from a PLL. However, this common source may be located off-chip.

The register timing constraints or generally book specific constraints are given by the timing rules, which also provide the setup and hold functions.

Primary Output Constraints

Apart from register timing constraints there are *primary output constraints.* There can either be a *predefined required arrival times* $\mathrm{rat}(p,\sigma)$ for a signal-pin pair $(p,\sigma), \sigma \in \mathfrak{S}(p)$ that do not depend on other signals, or *user defined tests* between any two signals $p, \sigma_p \in \mathfrak{S}(p)$, $q, \sigma_q \in \mathfrak{S}(q)$ in the design. Predefined required arrival times define an inequality constraint:

$$\mathrm{at}(p,\sigma) \le \mathrm{rat}(p,\sigma) \qquad \text{if } \eta(\sigma) = \text{late, and} \tag{2.20}$$

$$\mathrm{at}(p,\sigma) \ge \mathrm{rat}(p,\sigma) \qquad \text{if } \eta(\sigma) = \text{early}\,. \tag{2.21}$$

User defined tests can be represented by inequalities similar to (2.12) between a late and an early signal, with usually constant setup and adjust times.

$$\mathrm{at}(p,\sigma_p) + \mathrm{setup}\Big(\mathrm{slew}(p,\sigma_p), \mathrm{slew}(q,\sigma_q)\Big) \le \mathrm{at}(q,\sigma_q) + \mathrm{adj}(\sigma_p,\sigma_q). \tag{2.22}$$

User defined test can often be found between a set of primary output signals, which may leave the chip at any time, but simultaneously. Sometimes user defined tests are set between primary outputs and primary inputs, and replace predefined start times and required arrival times for data signals. Although intended for modeling primary output constraints, both constraint types can basically be found everywhere on a design.

2.4.7 Timing Graph

As timing constraints describe the relation between two signals, they are often represented as arcs between the early test end to the late test end on the underlying timing nodes. A *test arc* (p,q) between two timing nodes $p, q \in V^{\mathcal{T}}$ is labeled by a triple

$$(\eta, \zeta, \zeta') \in \{early, late\} \times \{rise, fall\} \times \{rise, fall\},$$

and is created if there is a test between a signal $\sigma_p \in \mathfrak{S}(p)$ of timing mode $\eta(\sigma_p) = \eta$ and transition $\zeta(\sigma_p) = \zeta$ and a signal $\sigma_q \in \mathfrak{S}(q)$ of timing mode $\eta(\sigma_q) = \eta^{-1}$. The set of *test arcs* is denoted $E_t^{\mathcal{T}}$.

Definition 2.2. *The graph* $G^{\mathcal{T}} = (V^{\mathcal{T}}, E^{\mathcal{T}})$ *is called the timing graph, where* $E^{\mathcal{T}} := E_p^{\mathcal{T}} \cup E_t^{\mathcal{T}}$ *is the set of timing edges.*

2.4.8 Slacks

Each timing constraint induces a required arrival time (*rat*) for the tested signal. The setup test (2.12) transforms to

$$\mathrm{at}(d,\sigma_d) \;\le\; \mathrm{rat}(d,\sigma_d), \tag{2.23}$$

with

$$\begin{aligned}\operatorname{rat}(d,\sigma_d) \;:=\; & \operatorname{at}(c,\sigma_c)+\operatorname{adj}(\sigma_d,\sigma_c)\\ & -\operatorname{setup}\Big(\operatorname{slew}(d,\sigma_d),\operatorname{slew}(c,\sigma_c)\Big).\end{aligned} \tag{2.24}$$

The hold test (2.15) becomes

$$\operatorname{at}(d,\sigma_d) \;\geq\; \operatorname{rat}(d,\sigma_d), \tag{2.25}$$

with

$$\begin{aligned}\operatorname{rat}(d,\sigma_d) \;:=\; & \operatorname{at}(c,\sigma_c)+\operatorname{adj}(\sigma_d,\sigma_c)\\ & -\operatorname{hold}(\operatorname{slew}(d,\sigma_d),\operatorname{slew}(c,\sigma_c)).\end{aligned} \tag{2.26}$$

The required arrival times are propagated to every vertex in the timing graph in backward topological order.

$$\begin{aligned}\operatorname{rat}(p,\sigma_p) := \; & \min\{\operatorname{rat}(q,\sigma_q)-\vartheta_e(\operatorname{slew}(p,\sigma_p))-\operatorname{adj}_e(\sigma_p,\sigma_q)\mid\\ & e\in\delta^+(p),\sigma_q\in\mathfrak{S}(e) \text{ induced by } \sigma_p\},\end{aligned} \tag{2.27}$$

if $\eta(\sigma)=late$, and

$$\begin{aligned}\operatorname{rat}(p,\sigma_p) := \; & \max\{\operatorname{rat}(q,\sigma_q)-\vartheta_e(\operatorname{slew}(p,\sigma_p))-\operatorname{adj}_e(\sigma_p,\sigma_q)\mid\\ & e\in\delta^+(p),\sigma_q\in\mathfrak{S}(e) \text{ induced by } \sigma_p\},\end{aligned} \tag{2.28}$$

if $\eta(\sigma)=early$.

The difference between required and computed arrival time of a signal $\sigma\in\mathfrak{S}(p), p\in V^{\mathcal{T}}$ is called *slack*. It is defined as

$$\operatorname{slk}(p,\sigma):=\operatorname{rat}(p,\sigma)-\operatorname{at}(p,\sigma), \tag{2.29}$$

if $\eta(\sigma)=late$, and as

$$\operatorname{slk}(p,\sigma):=\operatorname{at}(p,\sigma)-\operatorname{rat}(p,\sigma), \tag{2.30}$$

if $\eta(\sigma)=early$.

Assuming individual but constant slews $\operatorname{slew}(p,\sigma)$ for all $\sigma\in\mathfrak{S}(p)$ and all $p\in V^{\mathcal{T}}$, following properties of the slack values can be obtained by simple calculation.

Remark 2.3. *Let $p\in V^{\mathcal{T}}$ and $\sigma\in\mathfrak{S}(p)$ be a late signal. Then* $\operatorname{slk}(p,\sigma)$ *specifies the maximum delay, which—when added either to all incoming or to all outgoing edges of p—guarantees, that the timing constraints of signal σ at any tested vertex reachable from p are satisfied.*

If $\sigma\in\mathfrak{S}(p)$ is an early signal $\operatorname{slk}(p,\sigma)$ *specifies the maximum delay, which—when removed either from all incoming or from all outgoing edges of p—guarantees, that the timing constraints of signal σ and any tested vertex reachable from p.*

Remark 2.4. *Let $p \in V^{\mathcal{T}}$ and $\sigma \in \mathfrak{S}(p)$, then $\operatorname{slk}(p,\sigma) \geq 0$ if and only if the timing constraints of all paths of signal σ through p are satisfied.*

The slack $\operatorname{slk}(p,\sigma)$ specifies the worst slack of a signal path containing the pair $(p,\sigma), p \in P, \sigma \in \mathfrak{S}(p)$. We can also define the slack $\operatorname{slk}(p,\sigma_p,q,\sigma_q)$ of a timing arc $(p,q) \in E^{\mathcal{T}}$ and two related signals $\sigma_q \in \mathfrak{S}(p), \sigma_q \in \mathfrak{S}(q)$ as the worst slack of a path that contains this arc and signals. Like the slack of a pin-signal pair, it can be computed from the local information at the arc and its incident nodes.

For a test arc $(p,q) \in E_t^{\mathcal{T}}$ with the relation $\operatorname{at}(p,\sigma_p)+d(p,\sigma_p,q,\sigma_q) \leq \operatorname{at}(q,\sigma_q)$ (after simple algebraic transformation), where $\sigma_p \in \mathfrak{S}(p), \sigma_q \in \mathfrak{S}(q)$, and $d(p,\sigma_p,q,\sigma_q)$ summarizes the setup or hold time and the adjust, the slack is simply

$$\operatorname{slk}(p,\sigma_p,q,\sigma_q) := \operatorname{at}(q,\sigma_q) - \operatorname{at}(p,\sigma_p) - d(p,\sigma_p,q,\sigma_q). \tag{2.31}$$

For a propagation arc $(p,q) \in E_p^{\mathcal{T}}$ the situation is slightly different. Given the relation $\operatorname{at}(p,\sigma_p) + d(p,\sigma_p,q,\sigma_q) \leq \operatorname{at}(q,\sigma_q)$, where $\sigma_p \in \mathfrak{S}(p), \sigma_q \in \mathfrak{S}(q)$, and $d(p,\sigma_p,q,\sigma_q)$ summarizes the delay and the adjust, we must consider the required arrival time of (q,σ_q) if $\eta(\sigma_p) = \text{late}$ and of (p,σ_p) if $\eta(\sigma_p) = \text{late}$ to compute the worst slack of a path through (p,σ_p,q,σ_q). The slack is given by

$$\operatorname{slk}(p,\sigma_p,q,\sigma_q) := \operatorname{rat}(q,\sigma_q) - \operatorname{at}(p,\sigma_p) - d(p,\sigma_p,q,\sigma_q), \tag{2.32}$$

if $\eta(\sigma_p) = \text{late}$, and

$$\operatorname{slk}(p,\sigma_p,q,\sigma_q) := \operatorname{at}(q,\sigma_q) - \operatorname{rat}(p,\sigma_p) - d(p,\sigma_p,q,\sigma_q), \tag{2.33}$$

if $\eta(\sigma_p) = \text{early}$.

If slews are not constant, the required arrival times defined in (2.27) and (2.28) are unsafe, in the sense that Remarks 2.3 and 2.4 might not be true. This is because the rat propagation does not account for the slew effects on the delays and slews in the forward cone. As for the slew propagation, Vygen [2001, 2006] gave a safe rat propagation formula, that considers the slew effects as well. Given $\nu \in (0,\infty]$ as in (2.6) and (2.8) safe required arrival times are propagated as follows:

$$\begin{aligned}\operatorname{rat}(p,\sigma_p) \quad := \min\Big\{&\operatorname{rat}(q,\sigma_q) - \vartheta_e(\operatorname{slew}(p,\sigma_p)) - \operatorname{adj}_e(\sigma_p,\sigma_q)\\ &- \tfrac{1}{\nu}\max\{0, \lambda_e(\operatorname{slew}(p,\sigma_p) - \operatorname{slew}(q,\sigma_q))\ :\\ &e = (p,q) \in \delta^+(p), \sigma_q \in \mathfrak{S}(e)\Big\},\end{aligned} \tag{2.34}$$

if $\eta(\sigma) = \textit{late}$, and

$$\begin{aligned}\operatorname{rat}(p,\sigma_p) \quad := \max\Big\{&\operatorname{rat}(q,\sigma_q) - \vartheta_e(\operatorname{slew}(p,\sigma_q))\operatorname{adj}_e(\sigma_p,\sigma_q)\\ &- \tfrac{1}{\nu}\min\{0, \lambda_e(\operatorname{slew}(p,\sigma_p) - \operatorname{slew}(q,\sigma_q))\ :\\ &e = (p,q) \in \delta^+(p), \sigma_q \in \mathfrak{S}(e)\Big\},\end{aligned} \tag{2.35}$$

if $\eta(\sigma) = \textit{early}$.

However, in industrial timing engines these rules are not used. The main reason is that on critical paths slews tend to be similarly tight after optimization. Therefore, the difference between classical and improved propagation rules emerge hardly. For final timing sign-off most critical paths will be analyzed individually anyway.

2.4.9 Signal Graph

In this Section we describe how arrival time constraints and feasible node potentials in directed graphs are related, assuming delays and slews to be fixed. First, we relax the formulation of the arrival time propagation rules, by replacing the maximum (2.6) and minimum (2.8) functional equations by inequalities. If the signal arrival times fulfill all late mode propagation rules (2.6), they will also fulfill the following inequality:

$$\mathrm{at}(q,\sigma_q) \geq \mathrm{at}(e,\sigma_p) = \mathrm{at}(p,\sigma_p) + \vartheta_e(\mathrm{slew}(p,\sigma_p)) + \mathrm{adj}_e(\sigma_p,\sigma_q) \tag{2.36}$$

for all $e = (p,q) \in E_p^{\mathcal{T}}$, $\sigma_q \in \mathfrak{S}(q)$, and $\sigma_p \in \mathfrak{S}(p)$, where e is labeled by $(\mathrm{late}, \zeta(\sigma_p), \zeta(\sigma_q))$, and $\eta(\sigma_p) = \eta(\sigma_p) = \mathrm{late}$. Accordingly, if signal arrival times fulfill all early mode propagation rules (2.8), they will also fulfill the relaxed inequality formulation:

$$\mathrm{at}(q,\sigma_q) \leq \mathrm{at}(e,\sigma_p) = \mathrm{at}(p,\sigma_p) + \vartheta_e(\mathrm{slew}(p,\sigma_p)) + \mathrm{adj}_e(\sigma_p,\sigma_q) \tag{2.37}$$

for all $e = (p,q) \in E_p^{\mathcal{T}}$, $\sigma_q \in \mathfrak{S}(q)$, and $\sigma_p \in \mathfrak{S}(p)$, where e is labeled by $(\mathrm{early}, \zeta(signal_p), \zeta(\sigma_q))$, and $\eta(\sigma_p) = \eta(\sigma_p) = \mathrm{early}$.

Now all propagation and test relations represent inequalities between two arrival times plus some constant delay. Thus, they can be represented by an edge-weighted *signal graph* $(G^{\mathfrak{S}}, c^{\mathfrak{S}})$.

The vertex set $V^{\mathfrak{S}} := V(G^{\mathfrak{S}})$ is given by all signal/pin pairs and some extra node v_0:

$$V^{\mathfrak{S}} := \{(p,\sigma)\,|\, p \in P, \sigma \in \mathfrak{S}(p)\} \cup \{v_0\},$$

where v_0 represents the time '0'.

There are different type of edges:

- for each late propagation inequality (2.36) an edge $e = \big((q,\sigma_q),(p,\sigma_p)\big)$ with edge cost $c^{\mathfrak{S}}(e) = -\,\vartheta_e(\mathrm{slew}(p,\sigma_p)) - \mathrm{adj}_e(\sigma_p,\sigma_q)$;

- for each early propagation inequality (2.37) an edge $e = \big((p,\sigma_p),(q,\sigma_q)\big)$ with edge cost $c^{\mathfrak{S}}(e) = \vartheta_e(\mathrm{slew}(p,\sigma_p)) + \mathrm{adj}_e(\sigma_p,\sigma_q)$;

- for each late start time assertion $\mathrm{at}(p,\sigma), p \in V^{\mathcal{T}^{start}}, \eta(\sigma) = \mathrm{late}$, an edge $e = \big((p,\sigma),v_0\big)$ with edge cost $c(e) := -\,\mathrm{at}(p,\sigma)$;

- for each early start time assertion $\mathrm{at}(p,\sigma), p \in V^{\mathcal{T}^{start}}, \eta(\sigma) = \mathrm{early}$, an edge $e = \big(v_0,(p,\sigma)\big)$ with edge cost $c^{\mathfrak{S}}(e) := \mathrm{at}(p,\sigma)$;

- for each setup test (2.12) an edge $e = \big((c,\sigma_c),(d,\sigma_d)\big)$ with edge cost $c^{\mathfrak{S}}(e) = \mathrm{adj}(\sigma_d,\sigma_c) - \mathrm{setup}\big(\mathrm{slew}(d,\sigma_d), \mathrm{slew}(c,\sigma_c)\big)$;

- for each hold test (2.15) an edge $e = \big((d, \sigma_d), (c, \sigma_c)\big)$ with edge cost $c^{\mathfrak{S}}(e) = \text{hold}\big(\text{slew}(d, \sigma_d), \text{slew}(c, \sigma_c)\big) - \text{adj}(\sigma_d, \sigma_c)$;
- for each user defined test (2.22) an edge $e = \big((q, \sigma_q), (p, \sigma_q)\big)$ with edge cost $c^{\mathfrak{S}}(e) = \text{adj}(\sigma_d, \sigma_c) - \text{setup}\big(\text{slew}(p, \sigma_p), \text{slew}(q, \sigma_q)\big)$;
- for each predefined late-mode required arrival time $\text{rat}(p, \sigma), p \in V^{\mathcal{T}}$, $\eta(\sigma) =$ late, an edge $e = \big(v_0, (p, \sigma)\big)$ with edge cost $c^{\mathfrak{S}}(e) := \text{rat}(p, \sigma)$;
- for each predefined early-mode required arrival time $\text{rat}(p, \sigma), p \in V^{\mathcal{T}}, \eta(\sigma) =$ early, an edge $e = \big((p, \sigma), v_0\big)$ with edge cost $c^{\mathfrak{S}}(e) := -\text{rat}(p, \sigma)$.

The first four types of edges are signal initialization and propagation relations. We denote the set of these edges by $E_p^{\mathfrak{S}}$. The last five types of edges correspond to the arrival time tests, and we denote the set of these edges by $E_t^{\mathfrak{S}}$.

As usual we define a node potential $\pi : V \to \mathbb{R}$ for a directed graph (V, E) with edge weights $c : E \to \mathbb{R}$ to be *feasible*, if

$$\pi(v) + c(v, w) \geq \pi(w) \tag{2.38}$$

holds for all $(v, w) \in E$. The difference $c_\pi(v, w) := c(v, w) + \pi(v) - \pi(w)$ is called the *reduced cost* of $(v, w) \in E$, which we will also call *slack*, the customary term in chip design.

The correspondence between node potentials in the signal graph and the arrival time constraints in static timing analysis is given in following Lemma.

Lemma 2.5. *Assume that all delays, setup and hold times are pre-computed and fixed. Let π be a node potential for the signal graph $(G^{\mathfrak{S}}, c^{\mathfrak{S}})$ with non-negative reduced costs $c_\pi(e)$ for all $e \in E_p^{\mathfrak{S}}$. Then each slack of an arrival time constraint is greater than or equal to the reduced cost of the corresponding test edge in $E^{\mathfrak{S}} := E(G^{\mathfrak{S}})$.*

Proof. Let $\pi : V^{\mathfrak{S}} \to \mathbb{R}$ be a feasible node potential, and let

$$\text{at} : \{(x, \sigma_x) \mid x \in P, \sigma_x \in \mathfrak{S}(x)\} \to \mathbb{R}$$

be the static timing arrival times. Due to the relaxation of the propagation and start time initialization rules we have $\pi((p, \sigma)) \geq \text{at}(p, \sigma)$ for all $(p, \sigma) \in \{(x, \sigma_x) \mid x \in P, \sigma_x \in \mathfrak{S}(x), EL(\sigma) = \text{late}\}$, and $\pi((p, \sigma)) \leq \text{at}(p, \sigma)$ for all $(p, \sigma) \in \{(x, \sigma_x) \mid x \in P, \sigma_x \in \mathfrak{S}(x), EL(\sigma) = \text{early}\}$.

As late arrival times are not greater than the corresponding node potential and early arrival times are not smaller than the corresponding node potential, the slacks in timing analysis cannot be smaller than the corresponding reduced costs c_π.

□

Besides this simple result we will use the signal graph in Chapter 5 to optimize clock schedules and for timing analysis in presence of cycles in the timing propagation graph $G_p^{\mathcal{T}}$.

2.5 Sign-Off Timing Constraints

Due to its computational efficiency the static timing analysis is used during prerouting physical synthesis, and therefore it is the basis of this thesis. In this section we give a short overview on the more accurate but runtime-intensive postrouting timing-analysis. Though many effects can only be analyzed when a routing is given, their impact can be considered to some extent during prerouting optimization, especially within clocktree design. The main difference between pre- and postrouting timing analysis are the inclusion of coupling between adjacent wires and transistors, and the uncertainties due to process variations.

Delay variations occur due to different reasons. The environmental conditions, such as temperature and voltage, have big impact on the delay. Within a typical temperature range of -40 to $125°$ Celsius and a voltage range of 0.8 to 1.25 V, delays differ by a factor of 2 or more. This means that under cold, high voltage conditions, signals propagate about twice as fast as under hot, low voltage conditions. But the same chip has to work correctly under both conditions.

Other examples for variation are caused by

- fabrication variation during lithography affecting the width, length and thickness of wires, transistors, etc.,
- a hardly predictable voltage and temperature distribution on a chip in addition to varying external temperature and voltage,
- shape transformations during the lifetime of a chip caused by electromigration.

The variation sources are usually split into two sets. First, there are *inter-chip variations* that occur only on different chips or under different environmental conditions. Second, there are *intra-chip variations*, also known as *on-chip variations*, that occur on a single chip under a single environment condition.

In a sign-off timing check a static timing analysis is performed for a finite set of so called *process corners*. Each process corner consists of extreme settings for certain variation parameters, for example high temperatures combined with low voltage and slow devices. In each process corner “small variations” are assumed between early and late mode timing. Theoretically this sign-off methodology does not prove that a produced chip will work, because not all combinations of process parameters are checked for running time reasons. However, the large amount of pessimism in the process assumptions ensures the functionality in practice. Section 2.5 shows how some of the small variation pessimism can be removed.

The future trend for sign-off timing analysis seems to be the statistical analysis. This targets to predict the final yield, which is the percentage working chips. Although being discussed in the literature for a while, the running time overhead and limited accuracy of these models have not yet led to a broad application in industry.

Common Path Analysis

Static timing analysis makes several pessimistic assumptions to reduce the running time and memory effort. One such pessimism is to compare always earliest with latest signal arrival times at the timing tests. Now, throughout a path the early delays are smaller than or equal to the late delays to model process variations. In most cases both paths on which the compared arrival times were propagated to the test ends have a common starting subpath. Most variations on this common subpath apply equally to both paths and need not be incorporated into the slack computation.

Methods for the removal of this pessimism were given by Hathaway et al. [1997] and Zejda and Frain [2002]. In this section we summarize the basic ideas.

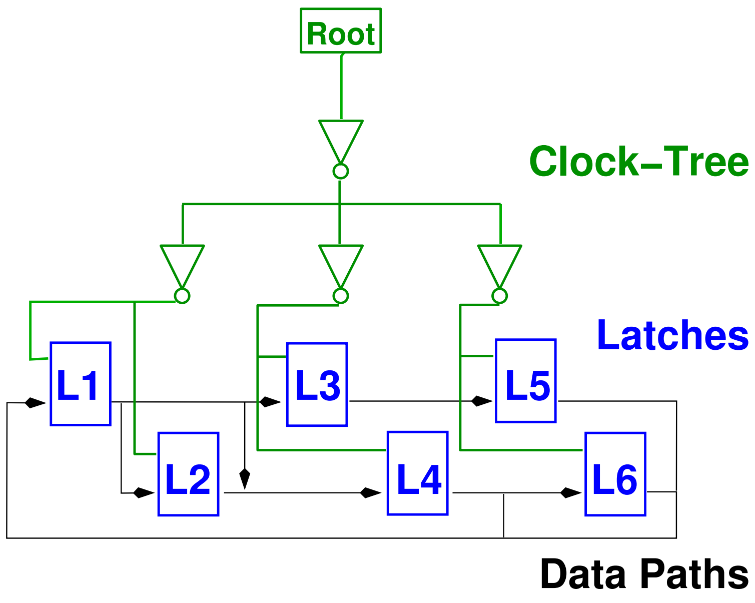


Figure 2.9: A clocktree example.

As an example consider the clocktree in Figure 2.9. The arrival times at the data input pin of latch $L3$ is triggered by a clock signal at the latch $L1$. Therefore, the data input arrival times at $L3$ are propagated on a path from the clocktree source through $L1$ to $L3$. Both paths, the data signal path and the clock signal path at $L3$, share a common path behind the clock root up to the first inverter.

Assume that in the example the variation of each clock net is 1, and circuits have no variation. Then the difference of the latest and earliest arrival time at a clock input is 3. If the cycle time would be 5, data paths between latches must have length no more than 2 to satisfy setup constraints.

However, the variations on the common path apply equally to the clock path and data path. In the example the maximum possible variation between any two latches is 2, as all paths share the first net behind the root. For certain paths, the actual variation is even smaller, for instance the variation difference between $L1$ and $L2$ is one, for the self loop of $L6$ there is, in fact, no variation difference.

Consider the setup test in (2.12) between a tested signal σ_d at a pin d and a testing signal σ_c at a pin c, and let $\text{cppr}(\sigma_d, \sigma_c)\mathbb{R}_{\geq 0}$ be the amount of variation that applies equally to both paths, then the setup test becomes:

$$\begin{aligned} &\text{at}(d, \sigma_d) + \text{setup}\Big(\text{slew}(d, \sigma_d), \text{slew}(c, \sigma_c)\Big) \\ \leq\ &\text{at}(c, \sigma_c) + \text{adj}(\sigma_d, \sigma_c) + \text{cppr}(\sigma_d, \sigma_c). \end{aligned} \tag{2.39}$$

Let $p, q, \in V^{\mathcal{T}}$ be the start and end vertices of the common path. In the usual case, where σ_d and σ_c are generated by the same transitions on the common path , that is, there are signals $\sigma_p^{early}, \sigma_p^{late} \in \mathfrak{S}(p)$ with $\zeta(\sigma_p^{early}) = \zeta(\sigma_p^{late})$, and $\sigma_q^{early}, \sigma_q^{late} \in \mathfrak{S}(q)$ with $\zeta(\sigma_q^{early}) = \zeta(\sigma_q^{late})$ such that σ_c is generated by σ_p^{early} and σ_q^{early}, and σ_d is generated by σ_p^{late} and σ_q^{late}, the *common path pessimism removal* adjust $\text{cppr}(\mathit{signal}_d, \mathit{signal}_c)$ is given by

$$\text{cppr}(\sigma_d, \sigma_c) = \text{at}(\sigma_q^{late}) - \text{at}(\sigma_q^{early}) - \Big(\text{at}(\sigma_p^{late}) - \text{at}(\sigma_p^{early})\Big). \tag{2.40}$$

In the general case different signal transitions on the common path may generate σ_d and σ_c. Then, a detailed variation analysis is required, to differentiate the common variation in the wires, and non-common variations in the transistors.

A common path analysis for every timing-test would be too time-consuming. The effort of static timing analysis is linear in the design instance size. The effort for common-path analysis grows with every bifurcation in the clocktree. Therefore, the general goal of timing optimization is to satisfy all tests with respect to the pessimistic constraints, disregarding common paths. Only a small set of violations can be checked by a more accurate common path analysis. However, in clock skew optimization in Chapter 5 we will optimize paths, instead of endpoints for the most timing critical parts of a chip, to obtain a more robust sign-off timing.

2.6 Timing Closure Problem

Having defined placement, routing and timing constraints, we can introduce the Timing Closure Problem.

Timing Closure Problem

Input:

- A chip image $\mathcal{I}$,
- a netlist $(\mathcal{C}, P, \gamma, \mathcal{N})$,
- a library $\mathcal{B}$ and an initial circuit assignment $\beta : \mathcal{C} \to \mathcal{B}$,
- timing constraints.

Output:

- A logically equivalent netlist $(\mathcal{C}', P', \gamma', \mathcal{N}')$ according to Section 2.3.1,
- a legal placement $\mathrm{Pl} : \mathcal{C}' \to \mathbb{R}^2$ according to Section 2.3.2,

such that all late mode timing constraints are met according to Section 2.4, and the design is routable according to Section 2.3.3.

The Timing Closure Problem contains many *NP*-hard subproblems, like the minimum rectilinear Steiner tree problem, facility location problems, the feedback-arc set problem to name a few. In addition it combines difficult discrete problems with non-linear and non-convex timing constraints.

Furthermore, we are dealing with very large instances. The instance sizes are in the range of $|\mathcal{C}| \approx |\mathcal{N}| \approx 6\,000\,000$ and $|P| \approx 18\,000\,000$, and increasing with each new technology generation.

By these reasons, the Timing Closure Problem is usually decomposed into smaller subtasks, which are solved sequentially and combined into a timing optimization flow.

An optimization goal of these subproblems is to maximize the worst slack within their scope. However, in order to reduce the effort for subsequent steps and the robustness of the chip, less critical negative slacks should be maximized as well. A good slack distribution can be characterized by the lexicographical order of the sorted slacks. We adapt the definition of a *leximin maximal* vector from Radunović and Le Boudec [2007]:

Definition 2.6. *Given $n_{\max} \in \mathbb{N}$, a set of finite-dimensional vectors*

$$\mathcal{X} \subseteq \bigcup_{n=1}^{n_{\max}} \mathbb{R}^n,$$

a vector $x \in \mathcal{X} \cap \mathbb{R}^n$, $1 \leq n \leq n_{\max}$, and a threshold $\mathrm{S}^{\mathrm{tgt}} \in \mathbb{R} \cup \{\infty\}$, let $\tilde{x} \in \mathbb{R}^{n_{\max}}$ result from x by

$$\tilde{x}_i = \begin{cases} \min\{x_i, \mathrm{S}^{\mathrm{tgt}}\} & \text{if } 1 \leq i \leq n, \\ \mathrm{S}^{\mathrm{tgt}} & \text{if } n < i \leq n_{\max}, \end{cases}$$

The vector $\overrightarrow{x} \in \mathbb{R}^{n_{\max}}$ arises from $\tilde{x}$ by reordering the components in non-decreasing order. A vector $x \in \mathcal{X}$ is **leximin maximal with respect to** $\mathrm{S}^{\mathrm{tgt}}$ *if for every vector $y \in \mathcal{X}$, $\overrightarrow{x}$ is lexicographically greater than or equal to $\overrightarrow{y}$.*

This definition quantifies an optimum slack-distribution of slacks below some target $\mathrm{S}^{\mathrm{tgt}} \in \mathbb{R}$, but has no further requirement on uncritical slacks above $\mathrm{S}^{\mathrm{tgt}}$. This allows optimization for power on uncritical parts of the chip as long as the slacks remain above $\mathrm{S}^{\mathrm{tgt}}$.

2.7 Test Data

We will carry out comprehensive experiments within each chapter on a set of instances, for which the TIMING CLOSURE PROBLEM proved to be difficult. The instances are recent ASIC designs from IBM. ASIC is the abbreviation of *application specific integrated circuit*, and stands for integrated circuits that are optimized for special purposes, such as main board controllers, network switch chips, graphic cards, etc.

Chip	**Technology**	**#Circuits**		**Frequency**
		Without Repeaters	With Repeaters	
Fazil	130 nm	55 K	61 K	200 MHz
Felix	130 nm	63 K	73 K	500 MHz
Julia	130 nm	183 K	202 K	200 MHz
Bert	130 nm	911 K	1043 K	1000 MHz
Karsten	130 nm	2776 K	3146 K	800 MHz
Trips	130 nm	5169 K	5733 K	267 MHz
Franz	90 nm	63 K	67 K	209 MHz
Arijan	90 nm	3288 K	3853 K	1334 MHz
David	90 nm	3658 K	4224 K	1250 MHz
Valentin	90 nm	4136 K	5404 K	1250 MHz
Lucius	65 nm	56 K	81 K	371 MHz
Minyi	65 nm	197 K	283 K	657 MHz
Maxim	65 nm	318 K	481 K	1000 MHz
Tara	65 nm	716 K	795 K	250 MHz
Ludwig	65 nm	3108 K	3490 K	250 MHz

Table 2.1: Test Instances

The instances were implemented in three current technologies, namely 65 nm, 90 nm, and 130 nm. Table 2.1 shows all instances by chip name, technology, number of circuits, and the maximum occurring clock frequency. As the number of circuits varies throughout the timing closure flow, we specify two numbers. The column "Without Repeaters" contains the number of circuits in the input netlist without buffers and a minimum number of inverters, which are required for parity reasons. The column "With Repeaters" contains the number of circuits at the end of the timing closure flow including all inserted repeaters. The instances range from small RLMs with less that 100 000 circuits to huge ASICs with multi-million circuits. Though being small, the selected RLMs are hard to implement, as they often represent the processing cores of larger chips, and have large combinatorial cones. The additional difficulty of larger ASICs are the long distances signals have to cover and of course the computational effort.

We thank IBM for providing the test cases. We also thank the University of Texas at Austin for providing us the Trips design data, see Burger et al. [2004] for information about that multi-core processor.

3 Repeater Trees

One of the key tasks in timing optimization is the construction of repeater trees. As shown in Section 2.4.3, the interconnect RC-delay of an unbuffered net grows quadratically in the length with respect to the Elmore delay metric and almost quadratically with respect to more accurate delay models. In order to control this delay increase, repeaters (inverters or buffers) that refresh the electrical signals are used to linearize the delay as a function of length and to keep it small over large distances. With shrinking feature sizes, the fraction of the total circuit count represented by repeaters that have to be added to a design in order to buffer interconnect is bound to grow. Saxena et al. [2003] try to quantify this increase for typical designs and predict that at the 45nm and 32nm technology nodes up to 35% and alarming 70% of all circuits might be repeaters. In this situation the efficient construction of high quality repeater trees has vital importance for timing closure.

In this Chapter we focus on the global topology generation of an interconnect. The subsequent repeater insertion combined with local modifications to the global topology will be described in Section 3.7.

We motivate the topology generation problem with a small example. Temporarily, consider a topology as a Steiner tree connecting a source pin with a set of sink pins. Finding a topology that minimizes the l_1-netlength is already a *NP*-hard task (Garey and Johnson [1977]). But a shortest topology can even result in impractical long delays, as it tends to form a daisy-chain, where the path-delay from the source to the last sink in the chain is extremely high. Figure 3.1 shows an example where, on the left, a shortest topology generates a long path from source r to sink s. If

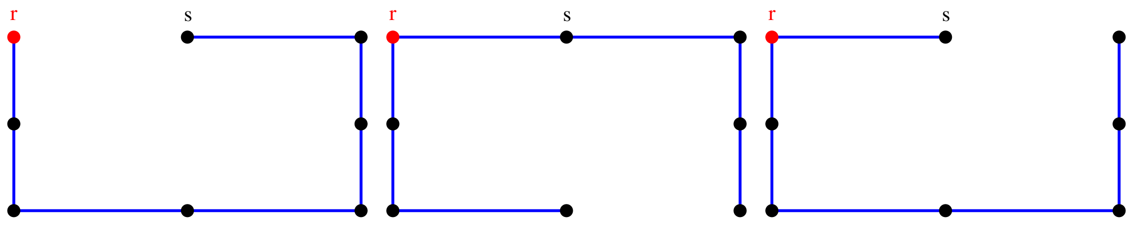


Figure 3.1: Three different minimum length topologies.

the critical path on the chip contains the edge $(r, s) \in G_p^{\mathcal{T}}$, this would be a very unfavorable topology. The topology in the middle connects s directly with r, which would be a better solution. In addition to the path lengths, the delay also increases with every branch-off and every terminal on the path as they introduce extra delays through extra capacitances. Under this considerations, the right topology would be fastest for s, because only the branch emerging in r adds extra capacitance.

A shortest Steiner tree topology of large trees with several hundred thousand sinks can easily result in source-sink delays that are 10–50 times higher than the cycle time. However, a timing aware topology must not introduce too much wiring overhead compared to a shortest topology because of limited power and wiring resources.

A further challenge is the huge number of instances that have to be solved. On today's ASIC designs there are up to 5 000 000 instances with up to 700 000 sinks.

The main feature of the new approach is that it efficiently balances the requirements of maximizing the worst slack and minimizing the wirelength. For this purpose, a new delay model on global topologies is proposed that estimates the final delays after repeater insertion.

The effectiveness is demonstrated by theoretical as well as experimental results on current industrial designs. We establish theoretical bounds on the worst slack of an optimum solution. We compare our results with respect to worst slack estimates and wirelength to theoretical bounds, proving that our solutions are indeed very good. Moreover, in contrast to many previous approaches, our algorithm is extremely fast.

Most results in this chapter were obtained in a joint work with Christoph Bartoschek, who made all repeater tree related implementations, Dieter Rautenbach and Jens Vygen (see also Bartoschek et al. [2006, 2007a,b]).

3.1 Previous Work

Most repeater tree algorithms split the task into two parts: topology generation and buffering. Some try to combine the tasks. Although we will change the topology while buffering, we also construct a preliminary topology first.

Several authors proposed heuristics for the construction of a suitable topology or tried to combine topology generation and buffering. Cong et al. [1993] proposed the A-tree algorithm, which constructs a shortest path tree targeting minimum wirelength. Okamoto and Cong [1996] proposed a repeater tree procedure using a bottom-up clustering of the sinks and a top-down buffering of the obtained topology. Similarly, Lillis et al. [1996] also integrated buffer insertion and topology generation. They considered the P-tree algorithm (P for placement), which takes the locality of sinks into account, and explored a large solution space via dynamic programming.

Hrkic and Lillis [2002, 2003] considered the S-tree algorithm (S for slack), which makes better use of timing information, and integrated timing and placement information using so called SP-trees. In these approaches the sinks are typically partitioned according to criticality. The initially given topology (for example a shortest Steiner tree) can be changed by partially separating critical and noncritical sinks. Whereas the results obtained by these procedures can be good, the running times tend to be prohibitive for realistic designs in which millions of instances have to be solved.

The C-tree algorithm (Alpert et al. [2002]) is a hybrid combination of the Prim heuristic (see Theorem 3.15) for minimum Steiner trees and Dijkstra's shortest path tree algorithm (Dijkstra [1959]). First, it computes several solutions for a small set of different tradeoffs. Finally, it chooses the solution which is ranked as the best by some performance metric.

Further approaches for the generation or appropriate modification of topologies and their buffering were considered in Cong and Yuan [2000], Alpert et al. [2001, 2003], Müller-Hannemann and Zimmermann [2003], Dechu et al. [2005], Hentschke et al. [2007], and Pan et al. [2007].

3.2 Repeater Tree Problem

A repeater tree instance is defined as follows.

REPEATER TREE INSTANCE

Input:

- a root r (an output pin of some circuit in the netlist or a primary input pin), its location $Pl(r) \in \mathcal{S}(\mathcal{I})$, a source arrival time $\text{at}(r, \zeta, \text{downcap})$ and slew $\text{slew}(r, \zeta, \text{downcap})$ at r depending on the load capacitance downcap,
- a set S of sinks s (input pins of circuits in the netlist), and for each sink s its parity $+$ or $-$, the location $Pl(s) \in \mathcal{S}(\mathcal{I})$, the input capacitance, and a slew-dependent required signal arrival time $\text{rat}(s, \zeta, \mathit{slew})$,
- a library $\mathcal{B}$ of repeaters (inverters and buffers), and for each repeater type $t \in \mathcal{B}$ its timing rule and input capacitance,
- a finite set of possible *wiring modes*, each of which consists of a routing plane (or a pair of routing planes with similar electrical properties), wire width, spacing to neighboring wires, and in particular resistance and capacitance information per unit length, and
- a shape set $\mathcal{D}(\mathcal{I})$ of blockages whose areas are unusable for placing repeaters.

A repeater tree $(\mathcal{C}_r, \text{Pl}_r, \mathcal{N}_r)$ consists of a set $\mathcal{C}_r$ of repeaters with placement locations $\text{Pl}_r(c) \in \mathcal{S}(\mathcal{I}) \setminus \mathcal{D}(\mathcal{I})$ for all $c \in \mathcal{C}_r$. It connects the root to all sinks through all circuits in $\mathcal{C}_r$ and all nets in $\mathcal{N}_r$, and the signal arrives at each sink with the correct parity, which means that the number of inversions on the path from the root to the sink is even if and only if the sink has parity $+$. Furthermore, the tree should be free of electrical violations if possible. In the definition of a repeater tree instance we ignore the case where signals with equal transition but different origins enter the root "r". We consider only the worst slack signal for each transition and ignore less critical signals.

Among the repeater trees satisfying the above conditions, one typically seeks for a tree that optimizes certain goals. One of the most important optimization goals is maximizing the worst slack, which is given by the slack at the root. If the timing restrictions allow, minimizing the use of wiring resources as well as the number (and size) of inserted repeaters are secondary goals; moreover we would like to minimize the resource allocation.

We assume that the resource allocation of a tree $T_r = (\mathcal{C}_r, \mathrm{Pl}_r, \mathcal{N}_r)$ is summarized by a single function-value

$$\rho : \{\text{Trees for } r\} \to \mathbb{R}^+.$$

Where applicable, we will be more specific on ρ. However, we assume that the resource allocation correlates well with (weighted) wirelength because the number of needed repeaters correlates with the total capacitance that is to be driven. The wire capacitance, in turn, depends on the wire length (area and fringing capacitance) and wiring congestion (coupling capacitance). During topology generation, we consider only wirelength (possibly weighted based on congestion estimates) as a second objective.

To balance these two main objectives, timing and resource allocation, we introduce a parameter $\xi \in [0,1]$, indicating how timing-critical a given instance is. For $\xi = 1$ we primarily optimize the worst slack, and for $\xi = 0$ we minimize resource allocation. Only in case of ties, we consider the most resource efficient objective value with respect to $\xi = 0$ as a tie breaker. In practice we mainly use values of ξ that are strictly between 0 and 1. (see Section 7.2).

Repeater Tree Problem

Input: A repeater tree instance (according to page 35), a performance parameter $\xi \in [0,1]$.

Output: A repeater tree $T = (\mathcal{C}_r, \mathrm{Pl}_r, \mathcal{N}_r)$ that maximizes the weighted sum

$$\xi \cdot \mathrm{slk}(r) - (1-\xi) \cdot \rho(T), \tag{3.1}$$

and in case of ties minimizes

$$\rho(T). \tag{3.2}$$

3.3 Analysis of Library and Wiring Modes

For a given technology and library we compute some auxiliary data and parameters once in advance, which are then used for all instances. The main goal is to allow for quick delay estimates when handling a particular instance.

3.3.1 Bridging Large Distances

The goal of this section is to compute a parameter d_{wire} that will act as a multiplier, to quickly estimate the delay of an optimally buffered two point connection through its length. For this purpose, we assume that we want to bridge an infinite long distance by an infinite uniform chain of inverters such that the objective function as a linear combination of delay per length unit and resource allocation per length unit is (approximately) minimized.

By a uniform chain we mean that all repeaters are mapped to the same book $t \in \mathcal{B}$, all distances between two consecutive repeaters are equal, and the wiring modes of all nets are equal. Alternatively, in the special case of inverters this configuration can be interpreted as a ring oscillator composed of a single inverter and a certain wire of certain length and wiring mode connecting its output with its input. At its eigenfrequency the oscillator has a certain circulation delay and resource allocation from which the per length unit values are derived.

For a repeater of type $t \in \mathcal{B}$, with input transition ζ and slew value s_{in}, driving a wire of length l of wiring mode m with an additional load capacitance c at the other end of the wire, let $\vartheta(t, \zeta, s_{in}, l, m, c)$, $\lambda_{out}(t, \zeta, s_{in}, l, m, c)$, $\rho(t, \zeta, s_{in}, l, m, c)$, and $\mathrm{tr}(\zeta)$ be the total delay (through the repeater and wire), the slew at the other end of the wire, the resource allocation, and the output transition (which is the input transition for the next stage), respectively. For the sake of a simpler notation we assume that all values are infinite if a load limit or slew limit is violated.

In the notation of Section 2.4, let $e = (\mathrm{late}, \zeta, \mathrm{tr}(\zeta)) \in G_{\mathcal{T}_t}$ be the corresponding model graph edge. Then the delay and slew function would be

$$\begin{aligned}\vartheta(t, \zeta, s_{in}, l, m, c) = {} & \tilde{\vartheta}_e(\mathrm{wirecap}(l, m) + c, s_{in}) + \\ & \left(\vartheta_{\mathrm{Elmore}}\left(\tilde{\lambda}_e(\mathrm{wirecap}(l, m) + c, s_{in})\right) \cdot res(l, m)\left(\tfrac{\mathrm{wirecap}(l,m)}{2} + c\right)\right),\end{aligned} \tag{3.3}$$

and

$$\begin{aligned}s(t, \zeta, s_{in}, l, m, c) = {} & \tilde{\lambda}_e(\mathrm{wirecap}(l, m) + c, s_{in}) + \\ & \left(\lambda_{\mathrm{Elmore}}\left(\tilde{\lambda}_e(\mathrm{wirecap}(l, m) + c, s_{in})\right) \cdot res(l, m)\left(\tfrac{\mathrm{wirecap}(l,m)}{2} + c\right)\right),\end{aligned} \tag{3.4}$$

where $\mathrm{wirecap}(l, m)$ and $\mathrm{wireres}(l, m)$ are the wire capacitance and resistance of a wire of length l and wiring mode m.

Recall that $\tilde{\vartheta}_e$ and $\tilde{\lambda}_e$ are the timing rules for e. If the currently considered repeater t is an inverter, let s_0 be any reasonable slew value of a rising transition $\zeta_0 = \mathrm{rise}$, and we define recursively

$$s_{i+1} := \lambda_{out}(t, \zeta_i, s_i, l, m, c) \qquad (i \geq 0).$$

Otherwise, if t is a buffer, let s_0, s_1 be reasonable slew values of a rising transition $\zeta_0 = \mathrm{rise}$ and falling transition $\zeta_1 = \mathrm{fall}$, and define the sequence $(s_i)_{i \in \mathbb{N}_0}$ by

$$s_{i+2} := \lambda_{out}(t, \zeta_i, s_i, l, m, c) \qquad (i \geq 0).$$

The sequences $(s_i)_{i=0,2,4,6,\ldots}$ and $(s_i)_{i=1,3,5,\ldots}$ of slews refering to equal transitions, typically converge very fast. We call $s'_\infty(t,l,m,c) := \lim_{i\to\infty} s_{2i-1}$ and $s''_\infty(t,l,m,c) := \lim_{i\to\infty} s_{2i}$ the *stationary slews* of (t,l,m,c). In practice it suffices to consider a chain of $N = 10$ repeater pairs and approximate $s'_\infty(t,l,m,c)$ by s_{2N-1} and $s''_\infty(t,l,m,c)$ by s_{2N}. Note that the absolute values of the two stationary slews can be quite different due to asymmetric behavior of rising and falling transition. We abbreviate

$$\begin{aligned}\vartheta(t,l,m,c) := \frac{1}{2}\Big[&\vartheta(t,s'_\infty(t,l,m,c),l,m,c)+\\ &\vartheta(t,s''_\infty(t,l,m,c),l,m,c)\Big],\end{aligned}$$

$$\begin{aligned}\rho(t,l,m,c) := \frac{1}{2}\Big[&\rho(t,s'_\infty(t,l,m,c),l,m,c)+\\ &\rho(t,s''_\infty(t,l,m,c),l,m,c)\Big],\end{aligned}$$

and

$$s_\infty(t,l,m,c) := \frac{1}{2}\Big[\,|s'_\infty(t,l,m,c)| + |s''_\infty(t,l,m,c)|\,\Big].$$

We choose a wiring mode m^*, a repeater $t^* \in L$, and a length $l^* \in \mathbb{R}_{>0}$ which minimizes the linear combination

$$\xi\bar{\vartheta}(t^*,l^*,m^*) + (1-\xi)\bar{\rho}(t^*,l^*,m^*) \tag{3.5}$$

of estimated delay per unit distance

$$\bar{\vartheta}(t^*,l^*,m^*) := \frac{\vartheta(t^*,l^*,m^*,icap(t^*))}{l^*}$$

and estimated resource allocation per unit distance

$$\bar{\rho}(t^*,l^*,m^*) := \frac{\rho(t^*,l^*,m^*,icap(t^*))}{l^*}.$$

This corresponds to optimal long chains of equidistant inverters or buffers of the same type. The minima can be found easily by binary search over all lengths for each wiring mode and each repeater type. We denote the resulting delay per unit distance by

$$d_{wire} := \bar{\vartheta}(t^*,l^*,m^*). \tag{3.6}$$

This parameter will be used for delay estimation during topology generation.

3.3.2 Further Preprocessing

We use the average stationary slews at the input pins of our optimal repeater chain as slew targets that we want to achieve during buffering. Therefore, we set:

$$\text{optslew}_{\text{rise}} := s''_\infty(t^*,l^*,m^*,icap(t^*)), \text{ and} \tag{3.7}$$

$$\text{optslew}_{\text{fall}} := s'_\infty(t^*,l^*,m^*,icap(t^*)). \tag{3.8}$$

Next, we compute a parameter d_{node} that is used during topology generation to model the expected extra delay along a path due to the existence of a side branch. It is determined as follows. Let $inv(c, s)$ denote the inverter with the smallest resource allocation that achieves a fall-slew of at most s at its output pin if its input rise-slew is $s''_\infty(t^\star, l^\star, m^\star, icap(t^\star))$ and the load is c. Let $t_1 := inv(maxcap, \text{optslew}_{\text{rise}})$, and let $t_2 := inv(icap(t_1), \text{optslew}_{\text{rise}})$. Note that $t_1 = t^\star$ if $t^\star$ is an inverter. We modify our inverter chain of length $2N$ by adding a capacitance load of $icap(t_2)$ in the middle of the first wire. Then d_{node} is set to the additional total delay through the inverter chain caused by this change.

Finally we compute the partial derivative $\frac{\partial \vartheta(t^*, l^*, m^*, icap(t^*))}{\partial icap(t^*)}$ and set

$$\text{capdelay}(c) := (c - icap(t^*))\frac{\partial \vartheta(t^*, l^*, m^*, icap(t^*))}{\partial icap(t^*)}.$$

Note that $\text{capdelay}(c) \geq 0$ if and only if $c \geq icap(t^*)$.

3.4 Topology Generation

The topology specifies the abstract geometric structure of the repeater tree.

Definition 3.1. *A (repeater tree)* topology*, for a repeater tree instance with root r, is an arborescence T (rooted at r), where r has exactly one child and all internal nodes have two children. All internal nodes $u \in V(T)$ are assigned placement coordinates $Pl(u) \in \mathcal{S}(\mathcal{I})$.*

The final topology will have S as the set of leaves. Nodes of the topology may have the same position, which can even be necessary to represent shortest Steiner trees. The reason for restricting the maximum out-degree to two is related to the delay model, which should account for branches.

3.4.1 Delay Model for Topology Generation

We estimate the delay caused by the actual wires and repeaters in the following way. For an arc $e = (u, v)$ of the topology, which represents an appropriately buffered point-to-point connection between the locations of u and v, we write

$$\vartheta_e = d_{wire} \cdot dist(Pl(u), Pl(v)).$$

This is justified as appropriate buffering linearizes the delay. The distance is usually the ℓ_1-distance, that is, $dist(Pl(u), Pl(v)) = ||\,\text{Pl}(u) - \text{Pl}(v)||_1$. However, different metrics can be used (cf. Section 3.8.1 and 3.8.2) to account for blockages and congestion.

Similarly, every internal node in the topology represents a bifurcation and thus an additional capacitance load for the circuit driving each of the two emanating branches (compared to alternative direct connections). This additional delay depends on the

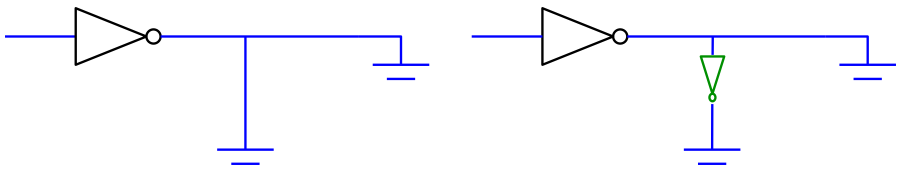


Figure 3.2: Load capacitance reduction by inserting a shielding inverter.

load at the other branch and can hardly be estimated beforehand. However, a less critical branch with a high load can be shielded by inserting a small repeater with a small input pin capacitance directly behind the branching. Figure 3.2 shows a net with two branches. The green inverter shields the capacitance in the bottom from the horizontal wire segment and thus the capacitance that the vertical branch adds to the horizontal one. This shielding repeater can increase the delay on the less critical branch. In fact, during repeater insertion there will be some room to balance the delays of the two branches by inserting one, or two shielding repeaters.

In our delay model we account for this, by demanding the distributing of an additional amount of delay d_{node} (cf. Section 3.3.2) to the two emanating branches. We denote by $\vartheta_{node}(e)$ the amount assigned to arc e; we require

$$\begin{aligned} \vartheta_{node}(e) &\geq \lambda\, \vartheta_{node} && \text{for all arcs } e, \text{ and} \\ \vartheta_{node}(e) + \vartheta_{node}(e') &= d_{node} && \text{for the two arcs } e, e' \in \delta^+(v) \end{aligned} \tag{3.9}$$

leaving the same internal node v. The parameter λ must lie between 0 and $\frac{1}{2}$. A value of $\lambda = 0$ corresponds to the possibility to insert an ideal shielding repeater with zero input pin capacitance. In the other extreme case, $\lambda = \frac{1}{2}$, there is no freedom to balance the delay ϑ_{node}. For the arc e leaving the root we always set $\delta_{node}(e) := 0$.

Next we take into account that the root may be weaker or stronger than the repeater t^* that we assume in our delay model. We compute the optimal average delay that we get if the root drives only a repeater of type t^* at distance l^* with a wire of mode m^*. The average is taken over both transitions. If the root is itself a repeater of type t^*, this delay is $\delta_{opt} := \vartheta(t^*, l^*, m^*, icap(t^*))$ if the input slews of the root are $\text{optslew}_{\text{rise}}$ and $\text{optslew}_{\text{fall}}$, as discussed above. Otherwise it may be useful to insert a buffer or two inverters. If d_r is the minimum delay we can obtain in this way, we set

$$\Delta\, \delta_r := \delta_r - \delta_{opt}\,.$$

Note that this number can be positive, zero, or negative.

To account for different delays caused by different input capacitances of the sinks, we add $\text{capdelay}(icap(s))$ (see Section 3.3.2) to the estimated delay from r to s. A linear delay model has the disadvantage of neglecting the internal delay of the root circuit. Thus, it is too optimistic for short connections. To compensate for this, we add the term

$$\max\Big\{0, (d_{opt} - d_r + d_r') \cdot \Big(1 - \frac{dist(Pl(r), Pl(s))}{l^*}\Big)\Big\}$$

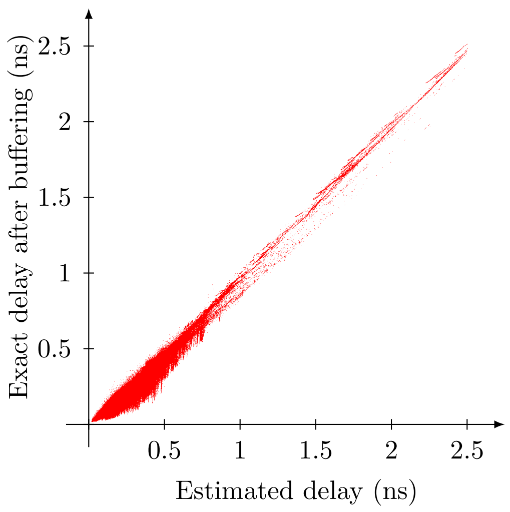


Figure 3.3: Correlation between estimated and exact delays.

to the delay estimation from r to s, where d'_r is defined as d_r, but without any distance between the root and the repeater to be driven. Let

$$\begin{aligned}\text{sinkdelay}(s) \;:=\; & \text{capdelay}(icap(s)) + \\ & \max\left\{0, (d_{opt} - d_r + d'_r) \cdot \left(1 - \frac{dist(Pl(r), Pl(s))}{l*}\right)\right\}.\end{aligned}$$

Altogether, during topology generation the delay from the root r to some sink s is modeled as

$$\begin{aligned}\sum_{e=(u,v)\in E(T_{[r,s]})} & \Big(d_{node}(e) + d_{wire} \cdot dist(Pl(u), Pl(v))\Big) \\ & + \text{rootdelay} + \text{sinkdelay}(s),\end{aligned} \tag{3.10}$$

where $E(T_{[r,s]})$ denotes the set of arcs on the path from r to s in the topology T. Note that the actual topology influences only the first part

$$\sum_{e=(u,v)\in E(T_{[r,s]})} \Big(d_{node}(e) + d_{wire} \cdot dist(Pl(u), Pl(v))\Big),$$

which is therefore the focus of the topology generation. In contrast, the residual term (rootdelay + sinkdelay(s)) is equal for any topology.

The simple delay model used by our topology generation is justified by the good correlation between our estimated delays and the final delays of the critical paths in repeater trees after buffering and sizing (Figure 3.3). Here the final delays were measured by static timing analysis with slew propagation.

3.4.2 Priority Ordering

Our topology generation inserts the sinks into the topology one by one. The resulting structure depends on the order in which the sinks are considered. In order to quantify correctly how critical a sink s is, it is crucial to take its required arrival times as well as its location $Pl(s)$ into account. We restrict the slew dependent required arrival time to a single value, assuming that the output slews equal the slews of our optimum repeater chain. We set

$$\mathrm{rat}_s := \min\Big\{\mathrm{rat}(s, \zeta, \mathrm{optslew}_\zeta) \mid \zeta \in \{\mathrm{rise}, \mathrm{fall}\}\Big\}.$$

Now, a good measure for the criticality of a sink s is the slack that would result from connecting s optimally to r and disregarding all other sinks. Within our delay model this slack equals

$$\begin{aligned}\mathrm{slk}_s := \mathrm{rat}_s - \mathrm{at}_r &- d_{wire} \cdot dist(Pl(r), Pl(s)) \\ &- \mathrm{sinkdelay}(s) - \mathrm{rootdelay}\,.\end{aligned} \tag{3.11}$$

The smaller this number is, the more critical we will consider the sink to be. However, we do not distinguish between very large values of these estimated slacks and give such sinks the same priority. The sinks are inserted into the topology in an order of non-increasing criticality.

3.4.3 Topology Generation Algorithm

At any time during the topology generation algorithm (Algorithm 1) we maintain a partial topology T' whose set of leaves is a subset S' of S. Moreover, we maintain a required arrival rat_v time for each node $v \in V(T')$ with the property

$$\mathrm{rat}_v = \max_{\lambda d_{node} \leq d \leq (1-\lambda) d_{node}} \min \Big\{ \mathrm{rat}_w - d, \mathrm{rat}_{w'} - (d_{node} - d) \mid w, w' \in \delta^+(v) \Big\}\,, \tag{3.12}$$

if $v \in V(T') \setminus \{r\}$, and $\mathrm{rat}_r = \mathrm{rat}_w$, where (r, w) is the single edge in $\delta^+(r)$. This way, we implicitly maintain the numbers $d_{node}(e)$ for each $e \in E(T')$ with the properties:

- $d_{node}(e) = 0$ for the arc e leaving the root r;
- $d_{node}(e) \geq \lambda d_{node}$ for all other arcs e for some fixed $\lambda \in \left[0, \frac{1}{2}\right]$.
- $d_{node}(e) + d_{node}(e') = d_{node}$ if the two arcs e and e' leave the same node.

The sinks $S = \{s_1, s_2, \dots, s_n\}$ are inserted into the topology one by one in the order of their priority, $\mathrm{slk}_{s1} \leq \mathrm{slk}_{s_2} \leq \dots \leq \mathrm{slk}_{s_n}$, as computed in Section 3.4.2. If there are ties, we choose a sink that is closest to the root.

Initially, S' contains just one most critical sink s_1, and the topology consists of the arc from r to s_1. Suppose now $s \in S \setminus S'$ is the next sink to be inserted into T'. To decide where to insert s, we consider each arc $e = (u, w) \in E(T')$ for an insertion of a new node v and edge (v, s).

Given the positions $Pl(u)$ and $Pl(w)$, we determine the closest point $Pl(v)$ to $Pl(s)$ within the set $SP(u, w)$ of locations, which are covered by a shortest path between $Pl(u)$ and $Pl(w)$. Using the l_1-metric, $SP(u, w)$ is simply the area within

Algorithm 1 Topology Generation Algorithm

```
 1: For each $s \in S$, compute $\mathrm{slk}_s$ according to (3.11);
 2: For each $s \in S$: $\mathrm{rat}_s := \mathrm{rat}_s - \mathrm{sinkdelay}(s)$;
 3: Sort $S$ by non-decreasing slack bound: $\mathrm{slk}_{s_1} \le \mathrm{slk}_{s_2} \le \cdots \le \mathrm{slk}_{s_n}$;
 4: $T' := (\{r, s_1\}, \{(r, s_1)\}); S := S \setminus \{s_1\}$;
 5: for $i = 2, \dots, n$ do
 6:     Create a new internal node $v$;
 7:     $V(T') := V(T') \cup \{s_i\} \cup \{v\}$;
 8:     $(u^\star, w^\star) = \emptyset$
 9:     $best_1 = -\infty$;
10:     $best_2 = -\infty$;

        /* Find best edge in $E(T')$ for attaching $s_i$.*/
11:     for $(u, w) \in E(T')$ do
12:         $E(T') := E(T') \setminus \{(u, w)\} \cup \{(u, v)\} \cup \{(v, w)\} \cup \{(v, s_i)\}$;
13:         $\Delta w := dist(\mathrm{Pl}(s), SP(u, w))$;

        /* Compute root slack.*/
14:         for $(x, y) \in E(P_{r,s_i}(T')) \setminus \delta^+(r)$ traversed bottom-up do
15:             Let $y'$ be the sibling of $y$;
16:             $\rho_y := \mathrm{rat}_y - d_{wire} \cdot dist(Pl(x), Pl(y))$;
17:             $\rho_{y'} := \mathrm{rat}_{y'} - d_{wire} \cdot dist(Pl(x), Pl(y'))$;
18:             $\mathrm{rat}_x := \max_{\lambda d_{node} \le d \le (1-\lambda) d_{node}} \min \{\rho_y - d, \rho_{y'} - (d_{node} - d)\}$ ;
19:             If $\mathrm{rat}_x$ was not changed: Goto 21;
20:         end for
21:         $\mathrm{rat}_r := \mathrm{rat}_w$, where $\{(r, w)\} = \delta^+(r)$;

        /* Maximize 1st and 2nd objective lexicographically.*/
22:         if $(\xi(\mathrm{rat}_r - at_r) + (\xi - 1) d_{wire} \cdot \Delta w, -\Delta w) >_{lex} (best_1, best_2)$ then
23:             $(u^\star, w^\star) := (u, w)$;
24:             $best_1 := \xi(\mathrm{rat}_r - \mathrm{at}_r) + (\xi - 1) d_{wire} \cdot \Delta w$;
25:             $best_2 := -\Delta w$;
26:         end if
27:         Revert changes from lines 12 and 18;
28:     end for

        /* Finally attach $s_i$ to $(u^\star, w^\star)$ and update $\mathrm{rat}_v$ $(v \in V(T'))$.*/
29:     $E(T') := E(T') \setminus \{(u^\star, w^\star)\} \cup \{(u^\star, v)\} \cup \{(v, w^\star)\} \cup \{(v, s_i)\}$;
30:     for $(x, y) \in P_{r,s_i}(T')$ traversed bottom-up do
31:         Let $y'$ be the sibling of $y$;
32:         $\rho_y := \mathrm{rat}_y - d_{wire} \cdot dist(Pl(x), Pl(y))$;
33:         $\rho_{y'} := \mathrm{rat}_{y'} - d_{wire} \cdot dist(Pl(x), Pl(y'))$;
34:         $\mathrm{rat}_x := \max_{\lambda d_{node} \le d \le (1-\lambda) d_{node}} \min \{\rho_y - d, \rho_{y'} - (d_{node} - d)\}$;
35:     end for
36: end for
```

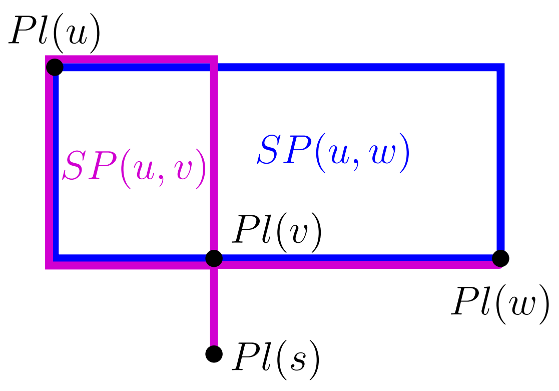


Figure 3.4: Insertion of the next sink s into an edge (u,w)

the bounding box of $Pl(u)$ and $Pl(w)$. At this point we tentatively place the new internal node v and replace the arc (u,w) by three new arcs (u,v), (v,w), and (v,s).

Figure 3.4 demonstrates this situation. The sink s is connected shortest possible to $SP(u,w)$ (blue box). The magenta edges and box indicate the resulting shortest path areas $SP(u,v)$, $SP(v,w)$, and $SP(v,s)$, where $SP(v,w)$ and $SP(v,s)$ are straight lines.

Having inserted s into an edge, we optimize the required arrival times rat_v of all nodes $v \in V(T')$ on the path from s to the root such that the resulting slack $(\text{rat}_r - \text{at}_r)$ at the root, according to our delay model, is maximized, and such that our rules regarding d_{node} are not violated. Note that there is no need for considering any other node; all required arrival times can be updated by scanning the path bottom-up to the root until there are no further changes.

We can now estimate the worst slack slk_r at the root as well as the additional wire length Δw, which would result from this change. Finally, we insert s into an arc $e = (u,v)$, for which the objectives (i) $\xi(\text{rat}_r - \text{at}_r) + (\xi - 1)d_{wire}\Delta w$ and (ii) $-\Delta w$ are maximized lexicographically. Before investigating the quality of the resulting topology in the next section, we determine its running time:

Theorem 3.2. *The worst case running time of the topology generation algorithm (Algorithm 1) is $O(n^3 + n^2\mathrm{T}_{\mathrm{SP}})$, where $n = |S|$ and T_{SP} is the running time for finding a shortest path between a point and an area, which is covered by all shortest paths between two points, in a given metric. The best case running time is $\Omega(n^2 + n^2\mathrm{T}_{\mathrm{SP}})$.*

Proof. There are exactly $n-1$ outer loop iterations (lines 5–36). When the sink s_i is considered for insertion, the tree has $2(i-1)-1$ edges (spanning r, $i-1$ leaves, and $i-2$ internal nodes). Thus, in total, there are $\sum_{i=2}^{n}(2(i-1)-1) = \Theta(n^2)$ updates (lines 11–28). In the worst case, where the topology has depth $O(n)$, there are at most $O(n^3)$ runs through lines 14–18, in total. Together with $\Theta(n^2)$ shortest path searches this give the worst case running time of $O(n^3 + n^2\mathrm{T}_{\mathrm{SP}})$.

In the best case no value rat_v of any node v on the path from s_i to r changes, and the most inner loop is stopped immediately in line 19. Therefore, we obtain the claimed best case running time.

□

If distances are computed with respect to the l_1-metric, the shortest path between $Pl(s)$ and $SP(u, w)$ can be determined in constant time, yielding overall running time bounds of $O(n^3)$ and $\Omega(n^2)$.

In practice the worst case occurs hardly, because the sinks are inserted with decreasing criticality, which makes it unlikely that required arrival times have to be updated on the complete paths up to the root. Moreover, the data that has to be manipulated and the calculations that are executed are all very simple and efficient. Indeed, our experiments will show that the running time of our algorithm is extremely small.

Nevertheless, for very large repeater tree instances, we reduce the running time by a pre-clustering approach from Maßberg and Vygen [2005, 2008] that runs in $O(|S| \log |S|)$ time. Furthermore, in the presence of blockages, the distances and shortest path areas have to be specified. These and further implementation issues are discussed in more detail in Section 3.8.

3.5 Theoretical Properties

In this section we explain how to compute efficiently an estimate for the maximum achievable slack within our delay model. Furthermore, we establish some optimality statements for the generated topology in the two extremal cases corresponding to maximizing worst slack ($\xi = 1$) and minimizing wirelength ($\xi = 0$).

3.5.1 Maximum Achievable Slack

Lemma 3.3. *There is always a slack-optimal topology such that all internal nodes are at the same position as the root, that is, every sink is connected individually to a node at the position of the root by a shortest connection.*

Proof. Let T be a slack optimum topology, let $v \in V(T)$ be an internal node with parent $u \in V(T)$ and $Pl(u) \neq Pl(v)$. Let $w_1, w_2 \in V(T)$ be the two children of v. Then we can change the placement of v to $Pl(u)$ without changing the wire delay, and thus, without changing the total delay to w_1 and w_2. The lemma follows by iterative application of this operation.

□

The transformation used in the above proof introduces two parallel wires between the old and new position of v to each of its children. Thus, numerous implementations of such slack-optimum topologies would inhibit the routability of a chip in practice.

In order to estimate the maximum achievable slack by any topology within our delay model we first define a λ-tree:

Definition 3.4. *Given a number $\lambda \in \left[0, \frac{1}{2}\right]$, a λ-tree is a pair (T, d), where $T = (V, E)$ is a rooted binary tree and*

$$d : E \to \mathbb{R}_{\geq 0}$$

such that the two different arcs $e, e' \in E$, leaving a common node, satisfy

$$\begin{aligned} d(e) + d(e') &= 1 \text{ and} \\ \min\{d(e), d(e')\} &\geq \lambda. \end{aligned}$$

Definition 3.5. *For $\lambda \in \left[0, \frac{1}{2}\right]$ and $D \in \mathbb{R}$, we denote by $f(D)$ the maximum number of leaves in a λ-tree (T, d) in which the depth of every leaf with respect to d is at most D.*

Lemma 3.6. *Given $\lambda \in \left(0, \frac{1}{2}\right]$, the maximum number of leaves $f(D)$ in a λ-tree satisfies the recursion*

$$f(D) = \begin{cases} 0, & \text{if } D < 0, \\ 1, & \text{if } 0 \leq D < \frac{1}{2}, \\ \max\limits_{\lambda \leq x \leq 1-\lambda} f(D - x) + f(D - (1 - x)), & \text{if } D \geq \frac{1}{2}. \end{cases} \tag{3.13}$$

Proof. The first two cases are clear. The “divide and conquer” recursion in the third case takes the maximum over all possibilities to attach two optimum λ-subtrees, with smaller depths $D - x$ and $D - (1 - x)$, to the root of the new λ-tree.

□

If $\lambda = 0$, the recursion formula (3.13) would be undefined, as $f(D) = \infty$ for $D \geq 1$. For $\lambda = 0$ and $D \geq 1$, we can construct trees with arbitrary many leaves. The construction starts with a path P with $d_{node}(e) = 0$ for all $e \in E(P)$. Now, a second edge e' with $d_{node}(e') = 1$ emanates from each internal node. The length of P and, therefore, the number of leaves can be chosen arbitrarily. Eventually, we have $f(D) = \infty$ for $\lambda = 0$ and $D \geq 1$. By the same construction, we can easily achieve a worst slack of $\min\{\text{slk}_s + d_{node} \mid s \in S\}$, regardless of the number of sinks $|S|$.

We will apply Kraft's inequality, originating from coding theory, which characterizes, whether an ordinary binary tree with unit edge length and prescribed maximum depths of the leaves can be realized:

Theorem 3.7 (Kraft [1949])**.**
There exists an ordinary binary tree with unit edge lengths and n leaves at given depths $l_1, l_2, \ldots, l_n$ if and only if

$$\sum_{i=1}^{n} \frac{1}{2^{l_i}} \leq 1. \tag{3.14}$$

We will also need one direction of Kraft's theorem generalized for λ-trees.

Lemma 3.8. *Given $S = \{1, 2, \ldots, n\}$ and $l_1 \le l_2 \le \cdots \le l_n$ and a λ-tree T with n leaves at depths at most $l_1, l_2, \ldots, l_n$, then*

$$\sum_{i=1}^{n} f(l_n - l_i) \le f(l_n). \tag{3.15}$$

Proof. Starting from T, we create a tree T' of depth at most l_n in the following way. To each leave $i \in S$, attach a λ-subtree of depth $(l_n - l_i)$ and a maximum number of leaves $f(l_n - l_i)$ with root i. Summation yields the number of leaves of T' on the left side of (3.15), which is certainly at most $f(l_n)$.

□

Multiplying both sides in (3.14) by 2^n, it can be seen that (3.14) is equivalent to (3.15) with $f(x) = 2^x$.

The lemmata above help us to derive an upper bound for the maximum achievable worst slack of any topology:

Theorem 3.9. *Given an instance of the repeater tree problem, let $\overline{\text{slk}}_s$ be the upper bound for the slack of sink $s \in S$, as defined in Section 3.4.2, and $\overline{\text{slk}} = \max\{\text{slk}_s \mid s \in S\}$ be the maximum of these bounds. Furthermore, let d_{node} be defined as in Section 3.4.1 and let $\lambda \in \left(0, \frac{1}{2}\right]$, the maximum achievable worst slack slk_{opt} of a topology is at most the largest value slk that satisfies*

$$\sum_{s \in S} f\left(\frac{\overline{\text{slk}} - \text{slk}_s}{d_{node}}\right) \le f\left(\frac{\overline{\text{slk}} - \text{slk}}{d_{node}}\right). \tag{3.16}$$

Proof. As $d_{node}(e) = 0$ for the arc e leaving the root of the topology, the root r and its only child can be considered as one root-node of a binary λ-tree. Therefore, if the maximum achievable worst slack of a topology for this instance equals slk_{opt}, then this is equivalent to the existence of an λ-tree (T, d) whose sinks are the elements of S such that the depth l_s of sink $s \in S$ in (T, d) satisfies

$$l_s \le \frac{\text{slk}_s - \text{slk}_{opt}}{d_{node}}.$$

By (3.15) we have

$$\sum_{s \in S} f\left(\frac{\overline{\text{slk}} - \text{slk}_s}{d_{node}}\right) = \sum_{s \in S} f\left(\frac{\overline{\text{slk}} - \text{slk}_{opt}}{d_{node}} - \frac{\text{slk}_s - \text{slk}_{opt}}{d_{node}}\right) \le f\left(\frac{\overline{\text{slk}} - \text{slk}_{opt}}{d_{node}}\right).$$

□

Unfortunately, solving the recursion for f is not easy, especially as it grows exponentially. For instances with a small number of sinks $n = |S|$, (3.16) could be used to determine a good upper bound for best possible slack slk_{opt}.

To use Theorem 3.9 for instances with many sinks, we need more insight into the structure of f. It was shown that the asymptotic growth of f follows a simple exponential formula:

Theorem 3.10 (Maßberg and Rautenbach [2007]).
Given $\lambda \in \left(0, \frac{1}{2}\right]$, there are positive constants α_λ and β_λ such that the maximum number of leaves $f(D)$ in a λ-tree with depth at most D fulfills

$$\lim_{D\to\infty} \frac{f(D)}{\beta_\lambda \cdot \alpha_\lambda^D} = 1, \tag{3.17}$$

Proof. (see Maßberg and Rautenbach [2007])

□

Using this theorem we obtain an approximate upper bound for the maximum achievable worst slack:

Theorem 3.11. *Given a set S of sinks with upper slack bounds* slk_s *($s \in S$), and assuming $f(D) = \beta_\lambda \alpha_\lambda^D$ for some $\beta_\lambda, \alpha_\lambda \in \mathbb{R}_+$. Then, the maximum achievable worst slack* slk_{opt} *of a topology is at most the largest value* slk *that satisfies*

$$\sum_{s\in S} 2^{-\frac{\log_2(\alpha_\lambda)\cdot(\mathrm{slk}_s - \mathrm{slk})}{d_{node}}} \leq 1. \tag{3.18}$$

Proof. Using the explicit presentation $f(D) = \beta_\lambda \alpha_\lambda^D$, dividing by its right side, and applying $2^{\log_2(.)}$ to each summand, (3.16) transforms to (3.18).

□

Note that (3.18) conforms Kraft's Theorem 3.7, which states that with $l_s = \frac{\log_2(\alpha_\lambda)\cdot(\mathrm{slk}_s - \mathrm{slk})}{d_{node}}$, there exists an ordinary binary tree (that is, with all edge lengths equal to one) with leave set S, such that each $s \in S$ has depth at most

$$\left\lceil \frac{\log_2(\alpha_\lambda) \cdot (\mathrm{slk}_s - \mathrm{slk})}{d_{node}} \right\rceil.$$

A binary tree which fulfills (3.7) and defines the supremum for slk can easily be determined through Huffman coding in its adaptation for min-max functions by Golumbic [1976] (see Algorithm 2).

Algorithm 2 Slack Bound Computation via Huffman Coding

```
Input: A finite set $X$ of real numbers;
1: while $|X| > 1$ do
2:     $x_1 := \max\{x \in X\}$;
3:     $x_2 := \max\{x \in X \setminus \{x_1\}\}$;
4:     $X = X \setminus \{x_1, x_2\} \cup \{\min\{x_1, x_2\} - 1\}$;
5: end while
6: Return the only remaining element $x^\star \in X$;
```

The algorithm implicitly builds up a binary tree in a bottom-up order. In each iteration the elements x_1 and x_2 are connected to a new node (their father), represented by the new element inserted into X.

Theorem 3.12 (Golumbic [1976]).
Given a set $X = \{x_1, x_2, \dots, x_n\}$ of integers, Algorithm 2 returns the value

$$x^\star = -\left\lceil \log_2 \left(\sum_{i=1}^{n} 2^{-x_i} \right) \right\rceil .$$

This is the maximum possible root label $l(r) = x^\star$ achievable by a binary tree with n leaves that conforms $l(i) = x_i$ for all leaves i and $l(v) = \min\{l(w) - 1 \mid w \in \delta^+(v)\}$ for all non-leave nodes.

Proof. See Golumbic [1976], who proved the theorem for a generalized r-ary trees. □

Implementing the set X as a heap, the runtime of Algorithm 2 obviously is $O(|X| \log |X|)$. When calling Algorithm 2 with $X = \left\{ \frac{\log_2(\alpha_\lambda) \cdot \mathrm{slk}_s}{d_{node}} \mid s \in S \right\}$, $\mathrm{slk}^\star := \frac{d_{node} \cdot (x^\star + 1)}{\log_2(\alpha_\lambda)}$ serves as an approximate upper bound on slk_{opt} for large instances. Again this bound is approximate due to the assumption $f(D) = \beta_\lambda \alpha_\lambda^D$.

For the two special cases $\lambda \in \left\{ \frac{1}{2}, \frac{1}{4} \right\}$ the slack bounds can be computed efficiently.

Theorem 3.13. *If $\lambda = \frac{1}{2}$, the maximum possible slack* slk_r *is at most*

$$-\frac{d_{node}}{2} \log_2 \left(\sum_{s \in S} 2^{-\frac{2 \cdot \mathrm{slk}_s}{d_{node}}} \right). \tag{3.19}$$

If $\frac{2 \cdot \mathrm{slk}_s}{d_{node}}$ $(s \in S)$ are integral, the slightly tighter bound

$$-\frac{d_{node}}{2} \left\lceil \log_2 \left(\sum_{s \in S} 2^{-\frac{2 \cdot \mathrm{slk}_s}{d_{node}}} \right) \right\rceil \tag{3.20}$$

is realizable by a topology.

Proof. As $\lambda = \frac{1}{2}$, there is no freedom to distribute delay, but each edge is assigned a delay of $\frac{d_{node}}{2}$. Let T be a topology with

$$\mathrm{slk}_r \le \mathrm{slk}_s - \frac{d_{node}}{2} l_s \qquad (s \in S),$$

where l_s is the number of edges on the r-s-path minus one. That is, l_s is the number of edges on the r-s-path without the edge leaving r. Using Kraft's inequality (3.14) with l_s and resolving by slk_r yields the desired bound. The second inequality (3.20) follows similarly from Theorem 3.12. □

For the case $\lambda = \frac{1}{4}$, the recursion formula for f was resolved by Maßberg and Rautenbach [2007]. They obtained following result:

$$\begin{aligned} f(k) \;=\; & 0.1038\, e^{-0.7644\, k} \cos(7.4259\, k) \\ & - 0.3649\, e^{-0.7644\, k} \sin(7.4259\, k) \\ & + 0.8961\, e^{1.528\, k} \end{aligned}$$

for all $k \in \left\{\frac{i}{4} \mid i \in \mathbb{N}_0\right\}$. Therefore,

$$\alpha_{\frac{1}{4}} = \left(\left(\frac{1}{2} + \sqrt{\frac{31}{108}}\right)^{\frac{1}{3}} + \left(\frac{1}{2} - \sqrt{\frac{31}{108}}\right)^{\frac{1}{3}}\right)^{-4} \approx 4.613 \tag{3.21}$$

and $\beta_{\frac{1}{4}} \approx 0.8961$.

3.5.2 Optimality Statements

In the last section, we investigated upper bounds for the achievable slack. Here we derive quality statements of Algorithm 1 in terms of slack and wire length. As already mentioned towards the end of the previous section, the case $\lambda = \frac{1}{2}$ allows stronger statements. The following result corresponds to slack optimization for instances whose sinks are more or less close to the root.

Theorem 3.14. *For $d_{wire} = 0$, $d_{node} = 2$, $\lambda = \frac{1}{2}$, $\mathrm{at}_r, \mathrm{rat}_s \in \mathbb{N}$ and $\mathrm{rootdelay} = \mathrm{sinkdelay}(s) = 0$ for $s \in S$, the topology constructed by Algorithm 1 realizes the maximum achievable slack slk_{opt}, and we have*

$$\mathrm{slk}_{opt} = -\left\lceil \log_2\left(\sum_{s\in S} 2^{-\mathrm{rat}_s + \mathrm{at}_r}\right)\right\rceil.$$

Proof. Since all relevant quantities are integers, the stated expression for slk_{opt} follows from Theorem 3.12.

The fact that our topology realizes the optimum slack slk_{opt} follows by induction on the number of sinks. For one sink the statement is trivial. Now let $s_1, s_2, \ldots, s_n$ denote the sinks ordered such that

$$\mathrm{rat}_{s_i} \leq \mathrm{rat}_{s_j}$$

for $i < j$. By induction, the topology T' containing all but the last sink s_n realizes the maximum possible slack slk'_{opt} for these sinks. Since the procedure has the option to insert s_n using all arcs of T' leading to sinks, we can assume that all sinks are exactly at the maximum allowed depth $l_s = \mathrm{rat}_s - \mathrm{at}_r - \mathrm{slk}'_{opt}$ within T'. By Kraft's inequality (3.14), this implies that $\sum_{i=1}^{n-1} 2^{-l_s}$ equals exactly 1. Thus $\sum_{i=1}^{n} 2^{-l_s} > 1$ which implies

$$\mathrm{slk}_{opt} \leq \mathrm{slk}'_{opt} - 1$$

which will clearly be realized by Algorithm 1.

□

In the second special case, we assume that timing is uncritical and can be disregarded by setting $\xi = 0$. In this case, the priority order, as calculated in Section 3.4.2, does not give a good hint in which order the sinks should be inserted. In such a situation we can actually use further criteria to select a good order for processing non-critical sinks. For minimizing wirelength we propose to choose that sink to be inserted next, which is closest to the current partial topology. Furthermore, we choose the arc into which the new sink will be inserted as to minimize the additional wiring, ignoring slacks ($\xi = 0$).

In fact, this will turn Algorithm 1 into a Steiner tree heuristic (sometimes called Prim heuristic) similar to one considered by Rohe [2002]. Note that another situation in which the described modification of the insertion is reasonable is an early design stage in which the estimates for the arrival times are unreliable or meaningless. The following result is a well known consequence of Hwang's theorem on the Steiner ratio (Hwang [1976]).

Theorem 3.15. *If the sinks are inserted into the topology as described above, then the ℓ_1-length of the final topology is at most $\frac{3}{2}$ times the minimum ℓ_1-length of a Steiner tree on the terminals $\{r\} \cup S$.*

Proof. For a short proof see Bartoschek et al. [2006].

□

3.6 Postoptimization

One drawback of the proposed approach is that it uses a fixed pair of delay and resource allocation parameters based on a single value of ξ. The algorithm accounts for this because non-critical sinks usually do not influence the root slack and thus are connected by the tie-breaker rule that minimizes resource allocation. In addition a subsequent circuit sizing will downsize non-critical branches and thereby decrease the resource allocation in terms of placement area and power consumption. However, we can take further advantage of non-critical subtrees.

Once a good topology is found, the resource allocation might be reduced by adding more delay to non-critical paths for the sake of less resource consumption. Higher, but more resource efficient, delays can be assigned to certain edges in the topology. This includes choosing lower cost wiring modes, detours of two-point connections through less congested areas, or larger assumed repeater spacings, that will guide the subsequent repeater insertion. The problem of optimizing a fixed topology T can be formulated as follows.

For each edge $e \in E(T)$ there a is set $\mathcal{T}(e) \subset \mathbb{R}$ of alternative delays, each of which is associated with a resource allocation cost function $c_e : \mathcal{T}(e) \to \mathbb{R}_{\geq 0}$. The set $\mathcal{T}(e)$ of alternative delays will usually be composed of intervals, potentially with length zero, for example when the next different delay can only be realized by changing the wiring plane. Later in Chapter 6 we will show how to chose delays for a given slack target using minimum resources, not only for tree structures, but in general directed graphs with cycles.

3.7 Repeater Insertion

After the topology is defined repeaters have to be inserted. Here, we give only a short summary on known buffering strategies that can be applied. Most approaches are based on dynamic programming. Van Ginneken [1990] was the first to use dynamic programming. He considered the problem of inserting a single buffer type into a given topology and a fixed set of potential buffer locations. The approach works in a bottom-up fashion and computes the best solutions given all solution combinations at the children in the tree. Inferior solutions can be pruned such that the solution-set stays relatively small.

This approach has been extended and refined several times. Lillis et al. [1995] extended it to allow for several buffer types. Shi and Li [2005, 2006b, 2006a] greatly improved the originally quadratic running time of van Ginneken's algorithm. Hu et al. [2006] use the same idea to primarily optimize slew and not slack.

Some authors also considered analytic models for locating and sizing buffers. Using rather restricted delay models, closed formulas were obtained by Dhar and Franklin [1991], Chu and Wong [1997]. Alpert and Devgan [1997] also applied methods from continuous optimization, and Mo and Chu [2000] combined van Ginneken's dynamic programming paradigm with quadratic programming. Clearly, the optimality statements made about these continuous approaches are only valid within the somewhat unrealistic circuit and delay model and do not capture the discrete nature of the buffering decisions. All of these solutions have rather high running times, which prevents their application on millions of instances.

In Bartoschek et al. [2007b] we proposed a faster algorithm that finds potential repeater positions on the fly. Then it applies a fast bottom-up buffering along the previously computed topology. The optimum repeater distances in an optimum chain, as computed in Section 3.3.1, define distance limits between two repeaters. Once that limit is reached, a repeater is inserted. Whenever two branches are merged, a small predefined finite set of the promising ways to combine the branches and add shielding repeaters is enumerated. If two branches of different parities are merged, the merge is not done immediately. Instead, two parallel branches are maintained, leaving the possibility of the inverter insertion to an appropriate point. This prevents excessive inverter insertion if too many distinct parities are to be merged.

As the topology generation, the repeater insertion balances efficiently between power and performance. The parameter $\xi \in [0, 1]$ controls the repeater spacing as well as the effort for inserting shielding repeaters during the branch merging and for inserting repeaters directly behind load sensitive root pins.

Altogether we are able to solve 1.3 million timing critical instances in 20 minutes on a Xeon E7220 processor, and achieve slacks comparable to high accuracy dynamic programming approaches, which would require days for such a task.

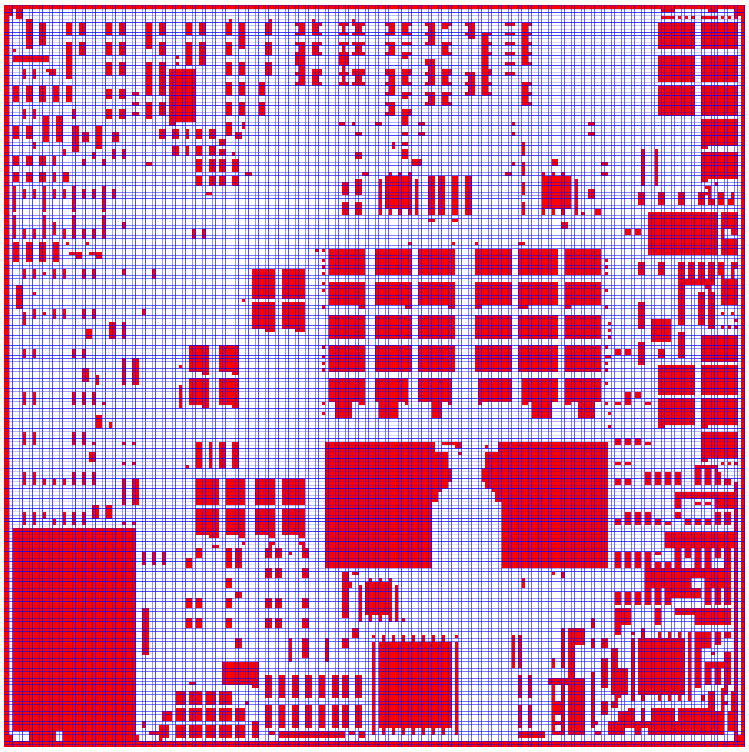


Figure 3.5: Blockages (red) on the chip Arijan with blockage grid (blue).

3.8 Implementation Issues

3.8.1 Blockages

Repeater tree algorithms have to be aware of blockages as they occur frequently in practice. In some areas it is impossible to place repeaters (placement blockages, mostly due to pre-placed macros). Some macros do not even allow any wiring across; these areas define wiring blockages. Some of these blockages can force us to make detours.

We introduce a blockage grid, which is the Hanan grid given by the coordinates of all edges of significantly large blockages (Hanan [1966]), enhanced by additional grid-edges if routing and placement congestion shall be considered. Figure 3.5 shows typical blockages on a chip as red areas and and a blockage grid by blue lines. Next we define a cost function on the edges of the blockage grid that reflects delay (similarly as Held et al. [2003]). As before, the cost of an edge is d_{wire} times its length unless the edge is in the interior of a blockage. For long distances within placement blockages we extend the linear function beyond l^* in a continuously differentiable way by a quadratic function. This models higher (and quadratically growing) delays in the case when the distance between two repeaters is larger than optimal. Edges within routing blockages have infinite cost.

Given the blockage grid with these edge costs, we can compute a shortest path between any pair of points fast with Dijkstra's algorithm (Dijkstra [1959]), and thus get a metric c on $\mathbb{R}^2$. In the absence of any relevant blockages this is d_{wire} times the ℓ_1-distance. However, to use c instead of ℓ_1-distances properly in our topology generation algorithm, we would have to compute shortest three-terminal Steiner trees with respect to c in every step, which is much more time-consuming than doing the same with respect to the ℓ_1-metric. This has not been implemented yet. However, we will re-route two-point connections based on the metric c during buffering.

3.8.2 Handling Placement and Routing Congestion

Routing congestion can be generated by repeater trees, and placement congestion (too little space for too many circuits) can also result from inserting many repeaters in a certain area. Thus it is advisable to take routing and placement congestion into account when constructing repeater trees.

Our repeater tree algorithm balances efficiently between wire length and performance. This results in an overall economic consumption of placement and wiring resources. When applied in cooperation with an efficient congestion-driven placement algorithm (like Brenner and Rohe [2003]), wiring congestion does hardly show up. Our current implementation and our experimental results do not take specific care of routing and placement congestion. Nevertheless all chips could be legalized and routed without any problems, not only in our experiments but also in the daily usage of our industrial partners.

Anyway, the algorithm can be extended to handle congestion directly, when augmenting the blockage grid by congestion information. Using the combined information we can define a metric which can be used instead of the ℓ_1-metric or c as defined in the previous subsection. Edges within congested areas become more expensive.

Estimating placement congestion is a routine job. For estimating routing congestion we suggest to use a fast method like Brenner and Rohe [2003] rather than a full global routing. Of course the information must be updated regularly if many repeater trees are built. One could even think of using our inverter tree routine as a subroutine of a multicommodity-flow-based global router, such as the one described by Müller [2006].

3.8.3 Very High Fanout Trees

As we will show in our experimental results, the running time of our algorithm is extremely small for instances up to 1 000 sinks. Nevertheless, our topology generation as described above has a cubic running time. On real designs there are some instances with several hundred thousand sinks, for which this would lead to intolerable running times.

One way to reduce the running time would be to consider only the nearest k arcs, were k is some positive integer. This would require to store the arcs as rectangles in a suitable geometric data structure (for example k-d-trees). However, we chose a different approach.

For instances with more than 1 000 sinks we first apply a clustering algorithm to all sinks, except for the 100 most critical ones if $\xi > 0$. More precisely, we find a partition $S' = S_1 \dot\cup \cdots \dot\cup S_k$ of the set S' of less critical sinks, and Steiner trees T_i for S_i $(i = 1, \dots, k)$ such that the total capacitance of T_i plus the input capacitances of S_i is at most *maxcap*. Among such solutions we try to minimize the total wire capacitance plus k times the input capacitance of the repeater t^*.

For this problem we use the approximation algorithm by Maßberg and Vygen [2008], which generates very good solutions in $O(|S| \log |S|)$ time. We introduce an appropriate inverter for each component; its input pin constitutes a new sink. This typically reduces the number of sinks by at least a factor of 10. If the number of sinks is still greater than 1 000, we iterate this clustering step. Finally we run our normal repeater tree algorithm as described above.

3.8.4 Plane Assignment and Wire Sizing

So far, we used only one wiring mode. However, the best wiring mode for bridging large distances may not be appropriate for short connections. On the other hand, if we concentrate on minimizing the use of routing resources (small ξ), it may still be appropriate to use higher planes and/or thicker wire types for some nets, for example for long nets crossing macros.

Our algorithm can be extended naturally to using several wiring modes within a repeater tree. In particular, preprocessing is done for all wiring modes, and the best choice depends on ξ. During buffering we will be able to use different wiring modes for different nets of the same repeater tree.

3.9 Experimental Results

To demonstrate the effectiveness of our algorithm we compare the results of our trees with lower bounds for the wire length and the number of inserted repeaters, as well as with upper bounds for the achievable worst slack.

We carried out experiments on all repeater tree instances of the chip Ludwig. The distribution of the instances by the number of sinks is given in Table 3.1. We have chosen Ludwig, because it is the largest design from the newest technology, 65 nm, in our testbed. Among the three technologies in our testbed, wire delays play the most important role in this technology, because here the wire resistances are highest. The chip contains 2.175 million instances. Thereof, 426 thousand instances are non-trivial, as they have more than two sinks.

The running time for constructing all 2.175 million topologies is less than 50 seconds on an Intel Xeon E7220 processor with 2.93 GHz. This speed proves the efficiency of the algorithm in practice.

Number of Sinks	Number of Instances
1	1391443
2	357057
3	152456
4	91859
5	48221
6	26195
7	24537
8	16147
9	13310
10	8254
11–30	32639
31–50	3913
51–100	7999
101–500	796
501–1000	181
> 1000	6
Total	2175013
> 2	426513

Table 3.1: Distribution of repeater tree instances.

The experiments were carried out with $\lambda = \frac{1}{4}$, and performed for five different topology tradeoffs $0.0, 0.25, 0.5, 0.75$, and 1.0.

For wirelength, we compare the lengths of our topologies to the length of a minimum Steiner tree. For up to 30 sinks we compute a provable optimum Steiner tree. For larger instances we approximate this bound by effective Steiner tree heuristics, which are usually within 1% of the optimum length on average.

Table 3.2 on page 58 shows the results for several topology tradeoffs ξ. The average numbers (column "Avg.") show the excess of the total length of our topologies compared to the total length of the minimum Steiner trees in percentage, for all instances matching the number of sinks in that row. The maximum numbers (column "Max") contain the maximum deviation among all instances covered by that row.

For $\xi = 0.0$, the total wire length exceeds its lower bound by only 0.71%. Note, we did not implement the variant of Theorem 3.15, which iteratively connects the sink which is closest to the partial topology and guarantees the maximum deviation to be within 50% of the minimum Steiner tree length. Hence, there are a few instances for which we exceed the minimum Steiner tree length by more than 50% percent for $\xi = 0.0$ (for instance by 97.66%). As the total length deviation is already very small, implementing the proposed variant is not too important. We could of course use other existing Steiner tree heuristics with a performance guarantee directly. Furthermore, we never use the setting $\xi = 0.0$ in our timing closure flow, but choose at least a small positive value for ξ to avoid extreme daisy chains also for resource efficient trees.

When targeting the topologies towards slack aware trees by increasing ξ, the average length deviations as well as the maximum deviations increase, as expected.

Upper bounds for the slack are computed as follows. For less than five 5 sinks we enumerated all topologies to obtain an upper bound for the achievable slack. For more sinks we applied Huffman coding, as described below Theorem 3.12, using the approximate representation (3.21) for $\alpha_{\frac{1}{4}}$. The slacks are measured in the delay model of our topology.

Table 3.3 summarizes the average slack deviation (of all instances covered by the number of sinks in that row) in picoseconds and the maximum worst slack deviation among those instances, also in picoseconds. It can be seen nicely how the slack deviations improve with increasing ξ. Also note that, for $\xi = 0.75$, we achieve already very good average slack deviations with a moderate total wire length deviation of only 5%.

The slack numbers above were computed using the delay model of the topology generation. Another interesting question is, whether the computed topologies prove useful after repeater insertion. For this purpose we computed a lower bound on the worst slack that includes repeater insertion. For each sink, we build a buffered path, ignoring any branchings. The worst slack over all sinks we obtain this way is an upper bound for the achievable slack of a buffered topology. Table 3.4 shows the slack deviation of the buffered topology with respect to this bound.

The table shows the average and maximum slack deviations for the extreme settings $\xi = 0.0$ and $\xi = 1.0$. The applied buffering algorithm scales between power and performance, like our topology algorithm. In these tables it was run with highest slack effort assuming unconstrained resources.

In the forth and seventh column we compare the total number of inserted repeaters (of all instances covered by that row) with a lower bound, which is computed as follows. Let $C_{\min}$ be total capacitance of a minimum Steiner tree connecting the root with all sinks plus the input capacitances of all sinks minus minus the maximum capacitance the root r may drive with the given input slew optslew_x such that its output slew is at most optslew_y, with suitable $x, y \in \{\text{rise}, \text{fall}\}$. Furthermore, let $c_{\max}(t)$ be the maximum load an inverter $t \in \mathcal{B}$ may drive, in order to maintain the optimum slew. Now, using $k_t \in \mathbb{N}_0$ inverters of type t the following inequality has to hold

$$C_{\min} + \sum_{t\in\mathcal{B}} icap(t)k_t \leq \sum_{t\in\mathcal{B}} c_{\max}(t)k_t,$$

in order to maintain the optimum slew specification. To obtain a lower bound for the number of repeaters we want to minimize

$$\sum_{t\in\mathcal{B}} c_t k_t,$$

where we simply set $c_t = 1$ if t is an inverter, and $c_t = 2$ if t is a buffer, which always consists of two internal inverters. Solving this integer linear program we obtain that

	$\xi = 0.0$		$\xi = 0.25$		$\xi = 0.5$		$\xi = 0.75$		$\xi = 1.0$	
#Sinks	Avg.	Max	Avg.	Max	Avg.	Max	Avg.	Max	Avg.	Max
1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	0.00	0.00	0.76	79.38	1.80	93.91	3.49	94.37	35.11	99.72
4	0.14	23.95	1.13	80.09	2.67	90.26	4.83	101.16	28.03	191.83
5	0.22	29.27	1.50	75.07	3.72	89.42	6.69	128.87	50.69	190.49
6	0.73	27.64	2.44	77.96	5.06	92.85	8.43	105.02	31.99	226.77
7	0.54	25.60	1.44	80.19	2.81	86.86	5.10	107.87	49.39	196.15
8	0.83	32.65	1.64	67.75	3.28	99.47	6.78	125.62	43.79	343.98
9	1.09	31.82	2.09	66.51	4.00	81.94	7.46	106.93	42.69	239.26
10	1.03	32.27	1.95	78.40	6.45	90.61	10.66	154.01	41.95	303.73
11–30	2.97	52.93	4.29	66.07	6.79	103.35	10.55	163.06	40.08	530.83
31–50	4.89	36.21	5.95	36.83	8.68	70.98	13.46	104.46	53.05	748.98
51–100	1.99	41.17	2.38	41.17	3.24	57.84	23.97	93.35	168.16	1204.69
101–500	8.68	34.84	10.25	40.33	12.28	48.18	15.65	106.04	67.22	1170.43
501–1000	17.60	52.73	21.50	56.09	26.59	63.51	34.93	95.91	331.91	2759.00
> 1000	55.11	97.66	55.08	98.02	55.35	97.67	56.62	97.67	65.09	97.67
Total	0.71	97.66	1.27	98.02	2.27	103.35	4.98	163.06	31.36	2759.00
> 2	1.27	97.66	2.26	98.02	4.04	103.35	8.87	163.06	55.84	2759.00

Table 3.2: Wire length deviations for varying topology tradeoff ξ (in %).

	$\xi = 0.0$		$\xi = 0.25$		$\xi = 0.5$		$\xi = 0.75$		$\xi = 1.0$	
#Sinks	Avg.	Max	Avg.	Max	Avg.	Max	Avg.	Max	Avg.	Max
1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	2.97	20.81	1.21	20.81	0.70	20.81	0.43	20.81	0.01	4.95
4	5.60	209.48	2.38	160.29	1.24	45.83	0.71	25.49	0.06	9.90
5	9.26	23.39	4.07	14.85	2.01	14.85	1.12	14.85	0.04	8.43
6	10.29	71.48	3.92	56.39	1.69	34.57	0.80	23.37	0.09	12.13
7	12.39	96.33	5.62	86.33	3.29	86.33	2.03	36.98	0.10	13.73
8	13.47	226.84	6.66	29.70	3.92	29.70	2.08	22.27	0.29	15.11
9	14.72	498.44	6.88	48.69	3.82	48.69	2.03	26.59	0.27	14.59
10	18.92	836.45	10.78	836.45	4.35	39.60	2.19	29.70	0.46	22.71
11–30	21.76	581.52	9.29	576.57	4.53	411.96	2.17	74.25	0.30	37.23
31–50	42.20	2584.16	20.26	228.37	10.61	199.12	5.23	119.10	1.15	43.47
51–100	76.68	2571.07	63.24	2566.12	57.10	1999.71	28.06	417.18	7.20	55.83
101–500	115.74	1079.06	39.32	220.90	19.73	172.04	9.53	102.41	1.64	36.16
501–1000	454.12	1117.79	135.21	476.82	62.17	198.58	39.24	128.69	11.63	113.03
> 1000	107.11	191.31	75.05	151.72	49.48	97.27	25.55	54.57	0.65	2.42
Total	1.94	2584.16	0.95	2566.12	0.58	1999.71	0.31	417.18	0.05	113.03
> 2	9.89	2584.16	4.86	2566.12	2.97	1999.71	1.56	417.18	0.24	113.03

Table 3.3: Slack deviations for varying topology tradeoff ξ (in ps).

	Topology Tradeoff $\xi = 0.0$			Topology Tradeoff $\xi = 1.0$		
	Slack (in ps)		# Repeaters (in %)	Slack (in ps)		# Repeaters (in %)
#Sinks	Avg.	Max	Avg.	Avg.	Max	Avg.
1	0.00	0.01	165.43	0.00	0.01	165.43
2	1.48	63.72	615.09	1.48	63.72	615.09
3	6.82	117.52	454.64	4.54	305.22	358.40
4	10.90	226.76	419.96	7.42	86.49	291.91
5	13.15	79.79	610.24	8.74	85.74	365.18
6	12.16	155.61	589.85	7.73	232.91	309.66
7	14.21	151.48	517.05	9.93	94.57	336.92
8	17.96	288.57	467.54	14.16	112.71	292.74
9	18.50	530.88	518.60	14.91	104.47	312.30
10	18.67	872.58	497.98	14.29	101.47	268.03
11–30	25.28	736.51	545.99	18.56	179.19	302.56
31–50	51.10	2842.37	557.88	36.72	169.57	397.22
51–100	51.47	3094.53	596.97	56.77	183.61	613.95
101–500	119.13	497.29	543.52	63.42	408.15	536.44
501–1000	499.97	1063.32	514.90	177.98	356.36	745.37
> 1000	191.81	420.41	686.79	136.66	253.57	687.42
Total	2.84	3094.53	416.52	2.12	408.15	329.50
> 2	13.26	3094.53	499.10	9.55	408.15	339.96

Table 3.4: Deviation of worst slacks and number of repeaters after high effort buffering (buffering tradeoff 1.0*).*

the optimum value $(k_t^\star)_{t\in\mathcal{B}}$ equals

$$\sum_{t\in\mathcal{B}} c_t k_t^\star = \left\lceil \min\left\{ \frac{c_t \cdot C_{\min}}{c_{max}(t) - icap(t)} \mid t \in \mathcal{B} \right\} \right\rceil .$$

Interestingly, the average slack deviation is relatively small even if the topology is constructed unaware of timing constraints (Table 3.4, $\xi = 0.0$). However, the big outliers are eliminated in the timing aware topologies ($\xi = 1.0$). The reason for the relatively good slack deviation of the timing-unaware topologies lies in the efficiency of the subsequent buffering algorithm to insert shielding repeaters. This becomes evident when comparing the number of inserted repeaters. Though having an overall shorter wire length much more repeaters are needed to achieve good slacks on timing unaware topologies.

Table 3.5 describes the mutual effect of varying the topology tradeoff and the buffering tradeoff. It shows the average and maximum slack deviations (after buffering) and the repeater usage deviation over all non-trivial instances (with more than two sinks) for several combinations of the buffering and topology tradeoffs. With a low (0.0) or medium (0.5) buffering tradeoff the number of inserted repeaters correlates with the topology tradeoff. But with a high buffering tradeoff, the relation between the topology tradeoff and the number of inserted repeaters is reciprocal.

Tradeoff		Slack (in ps)		# Repeaters (in %)
Buffering	Topology	Average	Max	Average
0.0	0.00	37	3056	126.45
0.0	0.25	37	3052	126.57
0.0	0.50	37	2285	126.64
0.0	0.75	37	796	127.98
0.0	1.00	36	556	154.13
0.5	0.00	26	3043	160.57
0.5	0.25	25	3039	160.99
0.5	0.50	25	2260	161.39
0.5	0.75	25	756	163.11
0.5	1.00	25	412	188.11
1.0	0.00	13	3094	499.10
1.0	0.25	10	3095	405.20
1.0	0.50	9	2236	363.25
1.0	0.75	9	744	340.71
1.0	1.00	9	408	339.96

Table 3.5: Buffering versus Topology Tradeoff

4 Circuit Sizing

4.1 Problem Description

In *circuit sizing* a layout or size for every circuit has to be found. Technically, a mapping $\beta : \mathcal{C} \to \mathcal{B}$ must be determined. For every $c \in \mathcal{C}$ there is a finite class $[\beta(c)] \subset \mathcal{B}$ of logically equivalent implementations for c. If $p \in P(\beta(c))$ we denote by $[p]$ the class of corresponding pins on logically equivalent books from $[\beta(c)]$. Figure 4.1 shows an example of three different layouts for an *INVERTER*.

Every layout has different delay characteristics. They depend on the size of the transistors, on the relative scaling between the p- and n-transistors, and on the applied materials. Precharacterized circuit libraries provide books which differ in four main characteristics:

- size,
- threshold voltage,
- beta ratio, and
- tapering.

The most important characteristic is the *size*. We say that a subset $\mathcal{B}' \subset [\beta(c)]$ *scales in size* if the relative sizes between the transistors within are almost constant throughout $\mathcal{B}'$, and if they do not differ in their materials. Generally, larger sizes yield higher drive strengths, as current can flow faster through a larger transistor-gate. This also implies higher capacitance limits at the output pins, and lower output slews, provided a constant input slew. On the other hand the input pin capacitances (blue wires in Figure 4.1) are increasing, and thereby slowing down the predecessors, which have to charge or discharge larger capacitances. Furthermore, larger circuits consume more power and area.

A characteristic that has become more and more important in the last years is the *threshold voltage* V_t. This is the voltage at which a transistor changes its state from insulation to conduction or vice versa. An n-transistor is switched off if the gate voltage is below the threshold voltage and turned on when it is larger than the threshold. A p-transistor behaves vice versa. Books with different thresholds can be realized, without changing the pin and transistor shapes, by varying the fabrication materials such as the gate conductor material (polysilicon vs. metal), the gate insulation material, the thickness of the gate material, and the concentration of the channel doping. As every threshold requires a separate mask and production step,

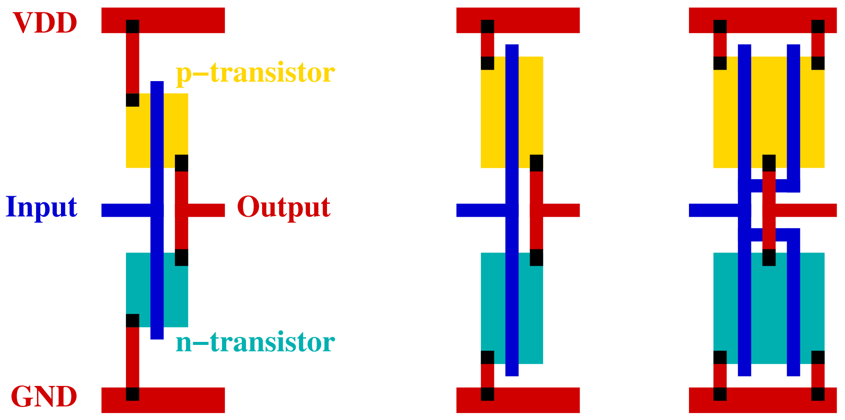


Figure 4.1: Example of three alternative layouts for an INVERTER. The figure shows a view from the top. The semiconductors are located in the center of the transistors, hidden below the blue input wires. The p-transistor is connected to the voltage supply network VDD. The n-transistor is connected to the ground network GND. If the input voltage is high the p-transistor is blocking and the n-transistor is admitting current flow. Potential load on the output net can be discharged to the ground (GND). If the input voltage is low the p-transistor is open and the n-transistor is blocking. Now the output is charged by the VDD-connection.

there are usually at most three different threshold voltages on a chip. The lower the threshold voltage the faster the circuit. As a drawback the leakage current increases exponentially when lowering the threshold voltage. Therefore, the number of low threshold voltage circuits must be kept small, as the leakage current creates power consumption even in a steady state.

The *beta ratio* determines the relative size of parallely arranged transistors within a single book. By changing the relative sizes, either the rising or the falling signal at the output can be accelerated.

The *tapering* characteristic is given for multiple input books with serially arranged transistors, for example *NAND*s. Here the relative size of serially arranged transistors is varied. This way the delays from individual input pins are decreased at the cost of other inputs, again by changing relative sizes of the internal transistors.

As it is difficult and probably impossible to provide strict definitions for the four categories, they are considered to be part of the input. For each characteristic there is a partition of $\mathcal{B}$ into its equivalence classes.

This section considers the optimization of the circuit sizes, which has the biggest impact on the path delays and electrical integrity, among the four categories. Varying beta ratio and tapering provide only small improvements of 5–10% of the worst path delay and will be considered shortly in Section 4.5, as a postoptimization routine. The threshold voltage assignment will be modeled and optimized as a Time-Cost Tradeoff Problem later in Chapter 6.

A widely used problem formulation for circuit sizing is to minimize the overall circuit power or size, while maintaining all late mode timing constraints:

$$\min \sum_{c \in \mathcal{C}} weight(c, \beta(c)) \tag{4.1}$$

such that (2.12), (2.20), and (2.22) are fulfilled for all late mode constraints between two signals (d, σ_d) and (c, σ_c). Here $weight : \mathcal{C} \times \mathcal{B} \to \mathbb{R}_+$ is some weighted sum of power, area, or other possible resources. Often *weight* depends only on the second parameter $\beta(c)$, but we want to allow that weights might be adjusted for individual circuits, for instance based on the local placement densities or the switching frequencies.

In practice global sizing of all circuits is mostly applied when no feasible solution with respect to the timing constraints (2.12) exists. A practical objective is to maximize the worst slack, but also push less critical negative slacks as far as possible towards the slack target, that is, find a leximin maximum vector of negative endpoint slacks. Such a solution would limit the need for other more resource-consuming optimization operations.

4.2 Previous Work

Circuit or transistor sizing is one of the key tasks during physical design and extensive literature exists on this subject. Here an overview of the most important, as well as recent results, is given. Most authors consider only single output circuits, called gates. Therefore, the problem is often called the *gate sizing problem.* The mathematically best-founded approaches rely on *geometric programming* formulations and assume continuously sizable circuits. Delay and slew functions are approximated by posynomial functions. A posynomial function $f : \mathbb{R}^n \to \mathbb{R}$ is of the form

$$f(x) = \sum_{k=1}^{K} c_k x_1^{a_{1k}} x_2^{a_{2k}} \cdots x_n^{a_{nk}}, \tag{4.2}$$

where $c_k > 0$ and $a_{ik} \in \mathbb{R}$. A *geometric program* is an optimization problem of the form

$$\min \; f_0(x) \tag{4.3}$$

$$\text{such that} \quad f_i(x) \leq 1, i = 1, \ldots, m, \tag{4.4}$$

$$g_i(x) = 1, i = 1, \ldots, p, \tag{4.5}$$

were f_i are posynomial functions, and g_i are monomials. A monomial is a posynomial with only one summand. The objective (4.3) reflects some power or area minimization while delay and circuit size constraints are modeled by (4.4) and (4.5). Many authors consider the gain-based delay model which falls into a special class of posynomials, where exponents are restricted to $a_{ik} \in \{-1, 0, 1\}$. Gain-based delays result from modeling not only wire delays by the Elmore delay formula, but also the circuits/transistor delays. Slews are not considered in this model, only the delay through a circuit is approximated. Let $\beta^{gain} : \mathcal{C} \to \mathbb{R}_+$ specify a continuously sizable realization of a book. The gain-based delay ϑ^{gain} of a timing edge $e = (v, w) \in E^{\mathcal{T}}$ depends only on the size of a circuit and the sizes of its successors. It is given by

$$\vartheta_e^{gain} = \vartheta_e^{intr} + \frac{\gamma_e + \sum\limits_{c' \text{ successor } of w} \delta_{e,c'} \cdot \beta^{gain}(c')}{\beta^{gain}(c)}, \tag{4.6}$$

where ϑ_e^{intr} denotes some intrinsic delay and $\gamma_e, \delta_{e,c'}$ are appropriate adjust factors.

A posynomial formulation for the circuit/transistor sizing problem was introduced by Fishburn and Dunlop [1985]. A globally optimum solution to the transistor sizing problem based on geometric programming was first given in Sapatnekar et al. [1993]. The nice property of geometric programs is that they can be turned into *convex programs* by variable transformation ($y_i = \log x_i$). Therefore, a local optimum is always a global optimum.

However, solvable instance sizes for general purpose geometric programming solvers, which typically implement an interior point method for convex programs, have at most 10 000 variables according to Boyd et al. [2005], who also give a detailed tutorial on geometric programming approaches.

Therefore, they are unemployable for full chip instances with multi-million circuits. Marple [1986] proposed Lagrangian relaxation methods for transistor sizing. Here the timing constraints are relaxed and equipped with Lagrangian multipliers. He combined an augmented Lagrangian method to obtain a globally near-optimum solution with a projected Lagrangian method for local refinement. Chen et al. [1999] gave a detailed theoretical analysis of the Lagrangian relaxation formulation. One key result is that the Lagrangian multipliers must form a combinatorial network flow. Thus the search for Lagrangian multipliers can be restricted onto the space of network flows. Langkau [2000] and Szegedy [2005b] applied that approach to standard circuit designs. Rautenbach and Szegedy [2007] proved that a subtask of the subgradient method—the optimization for fixed Lagrangian multipliers—can be done with linear convergence by cyclic relaxation for general networks. Though the Lagrangian relaxation formulation tends to handle larger instances than general purpose solvers it is based on an inaccurate timing model. Furthermore it can only be shown that the limes inferior converges towards the optimum solution. High sensitivity to step size multipliers and initial points were reported by Tennakoon and Sechen [2002]. Similar observations were made by Szegedy [2005a].

The rounding of a continuous solution to books from a circuit library was recently improved through dynamic programming by Hu et al. [2007]. However, the running times of dynamic programming are too large for global application on multi-million circuit instances. In that work an overview on the sparse literature for the rounding step can be found.

All geometric programming formulations have the disadvantage of low accuracy, which comes along with the simplified delay models that enable the convex problem formulation. Conn et al. [1998, 1999] proposed non-convex optimization based on accurate circuit simulations. Based on the simulation results, the local optimum is found by gradient descent.

The fastest known algorithms for large scale gate sizing rely on delay budget heuristics. In a first step a delay budget is assigned to each circuit or transistor and then in a second step each circuit is assigned to a minimum size solution that maintains this budget. Under the RC-delay model and by ignoring slews, the actual sizing can be performed in backward topological order of the circuit graph. The first delay budgeting approaches were proposed in Chen and Kang [1991], Dai and Asada [1989], Heusler and Fichtner [1991] for transistor sizing. Newer approaches can be found in Sundararajan et al. [2002], Kursun et al. [2004], Ghiasi et al. [2006]. The latter start with a feasible solution that preserves all timing constraints, and then distribute positive timing slacks of non-critical paths as an excess delay budget to each individual circuit or transistor. The initial solution is found by local search methods such as Fishburn and Dunlop [1985]. The budgets are used to minimize power by reducing circuit or transistor sizing sizes. The optimality statements as the one in Sundararajan et al. [2002] hold only under simplified delay models, where especially the delay through a circuit does not depend on the size of the predecessor. In the presence of slew propagation this is usually not true.

In the last years, most works on gate sizing considered the problem under statistical timing constraints (see Agarwal et al. [2005], Sinha et al. [2006], Dobhal et al. [2007], Singh et al. [2008]). Mani et al. [2007] consider combined statistical circuit sizing and voltage threshold optimization for power and timing by an interior point solver. A statistical gate sizing algorithm considering a postsilicon tunable clocktree was proposed by Khandelwal and Srivastava [2008]. Statistical effects can be approximated reasonably only after routing, therefore these models are of limited relevance to the prerouting timing closure problem. After detailed routing is done, the objective is to minimize rather the number of design changes, here circuit size changes, because the necessary reroute is typically very running time intensive. A global optimization that sizes all circuits at once is hardly possible in this stage.

4.3 New Approach

The mathematical beauty of geometric programming approaches lies in the computability of a global optimum. The drawback of all geometric programming models is that they can handle only small instances or rely on significant simplifications. In addition, when applied to standard cell/circuit ASICs, the inaccuracy of the timing model and the need for rounding to discrete circuits can undermine the advantage of a globally optimum solution.

The drawback of existing delay budget heuristics is that they also used simplified delay models and are especially ignoring slew effects, through which the predecessor layout has an influence on the delays through the currently considered gate. Our approach falls into the class of delay budget algorithms, but it overcomes the limitation of unrealistic delay models. Instead of delay budgets, we assign slew targets, which is principally equivalent. But the slew targets are not only used to size the circuits to which a slew target is assigned, but to deliver reasonable slew estimate when sizing the successors. Thus our approach enables more accurate local refinement. Using a fast lookup-table based refinement we are able to size multi-million circuit designs within one hour on a Xeon processor. Throughout the course of the algorithm real (discrete) circuit sizes are maintained. Delays are computed by static analysis according to Section 2.4. We do not focus on the globally critical paths but make choices mainly based on the local criticality of circuits compared to their predecessors. Therefore, our algorithm tolerates incomplete timing assertions, which often occur in early design stages and cause unreasonable bad slacks.

As timing analysis is time consuming it avoids incremental timing updates, but analyzes the complete timing of the chip, once all circuit sizes were refined. Furthermore it takes full advantage of a timing engine that performs parallel timing analysis. The algorithm itself can be parallelized easily.

After the fast global sizing, we apply local search to each circuit on the critical paths by evaluating the layout changes exactly. To enable optimization of less critical paths, it memorizes locally optimum circuits which could not be improved, to hide them from further processing.

We demonstrate the quality of our algorithms by comparing the results with lower bounds for the minimum achievable area and worst path delays. On average the results are within 16% of the minimum possible area consumption and within 2% of the lower bound on the achievable worst path delay.

4.4 Fast Circuit Sizing

The idea of the algorithm is that slews and delays have to be small on critical paths and can be relaxed on non-critical paths. A simple approach of forcing each slew to a tight value already gives reasonable slack results, but consumes too much area on non-critical paths. Additionally, critical circuits with more critical predecessors should be sized rather small, to reduce the load capacitances of the predecessors. We assume that driver strength and input pin capacitance correlate. In rare cases this assumption might not be valid, for example if a next larger equivalent circuit contains two more internal inverters at the output pin. Such situations will be absorbed by the subsequent local search described in Section 4.5.

Starting with some initial slew target $\text{slewt}(p) \in \mathbb{R}^+$ for each output pin p on a circuit, Algorithm 3 iteratively chooses smallest possible circuit sizes such that the targets are met. Based on the slack values the slew target $\text{slewt}(p)$ is then relaxed or tightened based on the global and local criticality of p.

The algorithm avoids incremental timing updates to prevent propagation of timing information through the forward and backward cone of each changed circuit. Instead, timing is updated for the complete design in line 4. This way the algorithm can take full advantage of a parallel timing engine. The actual optimization, line 3 and line 5, can also be parallelized.

As seen in Section 2.4, circuit delay and output slew are correlated and monotonically increasing functions in the two delay calculation parameters load capacitance and input slew. Therefore, we can equivalently assign either delay targets, slew targets, or any combination of them. The output slew is an input parameter of the delay calculation in the next stage, in which upper slew limits are given by design or technology rules. Furthermore slew values have a similar range for all books in the library $\mathcal{B}$ while delays vary with the internal logical depths. But most importantly, we can use the slew target at the predecessor circuits to obtain good estimates for the input slew when assigning circuits to books in line 3.

Algorithm 3 Fast Gate Sizing

```
1: Initialize slew targets for all circuit output pins;
2: repeat
3:     Assign circuits to books;
4:     Timing analysis;
5:     Refine slew targets;
6: until Stopping criterion is met
7: Return best assignment of all iterations;
```

The stopping criterion is met in line 6 when the current circuit assignment in comparison to the last iteration

1. worsens the worst slack, and
2. increases a weighted sum of -1 times the absolute value of the worst negative slack, the absolute value of the sum of negative slacks divided by the number of endpoints, and the average circuit area.

If the criterion is met, the assignment of the previous iteration, which achieves the best present objective value, is taken. The criterion is never met while the worst slack is improving. A lower worst slack is tolerated if it is accompanied by sufficiently large gains in the total negative slack or average circuit area.

4.4.1 Circuit Assignment

The task here is to assign each circuit $c \in \mathcal{C}$ to a logical equivalent book $B \in [\beta(c)]$ with smallest weight $weight(c, B)$ such that each slew target $\text{slewt}(p), p \in P_{out}(c)$ is met. As primary sizing constraints the load capacitance limits $\text{caplim}(p, \zeta), \zeta \in \{rise, fall\}$ in p must be met, as well as the slew limits at the sinks of the net $N \in \mathcal{N}, p \in N$. If the limits cannot be met, the amount of the violation is to be minimized. As VLSI engineers often assign low slew limits that are rather intended to be targets, capacitance violations are prioritized in case of conflicts between capacitance and slew violations.

The difficulty here is that circuits cannot be assigned independently, as the slews in p depend—besides on $\beta(c)$—on the input slews and the load capacitances downcap$(p, late)$, which in turn depends on the successor input pin capacitances. Algorithm 4 summarizes the circuit assignment.

Algorithm 4 Fast Gate Sizing—Circuit Assignment

```
1: $\beta(c) := \arg\min\{size(B) | B \in [\beta(c)]\}$;
2: repeat
3:     for $c \in \mathcal{C}$ in order of decreasing $d(c)$ do
4:         for $(p,q) \in G^{\mathcal{T}}, q \in P_{out}(c)$ do
5:             Estimate input slews $slew(p,\sigma)$ for all $\sigma \in \mathfrak{S}(p)$;
6:             $B_{(p,q)} := \arg\min\{size(B) | B \in [\beta(c)]$,slew targets in $q$ met$\}$;
7:         end for
8:         $\beta(c) := \arg\max\{size(B_{(p,q)}) | (p,q) \in G^{\mathcal{T}}, q \in P_{out}(c)\}$;
9:     end for
10: until $\beta(c)$ unchanged $\forall c \in \mathcal{C}$
```

We traverse the circuits in a cyclic order. Let the circuit graph $G_{\mathcal{C}}$ be defined as the directed graph that contains a vertex for each $c \in \mathcal{C}$ and an edge $e = (c, c') \in E(G_{\mathcal{C}})$, if there is a net connecting a pin $p \in P_{out}(c)$ with a pin $p' \in P_{in}(c')$. We assume that the circuit graph would become acyclic when removing all edges entering register vertices. This is usually true. In exceptional cases, where some cycles do not contain a register, we iteratively assign an arbitrary circuit from such a cycle to the set of registers until each cycle contains at least one assigned register.

Circuits are processed in order of decreasing longest distance from a register in the acyclic subgraph, which arises by removing the edges entering register vertices, as described above, and where all edge lengths are chosen as one. Let us first assume the assignments $\beta(r)$ of registers $r \in \mathcal{C}$ to be fixed. Thus when assigning a circuit $c \in \mathcal{C}, d(c) > 0$ the successors of c are already assigned and downcap$(p, late)$ is known for all $p \in P_{out}(c)$. The exact input slews are not known as the predecessors are still to be processed. We estimate the input slews by a weighted sum of the worst input slew computed by the last timing analysis and the anticipated input slew based on the predecessor slew targets. The weighting gradually shifts from pure predecessor target to "real" timing analysis results with each global iteration. Initially it is assumed that the predecessors will be sized such that their output slews are close to their targets. Later, when a change in the predecessor size is less likely, the real slew values dominate.

The load capacitance and the estimated input slew influence the circuit timing. Based on these values, circuits are assigned in line 6 to the smallest book that maintains the slew targets or to the largest book if no feasible book exists. Line 6 can be approximated efficiently via table look-up. For the pair $([p], [q])$ of classes of pin definitions we subdivide the feasible ranges of

- downstream capacitances $[0, \max\{\text{caplim}(q') | q' \in [q]\}]$,
- input slews $[0, \max\{\text{slewlim}(p') | p' \in [p]\}]$

- target slews $[0, \max\{\text{slewlim}(p') | p' \in [p]\}]$

into discrete sets and compute the smallest feasible layout $B_{(p,q)} \in [\beta(c)]$ for all discrete target slew, input slew, and downstream capacitance triples. In an actual assignment the three input values are then rounded to their next discrete value. The minimum book is then looked-up in the table. To guarantee the slew target, we can force rounding to the next pessimistic values, that means to round down the slew target, and to round up the input slew and the downstream capacitance. By refining the discretization arbitrary approximation guarantees can be achieved. In practice we use 20 discrete values for the slew ranges and up to 150 values for the cap values. We vary the latter with the number $|[\beta(c)]|$ of equivalent books.

To preserve the input slew limits at the successor gates, these limits minus the slew degradation on the wire may trim the output slew target. Therefore, it is not only necessary to provide valid load capacitances at all times, but also wire delays. After a level of circuits with equal distance labels is processed, we recompute wire delays for all input nets of altered circuits.

As the circuit graph is cyclic, the assignment algorithm needs to traverse the circuits several times until no more registers are altered. Registers usually have only a small set of 3 or 4 equivalent books, and their assignment does not change often. In practice only a few circuits are reassigned even in the second iteration. To speed up the overall algorithm, we perform only a single iteration, leaving it to the next global iteration to remove slightly suboptimum or illegal assignments.

4.4.2 Refining Slew Targets

In line 5 of Algorithm 3, the slew target $\text{slewt}(p)$ is refined for each output pin p on a circuit c based on what we call the *global* and *local criticality* of p. For simpler notation, we assume without loss of generality $\text{S}_\text{l}^\text{tgt} = 0$ throughout this section. The *global criticality* is simply the worst slack

$$slk^+(p) := \min\{slack(p,\sigma) | \, \sigma \in \mathfrak{S}(p)\} \tag{4.7}$$

at p. The pin p is *globally critical* if $slk^+(p) < 0$.

The *local criticality* indicates whether the worst slack at p and any direct predecessor pin of c can be improved either by accelerating c, that is, by decreasing $\text{slewt}(p)$, or by decreasing the input pin capacitances on c, that is, by increasing $\text{slewt}(p)$. We define the local criticality as the difference between the slack at p and at a worst-slack direct predecessor p' of c (see Figure 4.2).

Note that we consider p' even if it is not in the backward cone of p (in the timing graph), which is possible if c is a complex circuit and the internal timing graph is not a complete bipartite graph. Formally, we define the predecessor criticality of the circuit c by

$$slk^-(c) := \min\{slack(p',\sigma') \mid p' \text{ direct predecessor pin of } c, \sigma' \in \mathfrak{S}(p')\}. \tag{4.8}$$

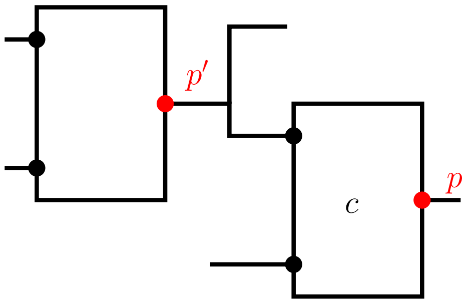


Figure 4.2: Local criticality: slack difference of p and the worst predecessor p'.

Obviously, $slk^+(p) \geq slk^-(c)$ for all circuits but registers, since a path that determines the slack in p must contain one of the predecessor pins. Registers may have smaller output slacks than their predecessors as inputs and outputs may belong to different data paths. Therefore, we define the *effective predecessor slack* for the output pin $p \in P_{out}(c)$ by $slk^-(p) := \min\{slk^-(c), slk^+(p)\}$. This implies $slk^-(p) = slk^-(c)$ for non-register circuits. Finally the local criticality $lc(p) \geq 0$ of p is defined by

$$lc(p) := slk^+(p) - slk^-(p). \tag{4.9}$$

If $lc(p) = 0$ then p is either located on a worst-slack path through a most critical predecessor of c, or p is an output pin of a register whose output path is at least as critical as any path through its predecessors. We call p *locally critical* if $lc(p) = 0$.

Algorithm 5 shows how the slew targets of the circuit c are updated in an iteration $k \in \mathbb{N}$. If p is globally and locally critical, we decrease $\text{slewt}(p)$ by subtracting a number that is proportional to $|slk^+(p)|$, but does not exceed some constant max_change (line 8). Otherwise we increase $\text{slewt}(p)$ by adding a number that is proportional to $\max\{lc(p), slk^+(p)\}$ (line 10).

The constant γ can be thought of as an estimate for $\frac{\partial \text{slewt}(p)}{\partial slk^+}$. Thus, if $\text{slewt}(p)$ is tightened, $\gamma \cdot |slk^+|$ expresses the required slew change to reach the slack target in p. If $\text{slewt}(p)$ is relaxed, $\gamma \cdot \max\{slk^+, lc\}$ expresses the required slew change, either to align the slack in p with the worst predecessor slack (if $slk^+ \leq lc$), or to decrease the slack to the target (if $slk^+ > lc$). As we modify all circuits in each iteration, γ should be in the range of a fraction of $\frac{\partial \text{slewt}(p)}{\partial slk^+}$. It could even be set individually for each output pin definition.

The other multiplier θ_k is a damping factor defined as $\theta_k = (\log(k + const))^{-1}$. Following the subgradient method for continuous optimization, it damps the slew target change and prevents potential oscillation. However, we observed that damping does not have a considerable influence on our algorithm.

To avoid too big changes, $\text{slewt}(p)$ is furthermore changed by at most max_change. The slew target is never increased above the slew limits given by the rules or assertions. Slew targets of locally non-critical circuits are relaxed, which indirectly decreases the load capacitance at the more critical predecessors. Slew targets of globally non-critical circuits are increased to save power.

In this scenario, the slew targets on the critical path would converge towards zero. Such a target is not realizable by any book and would lead to a very slow growth of the slew target if a critical circuit becomes non-critical in subsequent iterations. To avoid unrealistically small slew targets, a lowest allowed slew target $\underline{\text{slewt}}([p])$ is computed, where $[p]$ denotes the class of corresponding output pins on equivalent books in the library. This is done by constructing a long chain of equally sized circuits for all equivalent books. The circuits are placed in small distance to each other. Then stationary slews are computed as in Section 3.3.1. If several input pins exist, we always connect p to the one that achieves the largest output slew at the end of the chain. As $\underline{\text{slewt}}([p])$ the minimum over all output slews of the elements in $[p]$ is chosen.

Algorithm 5 Fast Gate Sizing—Refining Slew Targets

Input: A circuit c and an iteration number k

```
 1: procedure RefineSlewTargets($c, k$)
 2:     $\theta_k = 1/\log(k + const)$;
 3:     $slk^- \leftarrow \min\{slack(p) | p \in \Gamma^-(P_{in}(c))\}$;
 4:     for all $p \in P_{out}(c)$ do
 5:         $slk^+ \leftarrow \text{slk}(p)$;
 6:         $lc \leftarrow slk^+ - \min\{slk^-, slk^+\}$;
 7:         if $slk^+ < 0$ and $lc < 0$ then
 8:             $\text{slewt}(p) \leftarrow \text{slewt}(p) - \min\{\theta_k \cdot \gamma \cdot |slk^+|, \text{max\_change}\}$;
 9:         else
10:             $\text{slewt}(p) \leftarrow \text{slewt}(p) + \min\{\theta_k \cdot \gamma \cdot \max\{slk^+, lc\}, \text{max\_change}\}$;
11:         end if
12:         Project $\text{slewt}(p)$ into feasible range $[\underline{\text{slewt}}([p]), \text{slewlim}([p])]$;
13:     end for
14: end procedure
```

If no sizing has been performed yet, the slew targets are initialized by a multiple of the lowest allowed target in line 1 of Algorithm 3: $\text{slewt}(p) = \alpha \times \underline{\text{slewt}([p])}$ for all circuit output pins p in the design. Otherwise, when running on a preoptimized input, the slew targets are initialized by the currently computed slews.

The algorithm is quite stable with respect to α. The variation of final worst path delays stays within 10% when scaling α between 1 and ∞. If $\alpha = \infty$, slew targets are initialized by the slew limits. For running time reasons, α should be set to a value from which the lowest allowed slew targets as well as the slew limits, are reachable within a few iterations.

4.4.3 Enhanced Slew Targets

As described so far, Algorithm 3 yields already good results. The main disadvantage is that the sizing step assumes that predecessors can be enlarged sufficiently. In some cases this can lead to overloaded circuits that cannot be enlarged further, or to locally non-critical circuits that cannot be down-sized sufficiently because of too large successors, which in turn are locally critical. In the overall timing optimization flow, the gate sizing is interrupted by a repeater insertion step that absorbs such situations.

However, this can lead to unnecessary repeaters or to poor results on nets that must not be buffered for some reason. One solution would be to consider the predecessor driver strengths directly when sizing a circuit c. Unfortunately, the exact predecessor sizes and their load capacitances are unknown.

We propose a different approach that fits well into the sizing paradigm. When refining the slew target of an output pin p on a circuit c, we compute an estimated slew slew^- at the worst predecessor output pin p' of the circuit c. This is done by a weighted sum of $\text{slewt}(p')$ and $\text{slew}(p')$ as estimated in Section 4.4.1. If $\text{slew}^- > \text{slewt}(p)$, we increase the slew target in p by

$$\text{slewt}(p) := \lambda \cdot \text{slewt}(p) + (1 - \lambda) \cdot \text{slew}^-,$$

with $0 < \lambda \leq 1$. The increased target will lead to a smaller circuit size and smaller input pin capacitances, and thus smaller loads in p'. The effect of an extraordinary high value of slew^- declines exponentially in the number of subsequent stages.

The local search described in Section 4.5 is called after the fast global gate sizing in our timing closure flow (see Chapter 7). It improves the slacks on the most critical paths based on exact timing evaluation.

4.4.4 Power Reduction

If the circuit area after fast gate sizing exceeds the available placement area or the power limit is violated, circuit sizes have to be decreased to obtain a feasible solution. One possible approach would be to iteratively increase the targets for all circuits equally. But for effective size reduction, especially low slew targets have to be relaxed. Another point is that the knowledge obtained during power reduction should be transferred to subsequent circuit sizing calls within the timing closure flow.

We choose the following approach. Slews are relaxed by increasing the lowest allowed slew targets $\underline{\text{slewt}([p])}$ for all classes of output pin definitions. This prevents too small slew targets and too large circuits. In an area recovery mode Algorithm 3 tests whether one of the area and power constraints is violated, and, if so, increases the lowest allowed slew targets before refining the slew targets. The algorithm is continued until the area and power constraints are fulfilled. If the circuit sizing is called the for next time, the increased lower allowed slew targets will prevent a too high area and power consumption from the very beginning.

4.4.5 Electrical Correction

A special circuit sizing task is the electrical correction, which is the removal of load capacitance or slew limit violations, while slacks are ignored. In general, line 3 of Algorithm 3 chooses circuit sizes that preserve these limits on the output nets if possible. Combined with repeater insertion it usually does not leave any violations. However, a subsequent placement legalization can disturb the previously estimated wire capacitances and introduce capacitance and slew violations. Algorithm 3 can be adapted so that in each iteration only driver circuits of currently violated nets are sized with respect to a feasible slew target.

One potential problem of applying Algorithm 3 for the task of electrical correction is that it does resolve electrical violations on a net only by sizing-up driver circuits. In principle electrical violations could also be eliminated by sizing sinks to smaller books. However, there are reasons not to perform sink down sizing. First, the sink sizes were intended by the previous gate sizing, either to resolve electrical violations on their output nets, or to preserve positive slacks. Down sizing a sink might create too late arrival times that can only be resolved by a subsequent up-sizing. Second, for many violations the impact of the sink pin capacitances is too small to resolve it.

4.5 Local Search Refinement

The global circuit sizing heuristic from Section 4.4 creates worst slacks that are already close to optimum. However, they usually can be refined by local search, which improves book assignments based on exact effect evaluation. The local search in Algorithm 6 iteratively collects the most critical nets and sizes the attached circuits individually to a local optimum. The local objective for a circuit c is to maximize the worst slack of a pin in $P(c)$ and predecessor pins of c. If circuits are already at their local optimum the algorithm should proceed with less critical circuits in the next iteration. For this purpose unchanged circuits are marked as non-optimizable and neglected in the next iteration. Of course a non-optimizable circuit $c \in \mathcal{C}$ can become optimizable when circuits in its neighborhood changed or the slacks or slews of the signals in this neighborhood changed.

We first propose an algorithm that only looks at physical changes in the neighborhood, ignoring slack and slew changes.

In line 3 we sort the nets by increasing worst slack and collect all connected circuits until K circuits were selected. With a hard limit of K there may be uncollected circuits connected to a net with the same last slack $slack^K$. In order to obtain a deterministic algorithm that selects the same set of circuits regardless of the order in which they are stored, further circuits are added if they are connected to a net with source pin slack equal to $slack^K$. The selection of $\mathcal{C}^{crit}$ in line 3 of Algorithm 6

Algorithm 6 Local Search Circuit Sizing

```
1: Mark all $c \in \mathcal{C}$ OPTIMIZABLE;
2: repeat
3:     Select a set $\mathcal{C}^{crit}$ of critical and optimizable circuits;
4:     Mark all $c \in \mathcal{C}^{crit}$ as non-optimizable;
5:     for $c \in \mathcal{C}^{crit}$ do
6:         Map $c$ to locally best solution;
7:         Mark neighborhood of $c$ optimizable;
8:     end for
9: until Worst slack unchanged
```

is summarized by following formula

$$\mathcal{C}^{crit} := \arg\min\Big\{|\mathcal{C}'| : \mathcal{C}' \subseteq \mathcal{C}, |\mathcal{C}'| \geq K, \text{ there are no } \ c' \in \mathcal{C}', c \in \mathcal{C} \setminus \mathcal{C}', \\ \text{such that: } \mathrm{wslack}(P(N(c))) \leq \mathrm{wslack}(P(N(c')))\Big\},$$

where $\mathrm{wslack}(Q) := \min\{\mathrm{slk}(p, \sigma) \mid p \in Q, \sigma \in \mathfrak{S}(p)\}$ for a set of pins $Q \subseteq P$.

The locally best solution in line 6 of Algorithm 6 is a solution that achieves the maximum worst slack among pins of c and its predecessors. To determine the best solution $\beta^{opt}(c)$, $\beta(c)$ is set to all $B \in [\beta(c)]$ and the slack is evaluated. If this slack is above a given slack threshold $\mathrm{S}_\mathrm{l}^\mathrm{tgt}$ the most power efficient solution that achieves the $\mathrm{S}_\mathrm{l}^\mathrm{tgt}$ is chosen. If the predecessor c' has more critical slacks than c, c is located on a less critical side branch of a most critical path through c'. In this case c should be kept as it is or sized down in order to decrease load capacitances of c'.

Besides slack also load capacitance and slew violations need to be prevented or removed, though they occur rather seldom on critical paths. For this purpose load capacitance or slew violations at a pin $p \in P$ are multiplied by high numbers and then added to the slack $slack(p)$. This is a Lagrangian relaxation of capacitance and slew constraints. It allows to tradeoff between slack and small electrical violations.

The neighborhood in line 7 is defined by the set of all circuits that are adjacent to c via a net $N \in \mathcal{N}$. Slacks can change in the complete forward and backward cone of the predecessors of a changed circuit. In order to revisit all improvable circuits in the next iteration it would be necessary to mark all circuits as optimizable, which are located in these cones and where a slack change happened. To limit the running time, we mark only the direct neighbors, where the changed circuit size has the biggest impact.

Remark 4.1. *(Beta Ratio and Tapering Optimization)*
The local search method can naturally be applied to optimize beta ratios and tapering characteristics by also considering these alternatives in line 6 of Algorithm 6.

Chip	**Min Area**	**Fast Area**	**LS Area**	**LS/Min**
Fazil	548	648	642	1.171
Franz	655	786	766	1.170
Lucius	222	339	336	1.515
Felix	646	884	875	1.354
Julia	3 038	3 135	3 129	1.030
Minyi	4 699	4 804	4 805	1.023
Maxim	4 704	6 640	6 629	1.409
Tara	5 728	6 067	6 058	1.058
Bert	17 473	19 634	19 605	1.122
Karsten	38 968	39 590	39 572	1.016
Ludwig	41 616	42 376	42 356	1.018
Arijan	62 468	65 536	65 505	1.049
David	60 088	63 901	63 841	1.062
Valentin	77 766	96 951	96 911	1.246
Trips	69 097	79 719	79 615	1.152
Avg.				**1.160**

Table 4.1: Deviation from Area Bound

4.6 Quality of Results

To demonstrate the quality of our gate sizing approach, we compare the results of the gate sizing algorithms with lower bounds for the achievable area, and lower bounds for the achievable delay of the most critical path.

4.6.1 Area Consumption

A lower bound for the area consumption can be computed by starting with minimum sizes for all circuits and iteratively increasing the sizes of driver gates whose capacitance or output slew limit is violated. Table 4.1 shows the minimum feasible area consumption, the area consumption after fast gate sizing, the area consumption after local search refinement, and the ratio of the local search area over the minimum possible area. The reported area numbers are given in design specific units. On average the minimum area bound is exceeded by 16%. For the chips Lucius, Felix, and Maxim the area increase is significantly larger. These chips are small RLMs of very critical timing, where all logic paths have essentially the same criticality, leaving hardly room to safe area.

4.6.2 Delay Quality

For a lower bound on the worst delay, we identify the worst slack path P^{crit} after gate sizing. Then we size all circuits in the design to a minimum size as in the area bound computation above, fulfilling only capacitance and slew limits. We ignore and remove any arrival times and required arrival times on pins that are not

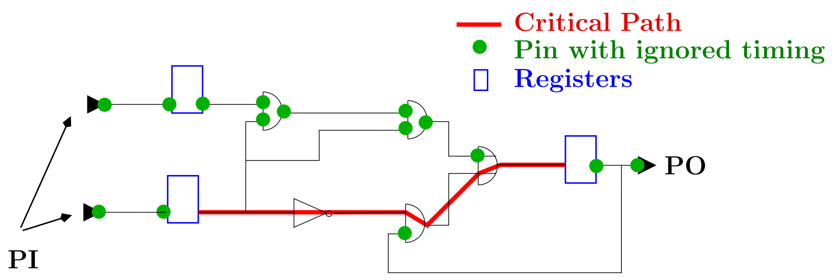


Figure 4.3: Lower Delay Bound Computation

located on P^{crit} as shown in Figure 4.3. This way, the arrival times and required arrival times on P^{crit} are independent from any other side-input or side-output of that path. Now the total delay of P^{crit} is reduced further by Algorithm 6 until no more improvement can be found. The gates on any branching from P^{crit} are set to a minimum possible size and therefore have a minimum impact on the load capacitances and delays on P^{crit}. The application of Algorithm 6 for minimizing the delay on the critical path is justified by following theorem.

Theorem 4.2. *Assuming posynomial delay functions and continuously sizable circuits, Algorithm 6 computes the minimum possible delay for the path P^{crit}.*

Proof. If all delays are posynomials, the minimization of the path delay corresponds to the minimization of a posynomial function $f : \mathbb{R}^n_+ \to \mathbb{R}$, with the circuit sizes $(x_i)_{1\le i\le n}, n := |\mathcal{C} \cap P^{crit}|$ as variables. The sizes are restricted to a (compact and convex) box $X \subset \mathbb{R}^n_+$ defined by lower $(l_i)_{1\le i\le n} \in \mathbb{R}^n_+$ and upper bounds $(u_i)_{1\le i\le n} \in \mathbb{R}^n_+$ with $l \le u$. The problem can be formulated as a geometric program:

$$\begin{array}{lcr} & \min f(x) := \sum_{k=1}^{K} c_k x_1^{a_{1k}} x_2^{a_{2k}} \cdots x_n^{a_{nk}} & \\ \text{such that} & l_i \le x_i \le u_i & \text{for all } 1 \le i \le n, \end{array}$$

where $c_k > 0$ and $\alpha_{ik} \in \mathbb{R}$ for $1 \le i \le n$ and $1 \le k \le K$.

After variable transformation $y := \log x := (\log x_i)_{1\le i\le n}$ the function $F(y) := f(e^y)$ with $e^y := (e^{y_i})_{1\le i\le n}$ is a convex function (Fishburn and Dunlop [1985]) restricted to the (compact and convex) box $Y \subset \mathbb{R}^n$ defined by: $\log l_i \le y_i \le \log u_i, 1 \le i \le n$. As the logarithm is strictly increasing and continuous, each local minimum $x^\star \in X$ of f corresponds to a local and global minimum $y^\star := \log x^\star := (\log x_i^\star)_{1\le i\le n} \in Y$ of F and thus is a global minimum of f on X. In this scenario, Algorithm 6 is a coordinate descent method that determines a global optimum of f on X.

□

In practice delay functions are not given as posynomials and sizes must be chosen from a discrete set. However, by complete enumeration on a huge number of paths with lengths bounded by 10, we verified empirically that the local search on P^{crit} finds the solution with minimum total delay in practice.

Chip	**Fast Gate Sizing**			**Local Search Sizing**		
	Delay	Bound	Ratio	Delay	Bound	Ratio
Fazil	6.65	5.85	1.14	4.38	4.30	1.02
Franz	4.47	4.25	1.05	4.75	4.69	1.01
Lucius	1.25	1.20	1.04	1.66	1.60	1.03
Felix	2.51	2.21	1.13	2.39	2.15	1.11
Julia	1.28	1.19	1.08	1.12	1.12	1.01
Minyi	3.01	2.96	1.02	2.96	2.96	1.00
Maxim	2.61	2.49	1.05	2.52	2.47	1.02
Tara	1.65	1.54	1.07	1.51	1.48	1.01
Bert	1.74	1.52	1.14	1.46	1.38	1.06
Karsten	9.36	9.00	1.04	8.68	8.66	1.00
Ludwig	12.18	11.97	1.02	11.80	11.80	1.00
Arijan	3.29	3.25	1.01	3.20	3.20	1.00
David	5.21	4.96	1.05	5.00	4.86	1.03
Valentin	4.69	4.55	1.03	4.56	4.48	1.02
Trips	6.71	6.00	1.12	5.89	5.64	1.04
Avg.			**1.07**			**1.02**

Table 4.2: Critical Path Delays and Lower Bounds (in ns)

The delay of the most critical path after fast gate sizing and the delay of the most critical path after the subsequent local search, as well as the respective path delay bound, are given in Table 4.2. The most critical paths and therefore the delay bounds can be different after fast gate sizing and after local search sizing. In particular the lower delay bound after fast gate sizing needs not to be a delay bound for the worst slack path after local search sizing. On the chip Felix the worst path delay after local search sizing misses the lower bound by 11%, which is much more than in the other cases. Here, the critical path contains several critical high fanout nets and trees with almost equally critical sinks. In the lower bound all less critical sinks are sized down aggressively, while this is not possible in the original problem. Thus, the lower bound is likely to be weak on this chip.

4.6.3 Running Time

The running times for the gate sizing in Table 4.3 were obtained on a 2.93 GHz Intel Xeon E7220. All computations were performed sequentially. Columns 2 and 6 show the number of iterations of fast gate sizing and refine gate sizing respectively. The timing analysis takes more than 50% of the total running time for fast gate sizing. It includes wire delay re-calculations for inputs nets of altered circuits. The circuit assignment needs a slightly higher percentage of the remaining running time than the slew target refinement. The number of fast gate sizing iterations was limited to 15.

In the experiments we have chosen $\lambda = 0.85$ (see Section 4.4.3) and a slew target initialization value of $\alpha = 2.5$. Empirically this yields a fast convergence to tight

Chip	**Fast Gate Sizing**				**Local Search**	
		Total	**Timing Analysis**			
	Iter.	**Run. Time**	**Run. Time**	**%**	**Iter.**	**Run. Time**
Fazil	8	0:00:45	0:00:22	49.29	15	0:00:49
Franz	8	0:00:52	0:00:29	55.79	20	0:01:28
Lucius	11	0:01:12	0:00:28	39.25	40	0:01:25
Felix	14	0:01:35	0:00:49	51.57	15	0:00:47
Julia	15	0:02:21	0:01:17	55.01	10	0:00:27
Minyi	15	0:03:52	0:02:18	59.48	10	0:00:41
Maxim	15	0:08:37	0:03:25	39.74	15	0:01:53
Tara	15	0:09:48	0:04:53	49.74	10	0:01:15
Bert	14	0:21:14	0:10:29	49.35	15	0:05:36
Karsten	7	0:40:38	0:19:17	47.48	10	0:05:19
Ludwig	7	0:31:49	0:19:50	62.30	10	0:03:56
Arijan	15	1:09:25	0:40:51	58.86	10	0:04:26
David	15	1:30:47	0:54:28	60.00	25	0:19:15
Valentin	15	2:14:05	1:21:16	60.61	5	0:04:10
Trips	13	2:03:50	1:02:42	50.64	40	0:50:33

Table 4.3: Gate Sizing Running Times (hh:mm:ss)

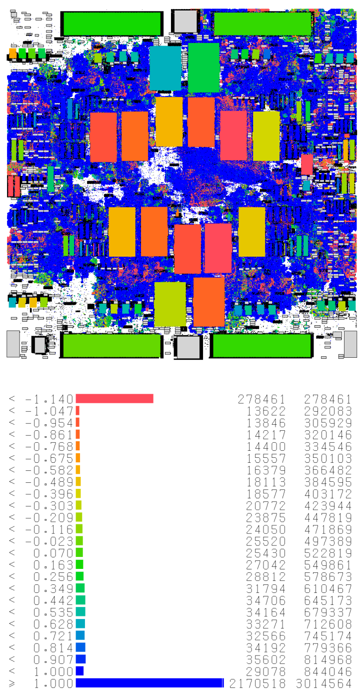
slews on critical paths and relaxed slews on non-critical paths.

The local search sizing was run in blocks of 5 iterations with $K = \left(\frac{|\mathcal{C}|}{1000} + 1000\right)$. This explains why all reported iteration numbers for the local search sizing are multiples of 5. After each block, the markers for optimizability were reset and the local search was restarted from the scratch, to account for those timing changes that are not recognized by the marking mechanism. The local search stopped if the worst slack was not improved within a block. The corresponding limit of 50 local search iterations was not attained in any run.

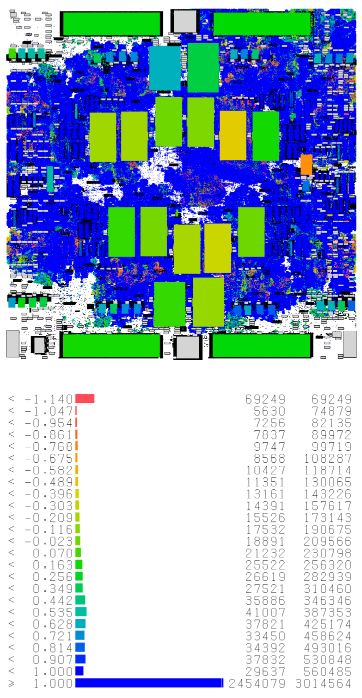
4.7 Circuit Sizing in Practice

We close this chapter with an example that demonstrates how the gate sizing approach works in practice on the chip "Karsten" with about 3 million sizeable circuits. Figure 4.4 shows six pairs of chip plots on the top and slack distribution histograms in the bottom. Each histogram contains one entry per circuit classified by the worst slack on one of its pins. The circuits in the plot are colored by slack with the same color as their corresponding histogram bar.

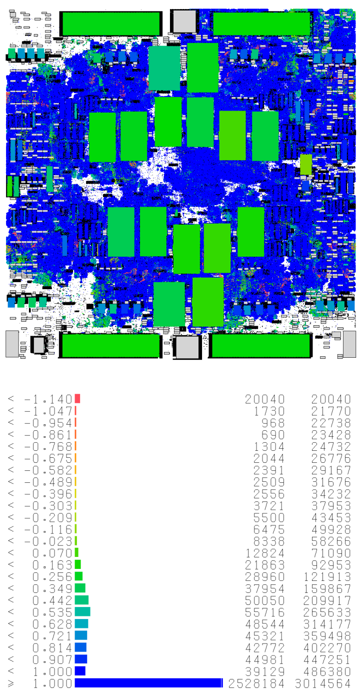
The first pair shows the slacks of a power optimal solution, where only electrical violations were optimized. The slack distribution quickly improves. After three iterations only a very few critical circuits are left. These are mainly eliminated after eight iterations of fast gate sizing. An extensive local refinement optimizes the worst slacks to almost zero, which is within 5% of the achievable optimum worst path delay. The final area increased by 1% in comparison to the minimum size area. Please note that these pictures were obtained on a slightly different placement than the results for Karsten in Table 4.2.



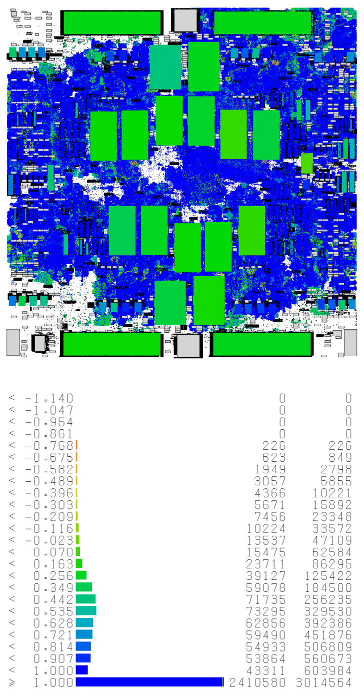
Slacks after Initialization



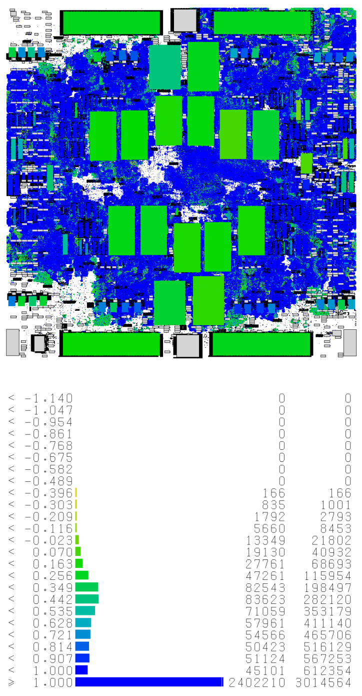
Slacks after 1. Iteration



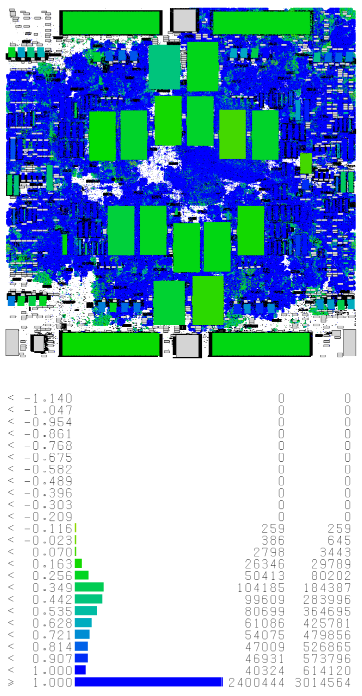
Slacks after 2. Iteration



Slacks after 3. Iteration



Slacks after 8. Iteration



Slacks after Local Search

Figure 4.4: Slack distributions on the chip Karsten in the course of Algorithm 3 and after local search.

5 Clock Skew Scheduling

For several decades the target of the clock distribution on computer chips was a *zero skew* tree or network, where all registers open and close simultaneously. In such a scenario timing optimization is restricted to modify delays on data paths, and the cycle-time is restricted by the slowest data path. Recall the setup constraint (2.12) between a late data signal σ_d at the data input pin d and an early testing clock signal σ_c at the clock pin c

$$\mathrm{at}(d, \sigma_d) + \mathrm{setup}\,(\mathrm{slew}(d, \sigma_d), \mathrm{slew}(c, \sigma_c)) \leq \mathrm{at}(c, \sigma_c) + \mathrm{adj}(\sigma_d, \sigma_c).$$

A closer look at the above inequality shows that it can potentially be resolved by increasing the clock arrival time $\mathrm{at}(c, \sigma_c)$ or by decreasing the arrival time of the clock signal that triggered σ_d. *Clock skew scheduling* is the optimization of clock signal delays instead of data path delays.

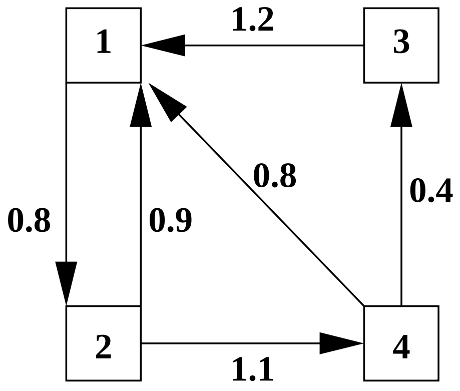


Figure 5.1: Example of a register graph.

We motivate the effectivity of clock skew scheduling by a simple example consisting of flip-flops only. The simple register graph $G^{\mathcal{R}}$ is defined as the directed graph whose vertex set is the set of flip-flops. Figure 5.1 shows an example of a register graph with four registers. Each $v \in V(G^{\mathcal{R}})$ represents the single triggering signal at the clock input of the flip-flop that is also the testing signal for the data input. $G^{\mathcal{R}}$ contains an arc (v, w) if the netlist contains a path from the data output of the register v to the data input of register w. The register graph defined here is called simple, because we will refine the definition of a register graph to account for further constraints in Section 5.2.1. Let $\vartheta(v, w)$ denote the maximum delay of a path from v to w. In the figure the delays are given by the edge labels. If all registers receive the same periodic clock signal with cycle time T, a zero skew clocktree will be feasible if and only if all delays are at most T. We simplify the setup constraints from (2.12) by assuming setup $\equiv 0$ and adj $\equiv T$.

In this example the minimum cycle time with a zero skew tree would be 1.2, bounded by the path $3 \to 1$. With clock skew scheduling this condition can be relaxed. Let register 3 in the above example switch earlier by 0.2 time units. Now signals on path $3 \to 1$ could spend 0.2 more time units per cycle. In turn the maximum allowed delay for signals on path $4 \to 3$ would decrease by 0.2 time units, which would not harm the overall cycle time, as the path delay of 0.4 is very fast. Now path $2 \to 4$ determines a limit for the minimum cycle time of 1.1.

Motivated by this observation, the question arises what is the best achievable performance for given delays? We ask for arrival times $\mathrm{at}(v)$ of the triggering clock signals at all registers v such that

$$\mathrm{at}(v) + \vartheta(v,w) \le \mathrm{at}(w) + T \tag{5.1}$$

holds for each arc (v,w) of the register graph. We call such arrival times *feasible.* In fact, the best achievable cycle time T due to clock skew scheduling is given by following theorem.

Theorem 5.1 (Szymanski [1992],Shenoy et al. [1992])**.**
Given a simple register graph $G^{\mathcal{R}}$ with arcs delays ϑ, the minimum cycle time T, for which feasible arrival times $\mathrm{at} : V(G^{\mathcal{R}}) \to \mathbb{R}$ *for the triggering clock signals exist, equals the maximum mean delay*

$$T = \frac{\vartheta(E(C))}{|E(C)|}$$

of a directed cycle in C in $G^{\mathcal{R}}$, where $\vartheta(E(C)) := \sum_{e\in E(C)} \vartheta(e)$.

Proof. T is feasible if and only if there are arrival times at such that:

$$\mathrm{at}(v) + \vartheta(v,w) \le \mathrm{at}(w) + T \quad \forall (v,w) \in E(G^{\mathcal{R}}).$$

From shortest path theory it is known that such arrival times (node potentials) exist if and only if $(\overleftarrow{G^{\mathcal{R}}}, c)$ does not contain any cycle with negative total cost, where $\overleftarrow{G^{\mathcal{R}}} := (V(G^{\mathcal{R}}), \{(w,v) \mid (v,w) \in E(G^{\mathcal{R}})\}$ and $c(w,v) := T - \vartheta(v,w)$ for all $(w,v) \in \overleftarrow{G^{\mathcal{R}}}$ (see Korte and Vygen [2008], Theorem 7.7). Now,

$$\begin{aligned} \sum_{(v,w)\in E(C)} (T - \vartheta(v,w)) &= \sum_{(v,w)\in E(C)} c(w,v) \ge 0 \quad \forall \text{ cycles } C \subseteq G^{\mathcal{R}} \\ \Leftrightarrow \qquad & \\ T &\ge \frac{\sum_{(v,w)\in E(C)} \vartheta(v,w)}{|E(C)|} \qquad \forall \text{ cycles } C \subseteq G^{\mathcal{R}}. \end{aligned}$$

Thus the minimum possible T equals the longest average delay of a cycle.

□

In the example of Figure 5.1 this gives an optimum cycle time of 0.9, which can be computed easily by enumerating all three cycles. In general the optimum feasible cycle time T and feasible clock signal arrival times at can be computed in strongly polynomial time by minimum mean cycle algorithms, for example by those of Karp [1978] or Young et al. [1991].

However, this simple situation is unrealistic. Today systems on a chip have multiple frequencies and often several hundred different clock domains. The situation is further complicated by transparent registers, user-defined timing tests, and various advanced design methodologies.

Moreover, it is not sufficient to maximize the frequency only. The delays that are input to clock skew scheduling are necessarily estimates. Detailed routing will be done later and will lead to different delays. Thus one would like to have as large a safety margin—positive slack—as possible. In analogy to the slack definition from Section 2.4.8 the arc slack $\mathrm{slk}(v,w)$ for given T and arrival times $a : V(G^{\mathcal{R}}) \to \mathbb{R}$ is defined as $\mathrm{slk}(v,w) := \mathrm{at}(w) - \mathrm{at}(v) - \vartheta(v,w) + T$ for every arc $(v,w) \in E(G^{\mathcal{R}})$. Note that $(\mathrm{at}(w) + T)$ defines a required arrival time for signals entering latch w. It turns out that maximizing the worst slack by clock skew scheduling is equivalent to minimizing the cycle time:

Lemma 5.2. *Let $G^{\mathcal{R}}$ be a simple register graph with arc delays d. Let T' be the minimum possible cycle time for $(G^{\mathcal{R}}, d)$, $T > 0$, and*

$$\mathrm{slk}'(T) = \max_{\mathrm{at}} \min_{(v,w)\in E(G^{\mathcal{R}})} \Big(\mathrm{at}(w) - \mathrm{at}(v) - \vartheta(v,w) + T\Big)$$

be the maximum achievable worst slack for the cycle time T. Then

$$T' = T - \mathrm{slk}'(T).$$

Proof. Analogously to the proof of Theorem 5.1, $\mathrm{slk}'(T)$ is the maximum value such that

$$\sum_{(v,w)\in E(C)} (T - \vartheta(v,w) - \mathrm{slk}'(T)) \geq 0 \quad \forall \text{ cycles } C \subseteq G^{\mathcal{R}}$$

$$\Leftrightarrow$$

$$T - \mathrm{slk}'(T) \geq \frac{\sum_{(v,w)\in E(C)} \vartheta(v,w)}{|E(C)|} \qquad \forall \text{ cycles } C \subseteq G^{\mathcal{R}}.$$

This shows that the minimum possible cycle time T' is given by $T' = T - \mathrm{slk}'(T)$. □

5.1 Previous Work

Clock skew scheduling was first considered by Fishburn [1990]. He formulated the problem of finding a schedule that minimizes the cycle-time as a linear program.

Sakallah et al. [1990] proposed a similar linear program, and furthermore proposed an iterative algorithm for timing verification, which turned out to be a variant of the Moore-Bellman-Ford Algorithm (see Korte and Vygen [2008], Chapter 7).

The correspondence between the existence of a feasible clock skew schedule and non-positive delay cycles in the register graph was first mentioned by Szymanski [1992] and Shenoy et al. [1992]. Both approaches, as well as Deokar and Sapatnekar [1995], apply binary search on T combined with a longest path computations to decide whether the current T is feasible. All of them also consider early mode constraints. They differ in slightly modified ways to check the feasibility of a cycle time T.

Albrecht et al. [1999, 2002] minimize T, and additionally optimize the overall slack distribution of late and early mode path constraints. As early mode constraints can be solved by adding delay buffers, they first optimize only late and then early mode constraints while late mode constraints are not worsened below a late slack target. They do not only compute singular feasible clock arrival times for each register but they determine a maximum feasible time window. This window allows more freedom during clocktree construction. They apply the strongly polynomial minimum balance algorithm by Young et al. [1991] on the register graph.

This approach was modified in Held [2001] to work directly on the signal graph instead of the register graph. The advantage of the signal graph model is its size, and the simpler inclusion of constraints such as user defined tests. In Held et al. [2003] the efficiency for the application on real ASICs was demonstrated.

Albrecht [2006] proposes a modified version of Albrecht et al. [1999, 2002], which creates the register graph incrementally on critical paths. It has the same worst case running time but tends to be much faster in practice.

Another approach to optimize the slack distribution was given by Kourtev and Friedman [1999]. They formulate a quadratic program that minimizes the sum of quadratic slacks. Early and late mode slacks are treated equally. The worst slack, which limits the cycle time and, therefore, the speed of the chip, is not necessarily maximized by this approach. For instances which consist only of transparent latches, the worst slack is limited by the longest average delay cycle in the (cyclic) propagation graph, independently of the actual register switching times. Here this approach is well-suited to optimize the robustness.

Rosdi and Takahashi [2004] considered clock skew scheduling under the presence of multiple clock domains and multi-cycle paths. They solved this problem by binary search of the smallest feasible reference cycle time. The approach was extended in Takahashi [2006] to minimize the estimated realization cost of the tree as a postoptimization step. The cost of a schedule is measured as the sum of differences between the scheduled clock latencies and some predefined target latencies, which could, for instance, refer to a zero skew realization.

Finally, we mention the work by Ravindran et al. [2003], who consider clock skew scheduling under the additional constraint that only a discrete set of large latency differences is realizable. They use a mixed integer linear programming formulation to split the clock domain into a set of discrete latency classes.

In Section 5.2 we describe traditional approaches by Albrecht et al. [1999], Held [2001], and Held et al. [2003] based on combinatorial algorithms. These results will also be used later in Chapter 6. In Section 5.2.3, we apply the Algorithms from Section 5.2 to obtain the first known strongly polynomial time algorithm for cycle time minimization on instances with multiple clock domains and multi-cycle paths. In Section 5.3 an iterative algorithm is described that was known for long by electrical engineers as a clock skew scheduling heuristic. We give the first proof that the iterative algorithm maximizes the worst late slack within an arbitrary accuracy $\epsilon > 0$ if the clock latencies are bound to some closed interval. In Section 5.3.5 we enhance the iterative approach to a *hybrid* approach that approximates an optimum slack distribution with arbitrary accuracy.

As all delays, slews, and arrival times that are computed during static timing analysis are of limited accuracy such an arbitrary accuracy result is sufficient in practice. Moreover, it turns out that it is much faster than the combinatorial algorithms and can absorb delay variations due to scheduling by incremental delay recalculations. Such delay variations occur, because varying clock arrival times lead to varying signals that determine which slews are propagated (see Section 2.4.4).

5.2 Slack Balance Problem

The primary task of clock skew scheduling is to find clock input arrival times at the registers, that maximize the worst slack. But also less critical slacks should be high to anticipate delay uncertainties, for example due to placement legalization, routing detours. With other words we want to find a schedule that yields a leximin maximal slack ordering.

The mathematical problem formulation that solves this problem is the Slack Balance Problem. We extend definitions from Held [2001] and Vygen [2001, 2006] by a slack target. Slacks above this target are sufficiently large and not further distinguished. The edge set $E(G)$ is divided into a subset $\mathrm{E_p}$, where a non-negative slack is a hard requirement, and a subset $\mathrm{E_t}$, where slack is to be distributed. Furthermore, slacks are weighted by positive real numbers, and the edge set $\mathrm{E_t}$ is partitioned into sets, where only the most critical slack within each partition is of interest:

Slack Balance Problem

Input:

- a directed graph G with edge costs $c : E(G) \to \mathbb{R}$
- a set $\mathrm{E_p} \subseteq E(G)$ such that $(V(G), \mathrm{E_p}, c)$ is conservative
- a partition $\mathcal{F}$ of $\mathrm{E_t} := E(G) \setminus \mathrm{E_p}$
- weights $w : \mathrm{E_t} \to \mathbb{R}_+$
- a slack target $\mathrm{S^{tgt}}$

Output: A node potential $\pi : V(G) \to \mathbb{R}$ with $c_\pi(e) := c(e) + \pi(x) - \pi(y) \geq 0$ for $e = (x,y) \in \mathrm{E_p}$ such that the vector

$$\left(\min\left\{\mathrm{S^{tgt}}, \min\left\{\frac{c(e) + \pi(x) - \pi(y)}{w(e)} \mid e = (x,y) \in F\right\}\right\}\right)_{F \in \mathcal{F}}$$

is leximin maximal with respect to $\mathrm{S^{tgt}}$.

Special cases of the Slack Balance Problem are the classic Minimum Ratio Cycle Problem ($\mathcal{F} = \{E(G)\}$), with a further specialization the Minimum Mean Cycle Problem ($\mathcal{F} = \{E(G)\}$, $w \equiv 1$), and the Minimum Balance Problem ($\mathcal{F} = \{\{e_1\}, \{e_2\}, \ldots, \{e_{|E(G)|}\}\}$, $w \equiv 1$) by Schneider and Schneider [1991] ($\mathrm{S^{tgt}} = \infty$ for all special cases).

The relation to the Minimum Balance Problem can be seen by the following theorem. It generalizes formulations from Albrecht [2001] (special case: $\mathrm{S^{tgt}} = \infty, \mathcal{F} = \{\{e\} \mid e \in \mathrm{E_t} = E(G)\}$, and $w \equiv 1$) and Vygen [2001] (special case: $\mathrm{S^{tgt}} = \infty$):

Theorem 5.3.
Let $(G, c, w, \mathcal{F}, \mathrm{S^{tgt}})$ *be an instance of the* Slack Balance Problem*, let* $\pi : V(G) \to \mathbb{R}$ *with* $c_\pi(e) \geq 0$ *for* $e \in \mathrm{E_p}$*, and let*

$$E_\pi := \left\{ f \in F \in \mathcal{F} \mid \frac{c_\pi(f)}{w(f)} = \min\left\{\frac{c_\pi(e)}{w(e)} \mid e \in F\right\} \leq \mathrm{S^{tgt}} \right\}.$$

Then π *is an optimum solution of the* Slack Balance Problem *with* $|E_\pi|$ *minimum if and only if for each* $X \subset V(G)$*, with* $E_\pi \cap \delta^-(X) \neq \emptyset$*:*

$$\begin{aligned} &\min_{e \in E_\pi \cap \delta^-(X)} \frac{c_\pi(e)}{w(e)} &&\geq \min\nolimits_{e \in E_\pi \cap \delta^+(X)} \tfrac{c_\pi(e)}{w(e)} \\ \textit{or} \quad &\min_{e \in \delta^+(X) \cap \mathrm{E_p}} c_\pi(e) &&= 0, \end{aligned} \tag{5.2}$$

where $\min \emptyset := \infty$*. Furthermore, the sets* E_π *and the vector* $(c_\pi(e))_{e \in E_\pi}$ *are identical for all optimum solutions* π *for which* $|E_\pi|$ *is minimum.*

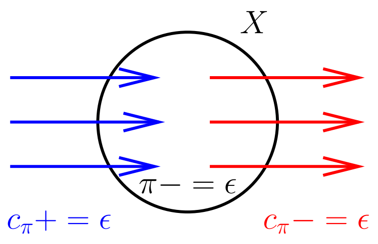


Figure 5.2: Reducing π within X.

Proof. Any optimum solution π of the Slack Balance Problem with minimum $|E_\pi|$ satisfies (5.2). Otherwise we could decrease $\pi(v)$ for all $v \in X$ by a sufficiently small ϵ (see Figure 5.2). Thereby, $c_\pi(e)$ for $e \in \delta^-(X)$ (blue edges) would be increased, and either a better solution would be found or at least one edge could be removed from $E_\pi \cap \delta^-(X)$ without adding an edge to $E_\pi \cap \delta^+(X)$.

To show the reverse direction, let $\pi, \pi' : V(G) \to \mathbb{R}$ be two vectors, with $c_\pi(e), c_{\pi'}(e) \geq 0$ for $e \in \mathrm{E_p}$, that satisfy (5.2). It suffices to show that $c_\pi(e) = c_{\pi'}(e)$ for all $e \in E_\pi \cup E_{\pi'}$.

Suppose there exists an edge $f = (x, y) \in E_\pi \cup E_{\pi'}$ (which implies $\mathrm{S^{tgt}} \geq \min\{\frac{c_\pi(f)}{w(f)}, \frac{c_{\pi'}(f)}{w(f)}\}$) with $\pi(y) - \pi(x) \neq \pi'(y) - \pi'(x)$. Among such edges, we choose an f that minimizes $\min\{\frac{c_\pi(f)}{w(f)}, \frac{c_{\pi'}(f)}{w(f)}\}$. Without loss of generality $c_\pi(f) < c_{\pi'}(f)$ or equivalently $\pi(y) - \pi(x) > \pi'(y) - \pi'(x)$.

This implies $f \in E_\pi$. Otherwise, if $f \notin E_\pi$, let $F \in \mathcal{F}$ be the partition set with $f \in F$. As $\frac{c_\pi(f)}{w(f)} \leq \mathrm{S^{tgt}}$, there must be an $f' \in F$ with $\frac{c_\pi(f')}{w(f')} < \frac{c_\pi(f)}{w(f)}$. Now, by the choice of f, $\frac{c_{\pi'}(f')}{w(f')} = \frac{c_\pi(f')}{w(f')} < \frac{c_\pi(f)}{w(f)} < \frac{c_{\pi'}(f)}{w(f)}$, and thus, $f \notin E_{\pi'}$, which contradicts our choice of $f \in E_\pi \cup E_{\pi'}$.

Let Q be the set of vertices that are reachable from y via edges in $\{e \in \mathrm{E_p} \mid c_\pi(e) = 0\} \cup \{e \in E_\pi \mid \frac{c_\pi(e)}{w(e)} \leq \frac{c_\pi(f)}{w(f)}\}$. As π fulfills (5.2) (with $X = Q$), we have $x \in Q$. Accordingly, there exists a y-x-path P in G that consists of edges $e \in \mathrm{E_p} \cap E(P)$ with $c_\pi(e) = 0 \leq c_{\pi'}(e)$ and edges $e \in E_\pi \cap E(P)$ with $\frac{c_\pi(e)}{w(e)} \leq \frac{c_{\pi'}(e)}{w(e)}$. Summation of the reduced costs over all edges in $E(P)$ yields $\pi(y) - \pi(x) \leq \pi'(y) - \pi'(x)$, a contradiction.

□

Remark 5.4. *A slack balance problem $(G, c, w, \mathcal{F}, \mathrm{S^{tgt}})$ can be transformed into an equivalent problem with slack target 0. By modifying the costs on all test edges $e \in \mathrm{E_t}$ by $c'(e) := c(e) - w(e) \cdot \mathrm{S^{tgt}}$, and using the original costs $c'(e) := c(e)$ for all $e \in \mathrm{E_p}$, the modified slack balance problem $(G, c', w, \mathcal{F}, 0)$ with a slack target of 0 is equivalent to $(G, c, w, \mathcal{F}, \mathrm{S^{tgt}})$.*

We generalize the classic Minimum Ratio Cycle Problem as follows:

Minimum Ratio Cycle Problem

Input:

- a directed graph G with edge costs $c : E(G) \to \mathbb{R}$
- a set $\mathrm{E_p} \subseteq E(G)$ such that $(V(G), \mathrm{E_p}, c)$ is conservative
- weights $w : \mathrm{E_t} \to \mathbb{R}_+$ with $\mathrm{E_t} = E(G) \setminus \mathrm{E_p}$

Output:

$$\min \left\{ \frac{c(C)}{w(C)} \mid C \subseteq G \text{ is a cycle with } w(C) > 0 \right\},$$

$$\text{where } c(C) := \sum_{e \in E(C)} c(e) \text{ and } w(C) := \sum_{e \in \mathrm{E_t} \cap E(C)} w(e).$$

The Minimum Ratio Cycle Problem is an important subtask when solving the Slack Balance Problem. It can be solved in strongly polynomial time:

Theorem 5.5 (Held [2001]).
The Minimum Ratio Cycle Problem *can be solved in strongly polynomial time*

$$O(n^3 \log n + \min\{nm, n^3\} \cdot \log^2 n \log\log n + nm \log m),$$

or in pseudo-polynomial time

$$O(w_{\max}(nm + n^2 \log n))$$

for integral weights $w : E(G) \to \mathbb{N}$ with $w_{\max} := \max\{w(e) \mid e \in \mathrm{E_t}\}$.

Proof. Here, we only point to the underlying core algorithms. Details can be found in Held [2001].

The running time of $O(n^3 \log n + \min\{nm, n^3\} \cdot \log^2 n \log\log n + nm \log m)$ is achieved by variants of Megiddo's strongly polynomial minimum ratio cycle algorithm for simple graphs (Megiddo [1983]). The last term $O(nm \log m)$ is needed to reduce a general graph to a simple one. It is only relevant for the running time if the number of weighted edges is not polynomially bounded in the number of vertices, as otherwise $O(\log m) = O(\log n)$ if $m \leq O(n^q)$ for a $q \in \mathbb{N}$.

The running time of $O(w_{\max}(nm + n^2 \log n))$ is obtained by the parametric shortest path algorithm of Young et al. [1991], which was introduced for $\mathrm{E_p} = \emptyset$ and $w \equiv 1$, but works also in the generalized case.

□

A key task for solving the Slack Balance Problem is to maximize the worst weighted reduced cost. This can be done by a minimum ratio cycle computation:

Lemma 5.6. *Given an instance of the* Minimum Ratio Cycle Problem*, the value $\lambda^\star$ of the minimum ratio cycle equals*

$$\max_{\substack{\pi : V(G) \to \mathbb{R} \\ c_\pi(e) \geq 0\ \forall e \in \mathrm{E_p}}} \min_{e \in \mathrm{E_t}} \frac{c_\pi(e)}{w(e)}. \tag{5.3}$$

Proof. The proof works analogously to the proof of Theorem 5.1. For $\lambda \in \mathbb{R}$, define edge costs $c_\lambda : E(G) \to \mathbb{R}$ by $c_\lambda(e) = c(e)$ if $e \in \mathrm{E_p}$, and $c_\lambda(e) = c(e) - \lambda \cdot w(e)$ if $e \in \mathrm{E_t}$. The value of (5.3) is given by the maximum λ such that (G, c_λ) has a feasible potential. A feasible potential for (G, c_λ) exists if and only if

$$\sum_{e \in E(C)} c_\lambda(e) \geq 0 \qquad \text{for all cycles } C \subseteq G$$

$$\Leftrightarrow$$

$$\sum_{e \in \mathrm{E_p} \cap E(C)} c(e) + \sum_{e \in \mathrm{E_t} \cap E(C)} (c(e) - \lambda w(e)) \geq 0 \qquad \text{for all cycles } C \subseteq G$$

$$\Leftrightarrow$$

$$\tfrac{c(C)}{w(C)} \geq \lambda,$$

for all cycles $C \subseteq G$ with $w(C) > 0$. The last equivalence holds because $(V(G), \mathrm{E_p}, c)$ is conservative.

□

The next theorem generalizes a result from Held [2001] and shows how the Slack Balance Problem can be solved.

Theorem 5.7. *The* Slack Balance Problem *can be solved in strongly polynomial time*

$$O(I \cdot (n^3 \log n + \min\{nm, n^3\} \cdot \log^2 n \log\log n + nm \log m)),$$

where $I := n + |\{F \in \mathcal{F} \; ; \; |F| > 1\}|$, *or in pseudo-polynomial time*

$$O(w_{\max}(nm + n^2 \log n) + I(n \log n + m))$$

for integral weights $w : E(G) \to \mathbb{N}$ *with* $w_{\max} := \max\{w(e) | e \in \mathrm{E_t}\}$.

Proof. Certainly, the overall worst weighted slack must be maximized in an optimum slack distribution. Then, keeping the worst reduced cost unchanged, the second worst reduced cost must be maximized, and so forth. By Lemma 5.6, the maximum possible minimum weighted slack can be determined by a minimum ratio cycle computation.

The slack balance algorithm (Algorithm 7) iteratively determines a minimum ratio cycle $C \subseteq G$, which defines the maximum possible minimum weighted reduced cost $\lambda^\star := \frac{c(C)}{w(C)}$. This will also be the final minimum reduced costs for all involved partitions $F \in \mathcal{F}$, with $F \cap E(C) \neq \emptyset$. To obtain a leximin maximal solution, no weighted reduced cost of any edge from such a partition F needs to be increased above $\lambda^\star$. Instead, the weighted reduced costs $\lambda^\star$ can simply be guaranteed in further iterations by modifying their costs to

$$c(e) \leftarrow c(e) - \lambda^\star w(e), \tag{5.4}$$

and canceling the weights:

$$\mathrm{E_t} := \mathrm{E_t} \setminus F \text{ and } \mathcal{F} := \mathcal{F} \setminus \{F\}. \tag{5.5}$$

Algorithm 7 Slack Balance Algorithm

```
1: repeat
2:     Compute Minimum Ratio Cycle $C$ in $(G, c, w)$;
3:     $\lambda^\star \leftarrow \frac{c(C)}{w(C)}$;
4:     for $F \in \mathcal{F}$ with $E(C) \cap F \neq \emptyset$ do
5:         for $f \in F \setminus E(C)$ do
6:             $c(f) \leftarrow c(f) - \lambda^\star w(f)$;
7:             $w(f) \leftarrow 0$;
8:         end for
9:         $\mathcal{F} \leftarrow \mathcal{F} \setminus F$;
10:    end for
11:    Contract $C$; adapt costs according to (5.6),(5.7), and remove irrelevant loops;
12: until $\mathcal{F} = \emptyset$ or $\lambda^\star \geq \mathrm{S}^{\mathrm{tgt}}$
```

To reduce the graph size for the next iteration, the minimum ratio cycle C can be contracted to a new node $v^\star$ (line 11), after anticipating the costs of any path through C by modifying incident edge costs:

$$c(x,y) \leftarrow c(x,y) - \pi(y) \qquad \text{for all } (x,y) \in \delta^-(V(C)), \text{ and} \tag{5.6}$$

$$c(x,y) \leftarrow c(x,y) + \pi(x) \qquad \text{for all } (x,y) \in \delta^+(V(C)). \tag{5.7}$$

This contraction is dispensable if C consists of a loop ($C = (v, (v,v))$). After the contraction, there may be loops in G. If such a loop $e = (v,v) \in E(G)$ fulfills one of the following three conditions, (i) $e \in \mathrm{E_p}$, (ii) $c_\pi(e) = \lambda^\star$, or (iii) $\{e\} \in \mathcal{F}$, the contribution of e to the global slack distribution is fixed and the loop e can be removed. We call such loops *irrelevant*. Note, in the case $c_\pi(e) = \lambda^\star$ and $e \in F \in \mathcal{F}$ with $|F| > 1$, lines 5–8 must of course be applied to F when removing e.

Otherwise, if a loop consists of an edge $e \in F \in \mathcal{F}$, $|F| > 1$, and $c_\pi(e) > \lambda^\star$, e might define the minimum reduced cost within F, and must remain in $E(G)$.

The running time of the cost adjustment plus the contraction (lines 4–11) is obviously $O(n+m)$. The next iteration starts with computing a new minimum ratio cycle in the modified graph.

In each iteration either a cycle with several nodes is contracted into a new node, or a loop consisting of an edge $e \in F \in \mathcal{F}$, $|F| > 1$, determines the current minimum ratio cycle. In the first case, the number of nodes is reduced by one, because at least two nodes are replaced by a new one. In the second case, the number $|\{F \in \mathcal{F}\ ;\ |F| > 1\}|$ is decreased by one.

Thus, there will be at most $I := n + |\{F \in \mathcal{F}\ ;\ |F| > 1\}|$ iterations of the outer loop. Applying Theorem 5.5 yields the desired strongly polynomial running time.

In the pseudo-polynomial case we apply the parametric shortest path algorithm of Young et al. [1991]. This algorithm computes a (finite) sequence $(T_0, \lambda_0), (T_1, \lambda_1), \ldots,$ (T_K, λ_K) of trees T_i and increasing $\lambda_i \in \mathbb{R}$ $(0 \leq i \leq K)$, such that T_i $(0 \leq i < K)$ is a shortest path tree with respect to edge costs c_λ for all $\lambda_i \leq \lambda \leq \lambda_{i+1}$, where c_λ

is defined as in the proof of Lemma 5.6. Whenever (G, c_{λ_i}) contains a zero length cycle, this is the next minimum ratio cycle.

The clue is that the tree remains a shortest path tree in the modified and contracted graph. Thus, the algorithm can be continued after updating some edge labels, vertex labels in the subtrees of contracted or modified edges and vertices, and updating a heap, which provides the minimum vertex label. These updates take $O(m + n \log n)$ per iteration. For details see Held [2001], or similarly Young et al. [1991].

In the worst case analysis, the bound for the total running time equals that of a single minimum ratio cycle computation plus $O(I(m + n \log n))$ for data structure updates in the parametric shortest path tree through all iterations.

$\square$

For small weights, especially $w_{\max} = 1$ the pseudo-polynomial algorithm yields the best known bound for the SLACK BALANCE PROBLEM. In an empirical study Dasdan et al. [1999] determined Howard's algorithm, to be the fastest choice for practical minimum ratio cycle computations, although only an exponential worst case running time bound is known for this algorithm. A detailed description of Howard's algorithm can be found in Cochet-Terrasson et al. [1998].

We observed in Held [2001] that the parametric shortest path algorithm of Young et al. [1991] yields comparable running times on sparse graphs but much faster running times on dense graphs.

Remark 5.8. *In some scenarios the* SLACK BALANCE PROBLEM *is extended to allow for negative weights* $w : \mathrm{E_t} \to \mathbb{R}$. *This complicates the computation of the initial minimum ratio cycle. The two minimum ratio cycle algorithms from Theorem 5.5 rely on the fact that a lower bound* $\underline{\lambda} \in \mathbb{R}$ *for the value* $\lambda^\star$ *of the minimum ratio cycle such that* $(G, c - \underline{\lambda} w)$ *is conservative is known or can be computed efficiently.*

If the total weight of each cycle $C \in G$ *is non-negative,* $w(E(C)) \geq 0$, *any* $\underline{\lambda} \leq \lambda^\star$ *fulfills this requirement. This is especially given if all weights are non-negative. Without this constraint on the total weight of all cycles, the problem of finding* $\underline{\lambda}$ *becomes very difficult. However, both algorithms can be used if a feasible lower bound* $\underline{\lambda}$ *is given.*

Remark 5.9. *(Cycles in the Propagation Graph)*

The slack balance algorithm (Algorithm 7) can be used to analyze the timing constraints in the presence of cycles in the propagation graph. Recall that the signal propagation in static timing analysis, as described in Section 2.4.4, requires an acyclic propagation graph. Such cycles can be induced only by transparent latches if these latches are analyzed with non-conservative constraints according to Remark 2.1.

Define a slack balance instance by the signal graph $G^{\mathfrak{S}} = (V^{\mathfrak{S}}, E^{\mathfrak{S}})$ *with edge costs* $c^{\mathfrak{S}}$ *from Section 2.4.9. Choose the test edges* $\mathrm{E_t}$ *as the signal graph test edges* $E_t^{\mathfrak{S}}$ *plus the set of flush propagation edges in* $E^{\mathfrak{S}}$. *Furthermore, set* $\mathcal{F} := \Big\{\{e\} \mid e \in \mathrm{E_t}\Big\}$, $\mathrm{E_p} = E^{\mathfrak{S}} \setminus \mathrm{E_t}$, *and* $w : \mathrm{E_t} \to \{1\}$.

Now the maximum possible worst reduced cost of a test edge from $\mathrm{E_t}$ *defines the worst slack in the design. Required arrival times can be propagated backwards, as usual, along the now acyclic propagation graph. This model ignores varying delays, when arrival times are re-scheduled and, therefore, slews are propagated differently. In practice it is sufficient to iterate delay-calculation and slack balancing for a few iterations.*

Note that we need not add all flush edges to $\mathrm{E_t}$. *Actually, we only need to find a feedback arc set within the flush edges, so that* $(V^{\mathfrak{S}}, \mathrm{E_p})$ *becomes acyclic. However, finding a feedback arc set of minimum size is NP-hard (Karp [1972]), but heuristics could be used to choose a small feedback arc set.*

5.2.1 Graph Models

In this section two models for clock skew scheduling as a Slack Balance Problem are presented. The timing constraints are either modeled by a *register graph* (Albrecht et al. [2002]) similar to the introductory example, or by a slightly modified *signal graph* (Held et al. [2003]), which was defined in Section 2.4.9. We modify the register graph model such that it can handle multi-frequency trees, where signals of different domains may arrive at a single clocktree sink. Based on this we invent a method to optimize the reference cycle time on a multi-frequency chip, with multi-cycle adjusts.

Modeling Clocktree Sinks

The clocktree sink model is the same for both graph models. It models implicit timing constraints between the clock signals arriving at a register, or a set of coupled registers, and bounds on feasible schedule values.

Definition 5.10. *The input to clock skew scheduling consists of*

- *a timing graph* $G^{\mathcal{T}}$ *on a netlist* $(\mathcal{C}, P, \gamma, \mathcal{N})$ *containing all information needed for static timing analysis,*
- *a set* $P_{cs} \subset P$ *of clocktree sinks, and*
- *lower and upper bounds,* $\underline{\pi}(p,x) \in \mathbb{R} \cup \{\infty\}$ *and* $\overline{\pi}(p,x) \in \mathbb{R} \cup \{\infty\}$, *for the signal arrival time* $\mathrm{at}(p,\sigma)$, *of a signal starting at a signal source* $x \in V^{\mathcal{T}^{start}}$, *and arriving in* $p \in P_{cs}$*:*

$$\underline{\pi}(p,x) \leq \mathrm{at}(p,\sigma) \leq \overline{\pi}(p,x) \tag{5.8}$$

for all $p \in P_{cs}, \sigma = (x,\eta,\zeta) \in \mathfrak{S}(p), \eta \in \{\text{early}, \text{late}\}, \zeta \in \{\text{rise}, \text{fall}\}$.

The elements from P_{cs} might belong to different trees, usually these are the clock inputs of the registers. Given a clocktree sink $p \in P_{cs}$, we add all vertices from the signal graph $G^{\mathfrak{S}}$ (see Section 2.4.9) that represent signals in $\mathfrak{S}(p)$. Restated, a vertex v_σ is added for each signal $\sigma \in \mathfrak{S}(p)$. The set of these vertices is named V_{cs}.

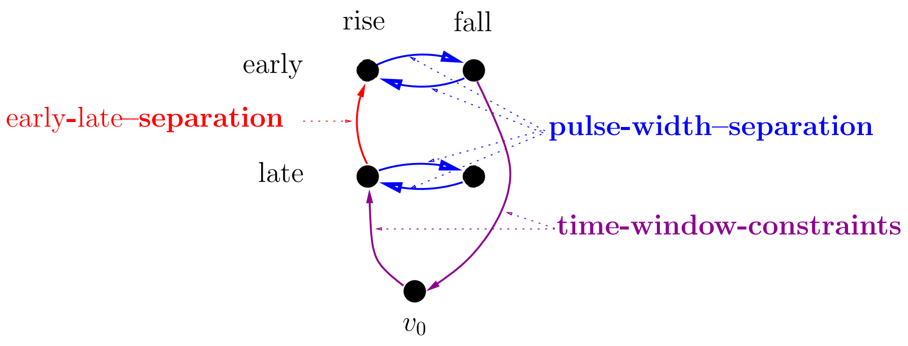


Figure 5.3: Graph model for clock sink signals.

Let now $(x, \text{early}, \text{rise}), (x, \text{late}, \text{rise}), (x, \text{early}, \text{fall}),\ (x, \text{late}, \text{fall}) \in \mathfrak{S}(p)$ be four signals with a common origin $x \in V^{\mathcal{T}^{start}}$. There are two types of implicit constraints for these four signals. First, early mode arrival times must not be higher than late mode arrival times:

$$\text{at}(p, (x, \text{early}, \text{rise})) \le \text{at}(p, (x, \text{late}, \text{rise})) \tag{5.9}$$

and

$$\text{at}(p, (x, \text{early}, \text{fall})) \le \text{at}(p, (x, \text{late}, \text{fall})). \tag{5.10}$$

Second, the clock signal arrival times represent an oscillating clock signal. Thus the given *pulse-width* $\text{pw}(x)$ between rising and falling edge (usually half the cycle time) must be maintained:

$$\text{at}(p, (x, \text{early}, \text{rise})) + \text{pw}(x) = \text{at}(p, (x, \text{early}, \text{fall})) \tag{5.11}$$

and

$$\text{at}(p, (x, \text{late}, \text{rise})) + \text{pw}(x) = \text{at}(p, (x, \text{late}, \text{fall})). \tag{5.12}$$

These constraints can be modeled by adding edges with adequate costs between the representing clock nodes as in Figure 5.3. Note, only a single early-late-separation edge is needed, because the pulse-width-separation implies that the other inequality is fulfilled. In the signal graph these constraints are given implicitly by the asserted clock arrival times at their start nodes in $V^{\mathcal{T}^{start}}$ and the propagation inequalities on the path to the clocktree sinks. We denote the set of all pulse-width-separation edges for all clocktree sinks by E_ς and the set of all early-late-separation edges by E_η.

At a clocktree sink $p \in P_{cs}$, clock signals $\sigma, \sigma' \in \mathfrak{S}(p)$ with different origins $x, x' \in V^{\mathcal{T}^{start}}$ may arrive. For example if the register is driven with varying clock frequencies. Multiplexers within the clock network determine which of the different clock signals is propagated to the sink p, when the chip is operating.

Usually these multiplexers are located at the root level of the clock network and different signals are propagated through a common physical clocktree. In this case the clocktree can only realize a single schedule for the sink p, and all clock signals of different origins, but equal transition and equal timing mode, must be scheduled equally.

Theoretically the multiplexers could be placed directly in front of each register. In this case there could be an individual tree and an individual schedule for each clock signal entering p. However, such a resource consumptive design is hardly applied.

In addition, there might be different clocktree sinks $p, p' \in P_{cs}$ that must be scheduled synchronously, for example the master and slave clock input pins of a master-slave-latch, which must receive exactly inverse signals. Such fixed time differences can be modeled by introducing two arcs between the corresponding vertices as for the pulse-width-separation.

As an additional constraint a *maximally allowed time window* $[\underline{\pi}(v), \overline{\pi}(v)]$ is given for each clock sink node v. Recall that v represents a pin signal pair (p, σ) with signal source x, such that $\underline{\pi}(v)$ is given by $\underline{\pi}(p, x)$ and $\overline{\pi}(v)$ is given by $\overline{\pi}(p, x)$ from Definition 5.10. Its purpose is to bound the maximum difference between earliest and latest clock sink arrival times in a clocktree, and indirectly the power consumption of the clocktree. The constraints $\underline{\pi}(v) \leq \pi(v) \leq \overline{\pi}(v)$ can be modeled by using an extra vertex $v_0 \in G^{\mathcal{R}}$ that represents the central time 0, and adding edges (v, v_0), with edge cost $c(v, v_0) = -\underline{\pi}(v)$, and (v_0, v), with edge cost $c(v_0, v) = \overline{\pi}(v)$, as indicated by the magenta edges in Figure 5.3. The set of these time-window edges is denoted E_w. Summarizing, there are

1. early-late separation edges,
2. cluster edges ensuring fixed arrival time differences between signals, and
3. maximally allowed time window edges.

A set $C \subseteq P_{cs}$ that is connected by such clock constraint edges is called a *clock sink cluster*. We assume that all intervals $[\underline{\pi}(v), \overline{\pi}(v)]$ preserve feasibility. With other words, there are no negative cycles in the subgraphs modeling clock sink constraints.

Remark 5.11 (Clock Sink Contraction).
Cluster edges within each clock sink cluster (2. item in the above list), like pulse-width constraints (5.11) and (5.12), induce zero-length cycles that can be contracted once the slack balance graph construction is completed, similar to critical cycles in Algorithm 7, and before any slack balance algorithm starts running. Costs of incoming and outgoing edges in the slack balance graph need to be modified as in (5.6) and (5.7).

After this contraction a clock sink cluster is represented by two nodes v_e and v_l. The node v_l, representing latest signals, and v_e, representing earliest signals, are connected by an edge (v_l, v_e).

Signal Graph Model

In the signal graph model the weighted signal graph $(G^{\mathfrak{S}}, c^{\mathfrak{S}})$ from Section 2.4.9 is expanded by the weighted clock sink edges from the last subsection. Furthermore, propagation edges in $E_p^{\mathfrak{S}}$ representing timing graph edges that enter clocktree sinks are removed because they would predefine the schedule. More precisely, for a node $v \in V^{\mathfrak{S}}$ that represents a late clock signal at a clocktree sink, we remove all propagation edges from $\delta^+_{(V^{\mathfrak{S}}, E_p^{\mathfrak{S}})}(v)$. Analogously, we remove all propagation edges from $\delta^-_{(V^{\mathfrak{S}}, E_p^{\mathfrak{S}})}(v)$, if v represents an early clock signal at a sink. Recall that late mode propagation edges in $G^{\mathfrak{S}}$ are reverse to the signal direction. Without removal, the clock sink signal arrival times would be predetermined by the signal propagation constraints within the clock network.

Remark 5.12 (Clock Gating)**.**
By the removal of the above edges, clock gating constraints will be hidden. Clock gates are used to switch off parts of the clocktree and the combinatorial logic to safe power, while the chip is operating. A clock gate can be realized by an AND-gate, with the clock signal entering one input and the gating signal entering the other input. If the gating (input) voltage is low the subsequent clocktree is switched off. If the gating voltage is high, the clock signal determines the output and the clocktree is operating. Timing tests ensure that the gating signal arrives while the clock signal is low, to keep a potentially high output signal stable within the current cycle. The problem is that these gating tests can be evaluated only after the clocktree is inserted.

The register launching the gating signal and the gated registers must be scheduled such that a feasible realization of the clocktrees behind the gates and the gating tree, which distributes the gating signal to the clock gates, can be found. That is, if a gated register is scheduled to a very early clock arrival time and the gating register to a very late one, it might be impossible to construct the gated tree and the gating signal tree such that the gating constraints can be preserved.

If a sufficiently large estimate for the expected sum of the two latencies is known, user defined tests, with the setup time set to the estimated sum of latencies, could be inserted between the output pin of the register, which is launching the gating signal, and the registers, which are gated. During clock skew scheduling, these tests reserve time for the latencies of the two trees.

Register Graph Model

The *register graph model* $G^{\mathcal{R}}$ (Albrecht et al. [2002]) filters longest and shortest paths between any pair of registers, as in our introductory example. In contrast to the simple register graph from the introduction, we describe a model that considers also early mode slacks and can apply final time window optimization. Furthermore we extend the model in Albrecht et al. [2002] by user defined tests and allowing multiple clock frequencies.

As for the signal graph model, clock sink vertices and edges are inserted into $G^{\mathcal{R}}$, according to Section 5.2.1. Then, additional edges representing data paths between clocktree sinks and primary IO-pins are added. Here, we must be aware that arrival time tests, especially user defined tests, can be performed anywhere between two data signals. Therefore, the construction of data path edges must be performed carefully.

For every test edge $(v_1, v_2) \in E_t^{\mathfrak{S}}$ and every two nodes

$$c_1, c_2 \in V_{cs} \cup \{v_0\}$$

such that there is a c_1-v_1-path and a v_2-c_2-path in $(V^{\mathfrak{S}}, E_p^{\mathfrak{S}})$ without internal vertices in V_{cs}, we insert a test edge (c_1, c_2) into $E_t^{\mathcal{R}}$. Here we assume that no path contains a clocktree sink as an internal vertex, that is, the signal graph was prepared as for the register graph model. Its edge cost are chosen as

$$c^{\mathcal{R}}(c_1, c_2) \quad := \quad \underline{c}^{\mathfrak{S}}(v_2, c_2) + \underline{c}^{\mathfrak{S}}(c_1, v_1) + c^{\mathfrak{S}}(v_1, v_2), \tag{5.13}$$

where $\underline{c}^{\mathfrak{S}}(v, w)$ denotes the length of a shortest v-w-path in the $(G^{\mathfrak{S}}, c^{\mathfrak{S}})$. The signal graph was used to construct the register graph only for simpler notation. In practice the register graph could be created directly. For instance, if (v_1, v_2) represents a setup test, (5.13) will correspond to the sum of the delay of a longest late-mode delay path from c_2 to v_2 and setup and adjust times of the underlying timing test. If the setup test occurs at a simple latch or flip-flop, the c_1-v_1-path will be of length zero.

Note that by construction each edge that arises from (5.13), represents exactly one timing test. But each timing test can be represented by many edges in the register graph.

5.2.2 Late, Early and Window Optimization

Given a slack balance graph (G, c), which can either be constructed according to the signal graph model or the register graph model, Algorithm 7 on page 90 could be applied immediately, with E_t as the set of edges that represent a setup test, hold test, user defined test, or predefined required arrival time test.

However, timing tests and slacks are of different relevance. For instance, an early mode slack can still be removed after clock skew scheduling by increasing the delay on the data path. This process of inserting delay to eliminate early mode problems is widely called *early mode padding*. In contrast, we assume that late mode delay optimization is fully exploited, and all late mode delays are minimum. Therefore, late mode slacks should be prioritized over early mode slacks.

We call the set of late mode test edges E_{late} and the set of early mode test edges by E_{early}, for both, the signal graph model and the register graph model. E_{late} is composed of those edges that represent setup tests, flush propagation edges, or late mode predefined required arrival time tests, while E_{early} consists of edges that represent hold tests, and early mode predefined required arrival times.

In principle, user defined test slack can be improved by adding delay on the path to the early mode test pin. However, signals that must leave the chip simultaneously must do this within each process corner. Therefore, such paths from the registers to the primary output pins are constructed with a large amount of regularity, which does not allow extensive delay insertion. Usually scheduling of registers feeding user defined tests, is even forbidden by the designers. If not, they should be added to E_{late} or E_{early}, depending on the ability to add delay to the early signal paths.

In the end, the computed schedule has to be realized by a clocktree. When each clock signal must arrive at a singular arrival time, the clocktree construction is quite difficult. Actually, clocktrees meeting exact arrival times cannot be constructed in practice. Instead, slight violations of the prescribed arrival times must be tolerated. It would be beneficial if information about uncritical registers, where such violations could be absorbed by large positive slacks, would be available.

The quality and power consumption of a clocktree improves if the clock signals are allowed to arrive within time windows instead of single arrival times. Clock skew scheduling can also be used to reallocate positive slacks at non-critical sinks for time windows for the clock signals. These considerations result in a three step clock skew scheduling scheme.

1. **Late Mode Optimization** of slacks below some target $\mathrm{S}_{\mathrm{l}}^{\mathrm{tgt}} \in \mathbb{R}$ that cannot be improved by early mode padding.
2. **Early Mode Optimization** of slacks below some target $\mathrm{S}_{\mathrm{e}}^{\mathrm{tgt}} \in \mathbb{R}$ that can be improved by early mode padding, while not worsening the slack distribution of late slacks with respect to $\mathrm{S}_{\mathrm{l}}^{\mathrm{tgt}}$.
3. **Time Window Optimization** for clock signal sinks, while not worsening the slack distribution of late slacks with respect to $\mathrm{S}_{\mathrm{l}}^{\mathrm{tgt}}$ and early slacks with respect to $\mathrm{S}_{\mathrm{e}}^{\mathrm{tgt}}$.

For late mode optimization we set $\mathrm{E}_{\mathrm{t}} := E_{\text{late}}$, $\mathcal{F} := \Big\{\{e\} \mid e \in \mathrm{E}_{\mathrm{t}}\Big\}$, and $w \equiv 1$. According to Remark 5.4, the costs on all (late) test edges are reduced by $\mathrm{S}_{\mathrm{l}}^{\mathrm{tgt}}$ and then, Algorithm 7 is run until $\lambda^\star \geq 0$.

The early mode optimization continues on the graph G, as it was contracted by Algorithm 7 during late mode optimization. To preserve the leximin maximality of the late slack distribution above the slack target, the reduced costs on these edges must be non-negative. Higher slacks than zero on any non-contracted edge from E_{late} are irrelevant. For early mode optimization the test edge set E_t is redefined as $E_t := E_{\text{early}}$, and w is restricted to the new test set E_t. Again the edge-costs of (early) test edges are reduced by $\mathrm{S}_{\mathrm{e}}^{\mathrm{tgt}}$, and then Algorithm 7 is continued until $\lambda^\star \geq 0$.

Finally, during time window optimization, the leximin maximality of the late mode and the early mode slack distribution with respect to their slack targets have to be preserved, that means the reduced costs must be non-negative, $c_\pi(e) \geq 0$ for all $e \in E_{\text{late}} \cup E_{\text{early}}$. Thus to compute time windows, the set E_t is redefined as $E_t := E_\eta$, and its edge-costs is set uniformly to some large negative constant $c(e) = -K$, which corresponds to a maximum desired time window size. In the presence of maximally allowed time window constraints, K could be chosen as the maximum length of all time window constraints. Finally, the node potentials on the incident nodes of each early-late-separation edge $(v_l, v_e) \in E_\eta$ define an admissible window $[\pi(v_e), \pi(v_l)]$.

5.2.3 Multi-Domain Cycle Time Minimization

In the presence of multiple clock frequencies or multi-cycle paths, Lemma 5.2 does not apply any more, because there is no one-to-one correspondence between the worst slack improvement and the cycle time reduction. In this Section we give a detailed model for realistic cycle time minimization. It maximizes a weighted worst slack that corresponds one-to-one to a reduction of the reference cycle time of the design. Furthermore, we prove that a feasible reference cycle time will exist if IO-constraints and user defined tests follow certain rules.

We assume that all interacting clock domains are derived from a common reference clock with cycle time T^{ref}. More precisely, for a clock definition $0 \leq t^{lead} < t^{trail} \leq T$ there are numbers $\dot{t}^{lead}, \dot{t}^{trail}, \dot{T} \in \mathbb{Q}_{\geq 0}$ such that

$$t^{lead} = \dot{t}^{lead} \cdot T^{ref}, \tag{5.14}$$
$$t^{trail} = \dot{t}^{trail} \cdot T^{ref}, \tag{5.15}$$

and

$$T = \dot{T} \cdot T^{ref}. \tag{5.16}$$

Signals from asynchronous sources obviously cannot be tested against each other (see also Section 2.4.6). Furthermore we exclude the hypothetical case that two pairwise asynchronous signals are propagated into a common clock subnetwork, and are controlling a common set of registers. Thus we can schedule all clock domains that are derived from a common reference clock independently from other reference clocks. Figure 5.2.3 shows an example of a reference clock signal generated by an oscillator and propagated to a phase locked loop (PLL). The PLL generates oscillating output clock signals, whose cycle times are rational multiples of the input cycle time. The PLL output signals are then propagated to the registers. On their way several clock signals might be merged. For instance, multiplexers in the clock network allow for running the subsequent registers with varying frequencies.

During the description of the routine, we consider only paths between registers. These paths usually represent the frequency-limiting constraints of a chip-core. In the end of this subsection we show under what conditions and how predefined required arrival times and user defined tests can be incorporated as well. Recall, the setup test equation (2.12):

$$\mathrm{at}(d, \sigma_d) + \mathrm{setup}\Big(\mathrm{slew}(d, \sigma_d), \mathrm{slew}(c, \sigma_c)\Big) \leq \mathrm{at}(c, \sigma_c) + \mathrm{adj}(\sigma_d, \sigma_c).$$

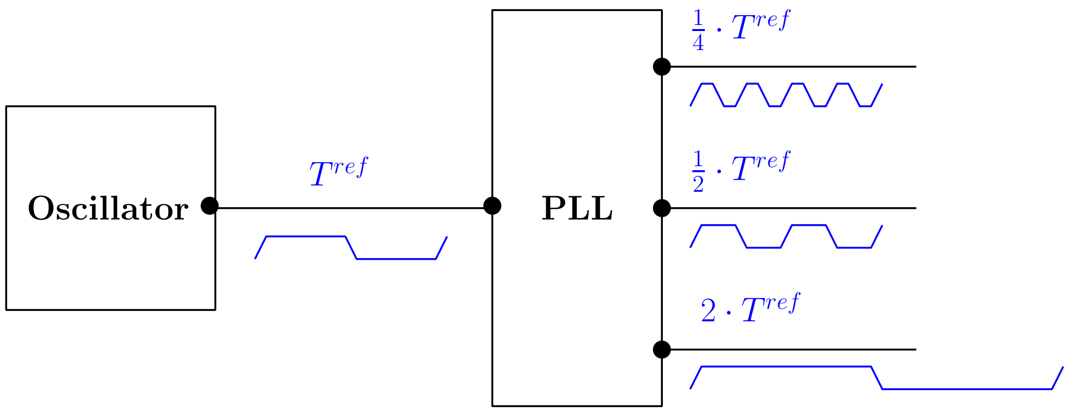


Figure 5.4: Example of clock signals generated by a PLL.

We need to investigate how this equation transforms under a variable reference cycle time T^{ref}. Let $\dot{\operatorname{adj}}(\sigma_d, \sigma_c)$ denote the partial derivative of $\operatorname{adj}(\sigma_d, \sigma_c)$ in T^{ref}. Then

$$\begin{aligned}
\dot{\operatorname{adj}}(\sigma_d, \sigma_c) &:= \tfrac{\partial \operatorname{adj}(\sigma_d,\sigma_c)}{\partial T^{ref}} \\
&= \dot{t}_d - \dot{t}_c + \tfrac{\partial \operatorname{mpd}(\sigma_d,\sigma_c)}{\partial T^{ref}} + (\operatorname{mc}_{\sigma_d,\sigma_c} - 1) \\
&= \dot{t}_d - \dot{t}_c \\
&\quad + \min\Big\{\dot{t}_c + i\dot{T}_c - (\dot{t}_d + j\dot{T}_d) \,\Big|\, i,j \in \mathbb{N}_0; \dot{t}_c + i\dot{T}_c - (\dot{t}_d + j\dot{T}_d) > 0\Big\} \\
&\quad + (\operatorname{mc}_{\sigma_d,\sigma_c} - 1).
\end{aligned} \tag{5.17}$$

With $\dot{t}_d, \dot{t}_c, \dot{T}_d, \dot{T}_c \in \mathbb{Q}_{\geq 0}$ we have $\dot{\operatorname{adj}}(\sigma_d, \sigma_c) \in \mathbb{Q}$. The adjust values of late mode test must be adapted proportionally to $\dot{\operatorname{adj}}(\sigma_d, \sigma_c)$ when changing the reference cycle time T^{ref}, and (2.12) transforms to

$$\begin{aligned}
\operatorname{at}(d, \sigma_d) + \operatorname{setup}\big(\operatorname{slew}(d, \sigma_d), \operatorname{slew}(c, \sigma_c)\big) &\leq \\
\operatorname{at}(c, \sigma_c) + \dot{\operatorname{adj}}(\sigma_d, \sigma_c) \cdot T^{ref}&.
\end{aligned} \tag{5.18}$$

Because of (5.14) and (5.15), also the pulse-width tests (5.11) and (5.12) will change with varying T^{ref}. Thus (5.11) and (5.12) must be rewritten as

$$\operatorname{at}(p, \sigma^{lead}) + (\dot{t}^{trail} - \dot{t}^{lead})T^{ref} \geq \operatorname{at}(p, \sigma^{trail}) \tag{5.19}$$

$$\operatorname{at}(p, \sigma^{trail}) + (\dot{t}^{lead} - \dot{t}^{trail})T^{ref} \geq \operatorname{at}(p, \sigma^{lead}) \tag{5.20}$$

where σ_{lead} and σ_{trail} refer to either early mode or late mode clock signals. If signals from different clock sources arrive in p or signals at several clock sinks need to be coupled, analogous constraints between leading (or trailing) edge pins need to be added.

Summarizing, for each edge $e \in E(G)$ in the slack balance graph G that is a setup test edge, flush propagation edge (in the signal graph model), or a pulse width test edge, there is a constant $w(e) \in \mathbb{Q}$ such that the edge costs are variable costs

$$c(e) + w(e) \cdot T^{ref} \tag{5.21}$$

in the reference cycle time T^{ref}. The higher T^{ref}, the higher are the variable costs.

In the register graph model flush propagation edges are not explicitly visible. The variable adjusts of flushing edges on a path must be accumulated and added to the variable costs of the corresponding register graph edges. Doing this, it might happen that different paths with different accumulated weights need to be merged during the creation of the register graph. This introduces piecewise linear costs that can be modeled by parallel edges with linear costs functions on each edge.

Note that (5.19) and (5.20) induce cycles C with zero total weight $w(E(C)) = 0$ and zero total variable costs $c(E(C)) + w(E(C)) \cdot T^{ref} = 0$ for any choice of T^{ref}. Here the weights w for pulse-width edges of type (5.20) are negative. However, we can state that the total weight of any cycle in the slack balance graph is non-negative.

Lemma 5.13. *Let $C \subseteq G$ be a cycle in the slack balance instance (G, c, w). Then C has non-negative total edge weight: $w(E(C)) \geq 0$. If C contains a setup test edge or flush propagation edge, the total weight is strictly positive: $w(E(C)) > 0$.*

Proof. Let C be a cycle in G. If $w(e) = 0$ for all $e \in E(C)$, the statement is clear. Let there be an edge $e \in E(G)$ with $w(e) \neq 0$. Then C can be divided into paths $P_0, P_1, \ldots, P_{k-1}$, where the end vertices of the paths represent exactly the clock signals in $V(C)$, and the end vertex of P_i equals the start vertex of $P_{(i+1) \bmod k}$, with $0 \leq i \leq k-1$. Each path contains either at least a setup test edge, a flush propagation edge, or a pulse-width edge.

Furthermore, let $\dot{t}_i$ be the slope of the clock edge reference time at the endpoint of P_i. Then the weight $w(P_i) = \sum_{e \in E(P_i)} w(e)$ of path P_i is the sum of the difference $\dot{t}_i - \dot{t}_{((k+i-1) \bmod k)}$ and some value $\delta_i \geq 0$. The latter is zero if the path consists of a pulse-width edge, or $\delta_i = \dot{mpd}(\sigma_i, \sigma'_i) + (\mathrm{mc}_{\sigma_i,\sigma'_i} - 1) > 0$ if P_i contains a setup test or flush propagation edge. Here σ_i and σ'_i shall be the underlying signals, and $\dot{mpd}(\sigma_i, \sigma'_i)$ is given by

$$\dot{mpd}(\sigma_i, \sigma'_i) = \min\Big\{\dot{t}_c + i\dot{T}_c - (\dot{t}_d + j\dot{T}_d) \,\Big|\, i, j \in \mathbb{N}_0; \dot{t}_c + i\dot{T}_c - (\dot{t}_d + j\dot{T}_d) > 0\Big\} > 0.$$

Now the differences $\dot{t}_i - \dot{t}_{((k+i-1) \bmod k)}$ cancel out in the total weight $w(C)$:

$$\begin{aligned} w(C) = & \sum_{e \in E(C)} w(e) & = \sum_{i=0}^{k-1} \sum_{e \in E(P_i)} w(e) & \\ = & \sum_{i=0}^{k-1} \Big(\dot{t}_i - \dot{t}_{((k+i-1) \bmod k)} + \delta_i\Big) & = \sum_{i=0}^{k-1} \delta_i & \geq 0, \end{aligned} \tag{5.22}$$

and $w(C) > 0$ if and only if $E(C)$ contains a setup test edge or flush propagation edge.

□

Now, according to Theorem 5.5 and Remark 5.8 we obtain the following result on the computation of the minimum reference cycle time.

Theorem 5.14. *Let G be a slack balance graph, $c : E(G) \to \mathbb{R}$ and $w : E(G) \to \mathbb{Q}$ be defined as for (5.21), $\mathrm{E_t} := \{e \in E(G) \mid w(e) \neq 0\}$, and $\mathcal{F} = \{\mathrm{E_t}\}$. Then the minimum feasible reference cycle time T^{ref} equals the negative minimum ratio cycle*

$$-\min\left\{\frac{c(e)}{w(e)} \;\middle|\; e \in E(C), C \in G \textit{ cycle }, w(E(C)) > 0\right\}. \tag{5.23}$$

It can be computed in strongly polynomial time

$$O(n^3 \log n + \min\{nm, n^3\} \cdot \log^2 n \log\log n + nm \log m),$$

or in pseudo-polynomial time

$$O(w_{\max}(nm + n^2 \log n)),$$

for integral weights $w : E(G) \to \mathbb{Z}$ with $w_{\max} := \max\{w(e) | e \in \mathrm{E_t}\}$.

Proof. The proof is the same as for Theorem 5.1, with $\sum_{(v,w)\in E(C)}(T - \vartheta(v,w))$ being replaced by $\sum_{(v,w)\in E(C)}(T^{ref} w(v,w) + c(v,w))$. The running times follow from Theorem 5.5.

□

The set E_{late} contains also edges that represent user defined tests, or late mode predefined required arrival times. Often, the involved edges in the slack balance graph do also have edge costs proportional to the reference cycle time T^{ref}. As long as such an edge does induce

- neither a cycle of negative total weight,
- nor a cycle of zero weight and negative costs,

such constraints do not limit the computation of the minimum T^{ref}.

A cycle C of negative weight, $w(E(C)) < 0$, would be relaxed, when increasing T^{ref}, which is very unlikely for practical situations. In the second case, $w(E(C')) = 0$ and $c(E(C')) < 0$, the timing constraints could not be met by any reference cycle time T^{ref}.

5.3 Iterative Local Balancing

The idea behind the *iterative local clock balancing* is to iteratively schedule each clock sink arrival time to its local optimum. To illustrate a *locally optimum schedule*, let $r \in \mathcal{C}$ be a (non-transparent) register with a single clock input $c \in P(r) \cap P_{cs}$, a single data input $d \in P(r)$, and a single data output $y \in P(r)$, and let $\mathrm{slk}_o := \min\{\mathrm{slk}(y, \sigma) \mid \sigma \in \mathfrak{S}(y)\}$ be the worst late slack at y and let $\mathrm{slk}_i := \min\{\mathrm{slk}(d, \sigma) \mid \sigma \in \mathfrak{S}(d)\}$ be the worst late slack at d. When shifting all clock arrival times in c by

$$\Delta\,\mathrm{at}(c) = \frac{\mathrm{slk}_o - \mathrm{slk}_i}{2},$$

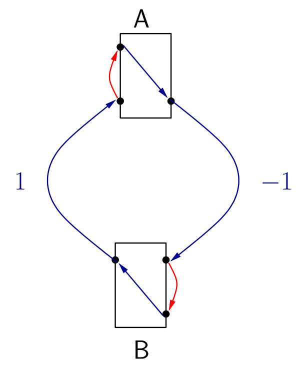


Figure 5.5: Example for iterative clock skew scheduling.

the resulting worst late slacks at y and d will be equal, and c will be scheduled to a locally optimum solution, assuming fixed delays (no slew effects).

The algorithm can be carried out in a Jacobi-style, where first $\Delta\,\mathrm{at}(c)$ is computed for each clock sink $c \in P_{cs}$, and then the arrival times at all sinks are updated simultaneously by

$$\mathrm{at}(c,\sigma_c) = \mathrm{at}(c,\sigma_c) + \begin{cases} \xi \cdot \Delta\,\mathrm{at}(c) & \text{if } \Delta\,\mathrm{at}(c) \geq 0 \\ (1-\xi)\cdot \Delta\,\mathrm{at}(c) & \text{if } \Delta\,\mathrm{at}(c) < 0 \end{cases}, \forall\, \sigma_c \in \mathfrak{S}(c), \qquad (5.24)$$

given some *movement rate* $0 \leq \xi \leq 1$.

The restriction of the local balancing through ξ is necessary to guarantee convergence and maximization of the worst (late) slack. The example in Figure 5.5 shows two registers A and B, a data path $\mathsf{A} \rightarrow \mathsf{B}$ with slack -1, and a data path $\mathsf{B} \rightarrow \mathsf{A}$ with slack 1. The local optimum schedule changes are $\Delta\,\mathrm{at}(\mathsf{A}) = -1$, and $\Delta\,\mathrm{at}(\mathsf{B}) = 1$. At a Jacobi-like simultaneous shifting, the full shifts would lead to an $\mathsf{A} \rightarrow \mathsf{B}$-slack of 1, and a $\mathsf{B} \rightarrow \mathsf{A}$-slack of -1. Evidently, this procedure would end up in an equivalently unbalanced situation. In contrast, when applying limited shifts according to (5.24), both path slacks would be 0 and the cycle would be optimally balanced after a single iteration.

This method has been used for a while as a heuristic in industrial tools, such as IBM EinsTimer$^{\mathrm{TM}}$. The question on the quality of the result was theoretically unsolved, and will be answered in the next subsections.

There is also link to the task of *matrix balancing*, for example as a pre-conditioning technique, which we shortly point out. Schneider and Schneider [1991] defined the combinatorial minimum balance problem to balance matrices with respect to the l_∞-norm, which is also known in algebraic optimization under the name *algebraic flow*. Matrix balancing with respect to any other norm l_p-norm, $1 \leq p < \infty$, can be solved optimally with a local search method very similar to that described above (see Osborne [1960]).

5.3.1 Quality of Iterative Local Balancing

The procedure can be formalized as an algorithm working on the register graph $G^{\mathcal{R}}$ (see Algorithm 8). Due to Remark 5.11, the set of clocktree nodes can be represented by cluster pairs $(v_e^1, v_l^1), (v_e^2, v_l^2), \dots, (v_e^k, v_l^k)$ and an additional node v_0 representing the time-origin, where $k \in \mathbb{N}$ is the number of clusters. During late and early optimization the reduced costs of the early-late separation edges must simply be non-negative: $c_\pi(v_l^i, v_e^i) = \pi(v_l^i) - \pi(v_e^i) \geq 0$. Only during the time window optimization $c_\pi(v_l^i, v_e^i)$ is of interest. Thus, these node-pairs can be considered as single nodes during late and early mode optimization. The maximally allowed time window edges are removed from $G^{\mathcal{R}}$. Instead, these constraints are handled explicitly in line 11 and line 12 of Algorithm 8.

Now an instance for the iterative scheduling consists of a weighted directed graph (G, c), where the set $E(G) = E_{\text{late}} \cup E_{\text{early}}$ consists of late mode constraints E_{late} and early mode constraints E_{early}. Furthermore, legitimated by Remark 5.4, we assume $\mathrm{S}_\mathrm{l}^{\mathrm{tgt}} = \mathrm{S}_\mathrm{e}^{\mathrm{tgt}} = 0$ throughout this section. The iterative algorithm for late- and early-mode optimization is given in Algorithm 8. Depending on the optimization mode $\eta \in \{\text{early}, \text{late}\}$, we denote

$$\mathrm{E}_\mathrm{t}(\eta) := \begin{cases} E_{\text{late}} & \text{if } \eta = \text{late}, \\ E_{\text{early}} & \text{if } \eta = \text{early}\,. \end{cases}$$

During early mode optimization ($\eta = $ early) late slacks must be preserved. Lines 13–18, first compute the maximum allowed reduction $-c_\pi^{+,\text{late}}(v)$ and the maximum allowed augmentation $c_\pi^{-,\text{late}}(v)$ of $\pi(v)$, and then restrict $\Delta\pi(v)$ to $[-c_\pi^{+,\text{late}}(v), c_\pi^{-,\text{late}}(v)]$. Although Algorithm 8 is a greedy algorithm, which does not solve the Slack Balance Problem, we can show that, during late mode optimization, the worst slack converges towards the best achievable worst slack if the movement rates ξ are chosen adequately.

To analyze the algorithm we make following notations. Let π^j, $j \in \mathbb{N}$, denote the node potential after the j-th iteration of the outer loop (lines 3–26), and π^0 the initial node potential, before the loop is entered. Furthermore, let

$$\lambda^j := \min\{c_{\pi^j}(e) \mid e \in E_{late}\},$$

$j \in \mathbb{N}_{\geq 0}$, be the worst reduced cost after the j-th iteration, and be $\Delta\pi^j(v), \xi^j$ the values of $\Delta\pi(v), \xi$ in the j-th iteration, that is, $\pi^j(v) = \pi^{j-1}(v) + \xi^j \Delta\pi^j(v)$.

The convergence of the worst slack towards the maximum possible value, and therefore the finiteness of Algorithm 8, can be guaranteed for a sufficiently balanced choice of the movement rate ξ.

Definition 5.15. *A sequence $(\xi^j)_{j\in\mathbb{N}}$, with values $\xi^j \in [0,1]$, is called* **sufficiently balanced** *with respect to some constant $N \in \mathbb{N}$ if there is a constant $K \in \mathbb{N}$ such that any set $\Xi = \{\xi^i, \xi^{i+1}, \dots, \xi^{i+K}\}$ of K consecutive sequence elements can be separated into two sets $\Xi = \Xi_1 \dot\cup \Xi_2$, with $|\Xi_1| \geq N-1$, $|\Xi_2| \geq N-1$, $\Xi_1 \subset \{\xi \in \Xi \mid \xi \geq \frac{1}{2}\}$, and $\Xi_2 \subset \{\xi \in \Xi \mid \xi \leq \frac{1}{2}\}$. Such a K is of* **sufficient length** *for N.*

Algorithm 8 Iterative Local Balancing Algorithm

```
Input: $(G, c), \eta \in \{\text{early}, \text{late}\}, \epsilon$.
 1: $\pi(v_0) = 0$;
 2: $\pi(v) = \frac{1}{2}\Big(\overline{\pi}(v) + \underline{\pi}(v)\Big)$;
 3: repeat
        /* Compute locally optimum shifts $\Delta\pi(v)$. */
 4:     for $v \in V(G) \setminus \{v_0\}$ do
 5:         $c_\pi^+(v) \leftarrow \min\{c_\pi(v,w) \mid (v,w) \in \delta^+(v) \cap \mathrm{E_t}(\eta)\}$;
 6:         $c_\pi^-(v) \leftarrow \min\{c_\pi(v,w) \mid (v,w) \in \delta^-(v) \cap \mathrm{E_t}(\eta)\}$;
 7:         $\Delta\pi(v) \leftarrow \frac{1}{2}(c_\pi^-(v) - c_\pi^+(v))$;
 8:     end for

        /* Chose $\xi$ (for instance with respect to objective function).*/
 9:     Chose $\xi \in [0,1]$;

10:     for $v \in V(G) \setminus \{v_0\}$ do
        /* Bound $\Delta\pi(v)$ to feasible region */
11:         $\Delta\pi(v) \leftarrow \min\{\Delta\pi(v), \overline{\pi}(v) - \pi(v)\}$;
12:         $\Delta\pi(v) \leftarrow \max\{\Delta\pi(v), \underline{\pi}(v) - \pi(v)\}$;

        /* Do not worsen late slacks below 0. */
13:         if $\eta = \text{early}$ then
14:             $c_\pi^{+,\text{late}}(v) \leftarrow \max\{0, \min\{c_\pi(v,w) \mid (v,w) \in \delta^+(v) \cap E_{\text{late}}\}\}$;
15:             $c_\pi^{-,\text{late}}(v) \leftarrow \max\{0, \min\{c_\pi(u,v) \mid (u,v) \in \delta^-(v) \cap E_{\text{late}}\}\}$;
16:             $\Delta\pi(v) \leftarrow \min\{\Delta\pi(v), c_\pi^{-,\text{late}}(v)\}$;
17:             $\Delta\pi(v) \leftarrow \max\{\Delta\pi(v), -c_\pi^{+,\text{late}}(v)\}$;
18:         end if

        /* Apply shifts. */
19:         if $\Delta\pi(v) \geq 0$ then
20:             $\pi(v) \leftarrow \pi(v) + \xi\Delta\pi(v)$;
21:         else
22:             $\pi(v) \leftarrow \pi(v) + (1-\xi)\Delta\pi(v)$;
23:         end if
24:     end for
25:     $\lambda = \min\{c_\pi \mid e \in E(\eta)\}$;
26: until $\lambda \geq 0$ or $\lambda$ is $\epsilon$-optimal
```

Theorem 5.16. *Let $G = (V, E)$ be a graph with edge costs c and lower and upper bounds for the node potentials $\overline{\pi}, \underline{\pi} : V(G) \to \mathbb{R}$, and let*

$$\lambda^\star = \max_{\substack{\pi: V(G)\to\mathbb{R} \\ \underline{\pi}\le\pi\le\overline{\pi}}} \min\{c_\pi(e) \mid e \in E(G)\}$$

be the maximum achievable minimum reduced cost. Furthermore, let Algorithm 8 be run in late mode ($\eta = \text{late}$) with a sequence $(\xi^j)_{j\in\mathbb{N}}$ of movement rates that is sufficiently balanced with respect to $|V| - 1$. Then the worst reduced costs converge towards the best possible worst reduced costs:

$$\lim_{j\to\infty} \lambda^j = \lambda^\star. \tag{5.25}$$

The special case $\underline{\pi} \equiv -\infty$ and $\overline{\pi} \equiv \infty$ was recently proved by Kleff [2008]. The proof of Theorem 5.16 is rather technical. We will split it into a series of lemmas and corollaries. Throughout this subsection we consider only late mode edges: $E = E_{\text{late}}, E_{\text{early}} = \emptyset$.

First, we show how the reduced costs of an edge and its "neighboring" edges are related after a single balancing iteration.

Lemma 5.17. *Let $c_{\pi^{j+1}}(v, w)$ be the reduced costs of an edge $(v, w) \in E$ after the $(j + 1)$-th (late mode) iteration of the main loop in Algorithm 8. Then, with $\gamma := \min\{c^-_{\pi^j}(v), c^+_{\pi^j}(w)\}$, either the inequality*

$$c_{\pi^{j+1}}(v, w) \ge \frac{1}{2}\Big(\gamma + c_{\pi^j}(v, w)\Big) \tag{5.26}$$

holds, or $\gamma > c_{\pi^j}(v, w)$ and one of $\Delta\pi^{j+1}(v), \Delta\pi^{j+1}(w)$ attains its limit, that is, $\Delta\pi^{j+1}(v) = \overline{\pi}(v) - \pi^j(v)$ or $\Delta\pi^{j+1}(w) = \underline{\pi}(w) - \pi^j(w)$.

Proof. Let us first assume $\overline{\pi} \equiv \infty$ and $\underline{\pi} \equiv -\infty$, that is, line 11 and line 12 are ignored. We denote $b := \frac{1}{2}(\gamma - c_{\pi^j}(v, w))$. Then the following inequalities hold:

$$\Delta\pi^j(v) = \frac{1}{2}(c^-_{\pi^j}(v) - c^+_{\pi^j}(v)) \ge \frac{1}{2}(\gamma - c^+_{\pi^j}(v)) \ge \frac{1}{2}(\gamma - c_{\pi^j}(v, w)) = b. \tag{5.27}$$

The first inequality uses the definition of γ, and the second the definition of $c^+_{\pi^j}(v)$. Analogously, it can be shown that $\Delta\pi^j(w) \le -b$.

First, if $\gamma \le c_{\pi^j}(v, w)$, we have to bound the potential decrease from $c_{\pi^j}(v, w)$ to $c_{\pi^{j+1}}(v, w)$ from below. The highest (absolute) decrease occurs if $\Delta\pi^j(v) \le 0$ and $\Delta\pi^j(w) \ge 0$. Thus $c_{\pi^{j+1}}(v, w)$ is bounded from below by

$$\begin{aligned} c_{\pi^{j+1}}(v, w) &\ge c_{\pi^j}(v, w) + (1 - \xi^j)\Delta\pi^j(v) - \xi^j\Delta\pi^j(w) \\ &\ge c_{\pi^j}(v, w) + (1 - \xi^j)b - \xi^j(-b) \\ &\ge c_{\pi^j}(v, w) + \tfrac{1}{2}(\gamma - c_{\pi^j}(v, w)) = \tfrac{1}{2}(\gamma + c_{\pi^j}(v, w)). \end{aligned} \tag{5.28}$$

If we add the bounds for π to this case, which means $\overline{\pi} \not\equiv \infty$ and $\underline{\pi} \not\equiv -\infty$, the decrease from $c_{\pi^j}(v, w)$ to $c_{\pi^{j+1}}(v, w)$ can only be lessened, and the inequality will still hold.

Second, if $\gamma > c_{\pi^j}(v,w)$, we have $\Delta\pi^j(v) \geq 0$ and $\Delta\pi^j(w) \leq 0$, and thus

$$\begin{aligned} c_{\pi^{j+1}}(v,w) &\geq c_{\pi^j}(v,w) + \xi^j \Delta\pi^j(v) - (1-\xi^j)\Delta\pi^j(w) \\ &\geq c_{\pi^j}(v,w) + \xi^j b - (1-\xi^j)(-b) \\ &\geq c_{\pi^j}(v,w) + \tfrac{1}{2}(\gamma - c_{\pi^j}(v,w)) = \tfrac{1}{2}(\gamma + c_{\pi^j}(v,w)). \end{aligned} \tag{5.29}$$

Now the increase from $c_{\pi^j}(v,w)$ to $c_{\pi^{j+1}}(v,w)$ can be restricted if we add bounds $\overline{\pi} \not\equiv \infty$ and $\underline{\pi} \not\equiv -\infty$. However, (5.29) will remain valid, if none of the involved inequalities becomes tight, which means $\Delta\pi^{j+1}(v) < \overline{\pi}(v) - \pi^j(v)$ and $\Delta\pi^{j+1}(w) > \underline{\pi}(w) - \pi^j(w)$.

□

As a consequence of this Lemma we obtain the (preliminarily weak) monotony and convergence of the sequence $(\lambda^j)_{j\in\mathbb{N}}$ of worst reduced costs.

Corollary 5.18. *The sequence $(\lambda^j)_{j\in\mathbb{N}}$ is non-decreasing, bounded from above by $\lambda^\star$, and convergent.*

Proof. If the reduced cost of an edge $(v,w) \in E_{\text{late}}$ decreases from an iteration j to an iteration $j+1$, that is, $c_{\pi^{j+1}}(v,w) < c_{\pi^j}(v,w)$, there must have been a preceding or succeeding edge with less reduced cost, that is, $c_{\pi^j}(v,w) > \gamma := \min\{c^-_{\pi^j}(v), c^+_{\pi^j}(w)\}$, and by Lemma 5.17

$$c_{\pi^{j+1}}(v,w) \geq \frac{1}{2}\Big(\gamma + c_{\pi^j}(v,w)\Big) \geq \lambda^j.$$

Thus we have $\lambda^{j+1} \geq \lambda^j$.

□

It remains to show that $\lim_{j\to\infty} \lambda^j = \lambda^\star$, which means that λ^j does not grow too slowly. First, we show how the reduced costs $c_{\pi^{j+1}}(e)$ of an edge $e \in E$ after an iteration $j+1$, which are now close to the previously worst reduced cost λ^j, depend on the reduced costs after the previous iteration j.

Corollary 5.19. *Let $(v,w) \in E$ be an edge with $c_{\pi^{j+1}}(v,w) \leq \lambda^j + \epsilon$, for some $\epsilon > 0$. Then the following inequalities hold for the reduced costs after the previous iteration*

$$c_{\pi^j}(v,w) \leq \lambda^j + 2\epsilon, \tag{5.30}$$

and either

$$\gamma := \min\{c^-_{\pi^j}(v), c^+_{\pi^j}(w)\} \leq \lambda^j + 2\epsilon. \tag{5.31}$$

or one of $\Delta\pi^{j+1}(v), \Delta\pi^{j+1}(w)$ attains its limit, that is, $\Delta\pi^{j+1}(v) = \overline{\pi}(v) - \pi^j(v)$ or $\Delta\pi^{j+1}(w) = \underline{\pi}(w) - \pi^j(w)$.

Proof. First, to show (5.30), assume $c_{\pi^j}(v,w) > \lambda^j + 2\epsilon > c_{\pi^{j+1}}(v,w)$. Thus $\gamma < c_{\pi^j}(v,w)$ and by Lemma 5.17 we get $c_{\pi^{j+1}}(v,w) \geq \frac{1}{2}(\gamma + c_{\pi^j}(v,w)) > \lambda^j + \epsilon$, a contradiction. Second, if $\gamma > \lambda^j + 2\epsilon$, then $\gamma > c_{\pi^j}(v,w)$, and according to Lemma 5.17 either $c_{\pi^{j+1}}(v,w) > \lambda^j + \epsilon$, a contradiction, or $\Delta\pi^{j+1}(v) = \overline{\pi}(v) - \pi^j(v)$ or $\Delta\pi^{j+1}(w) = \underline{\pi}(w) - \pi^j(w)$.

□

Slightly weaker inequalities as (5.31) can be restricted to one of $c^-_{\pi^j}(v)$ and $c^+_{\pi^j}(w)$, when taking the movement rate ξ^j into account:

Lemma 5.20. *Let $(v,w) \in E$ be an edge with $c_{\pi^{j+1}}(v,w) \leq \lambda^j + \epsilon$, for some $\epsilon > 0$. Then,*

- *if $\xi^j \geq \frac{1}{2}$, $c^-_{\pi^j}(v) \leq \lambda^j + 4\epsilon$ or $\overline{\pi}(v) - \pi^j(v) < 2\epsilon$, and*
- *if $\xi^j \leq \frac{1}{2}$, $c^+_{\pi^j}(w) \leq \lambda^j + 4\epsilon$ or $\pi^j(w) - \underline{\pi}(w) < 2\epsilon$.*

Proof. We set $e = (v,w)$ and $b := \frac{1}{2}(\lambda^j - c_{\pi^j}(e)) \leq 0$. The following inequalities hold independently from ξ^j: $\Delta\pi^{j+1}(v) \geq \frac{1}{2}(\lambda^j - c_{\pi^j}(e)) = b$ and $\Delta\pi^{j+1}(w) \leq \frac{1}{2}(c_{\pi^j}(e) - \lambda^j) = -b$.

First, we consider the case $\xi^j \geq \frac{1}{2}$. Assume $c^-_{\pi^j}(v) > \lambda^j + 4\epsilon$. By construction, either $\Delta\pi^{j+1}(v) > \frac{1}{2}(\lambda^j + 4\epsilon - c_{\pi^j}(e)) = b + 2\epsilon$, or, if this inequality is invalid, $\Delta\pi^{j+1}(v) = \overline{\pi}(v) - \pi^j(v)$. In the second case, we have

$$0 \leq \overline{\pi}(v) - \pi^j(v) = \Delta\pi^{j+1}(v) \leq \frac{1}{2}(\lambda^j + 4\epsilon - c_{\pi^j}(e)) \leq 2\epsilon.$$

If $\Delta\pi^{j+1}(v)$ is not constrained by $\overline{\pi}(v)$ we show that $c_{\pi^{j+1}}(v,w) > \lambda^j + \epsilon$, which would be a contradiction. Because of $b \leq 0$, we have $\pi^{j+1}(w) - \pi^j(w) \leq \xi^j(-b)$. As the sign of $(b + 2\epsilon)$ is unknown, we apply a case differentiation to estimate $c_{\pi^{j+1}}(v,w)$. First, if $b + 2\epsilon \geq 0$, then

$$\begin{aligned} c_{\pi^{j+1}}(v,w) &\geq c_{\pi^j}(v,w) + \xi^j\Delta\pi^{j+1}(v) - \xi^j(-b) \\ &> c_{\pi^j}(v,w) + \xi^j(b + 2\epsilon) + \xi^j(b) = c_{\pi^j}(v,w) + 2\xi^j b + 2\xi^j\epsilon \\ &= \xi^j\lambda^j + (1 - \xi^j)c_{\pi^j}(v,w) + 2\xi^j\epsilon \geq \lambda^j + \epsilon. \end{aligned}$$

Second, if $b + 2\epsilon < 0$, then

$$\begin{aligned} c_{\pi^{j+1}}(v,w) &\geq c_{\pi^j}(v,w) + (1 - \xi^j)\Delta\pi^{j+1}(v) - \xi^j(-b) \\ &> c_{\pi^j}(v,w) + (1 - \xi^j)(b + 2\epsilon) + \xi^j b = c_{\pi^j}(v,w) + b + 2(1 - \xi^j)\epsilon \\ &\geq c_{\pi^j}(v,w) + b = \lambda^j - b > \lambda^j + 2\epsilon. \end{aligned}$$

The second case $\xi^j \leq \frac{1}{2}$ is proven analogously.

□

Lemma 5.21. *If $(\xi^j)_{j\in\mathbb{N}}$ is sufficiently balanced with respect to $|V|$, then for all $\epsilon > 0$ there is an iteration j and*

- *a cycle $C \subseteq G$ such that $c_{\pi^j}(c) \leq \lambda^j + \epsilon$ for all $e \in E(C)$, or*
- *a path $P \subseteq G$ from a node $s \in V(G)$ to a node $t \in V(G)$ such that $c_{\pi^j}(c) \leq \lambda^j + \epsilon$ for all $e \in E(P)$, and $\pi(s) \geq \overline{\pi}(s) - \epsilon$ and $\pi(t) \leq \underline{\pi}(t) + \epsilon$.*

Proof. The idea behind this proof is to construct an edge progression, which either contains a cycle or a path with the required properties. Let K be of sufficient length for the given sequence $(\xi^j)_{j\in\mathbb{N}}$ and the vertex set cardinality $|V|$, according to Definition 5.15. Based on the given $\epsilon > 0$, we choose a smaller constant $0 < \epsilon' < 4^{1-K}\epsilon$. As $(\lambda^j)_{j\in\mathbb{N}}$ is a convergent sequence, there is an iteration j' such that $0 \leq \lambda^j - \lambda^{j'} < \epsilon'$ for all $j > j'$. Let Ξ_1, Ξ_2 be a separation of $\{\xi^{j'}, \xi^{j'+1}, \dots, \xi^{j'+K}\}$ as in Definition 5.15, with $|\Xi_1|, |\Xi_2| \geq |V| - 1$. Starting with $P_0 = v^{0,0}, e^{0,0}, v^{0,1}$, where $e^{0,0} = (v^{0,0}, v^{0,1}) := \arg\min\{c_{\pi^l}(e) \mid e \in E\}$ with $l := j' + K$, we construct a sequence $(P_k)_{k=\{0,\dots,K\}}$ of edge progressions with $P_0 \subseteq P_1 \subseteq \cdots \subseteq P_K$.

Thereby P_{k+1} is constructed from P_k, either by leaving it unchanged, $P_{k+1} = P_k$, or by augmenting P_k by an edge at one or both of its end vertices due to following rule. We start with $P_{k+1} \leftarrow P_k = (v^{k,0}, e^{k,0}, v^{k,1}, \dots, v^{k,|E(P_k)|-1}, e^{k,|E(P_k)|-1}, v^{k,|E(P_k)|})$.

If $\xi^{l-(k+1)} \geq \frac{1}{2}$ and P_k has an incoming edge $(u, v^{k,0})$ with

$$c_{\pi^{(l-(k+1))}}(u, v^{k,0}) \leq \lambda^{j'} + 4^{k+1}\epsilon',$$

we redefine $P_{k+1} \leftarrow (u, (u, v^{k,0}), P_{k+1})$, and

if $\xi^{l-(k+1)} \leq \frac{1}{2}$ and P_k has an outgoing edge $(v^{k,|E(P_k)|}, w)$ with

$$c_{\pi^{(l-(k+1))}}(v^{k,|E(P_k)|}, w) \leq \lambda^{j'} + 4^{k+1}\epsilon',$$

we redefine $P_{k+1} \leftarrow (P_{k+1}, (v^{k,|E(P_k)|}, w), w)$. Note that only if $\xi^{l-(k+1)} = \frac{1}{2}$, both ends may be augmented.

Each edge progression P_k, $k \in \{0, \dots, K\}$, fulfills the following property:

$$c_{\pi^{l-k}}(e) \leq \lambda^{j'} + 4^k\epsilon' \text{ for all } e \in E(P_k).$$

The prove is by induction on k. The claim is certainly true for the initial edge progression P_0, which contains the single edge $e^{0,0}$ with $c_{\pi^l}(e) = \lambda^l \leq \lambda^{j'} + \epsilon'$. An edge progression P_{k+1} contains all edges from P_k and potentially one or two additional edges. For all $e \in E(P_k)$ we have by induction hypothesis and Corollary 5.19

$$c_{\pi^{l-(k+1)}}(e) \leq \lambda^{j'} + 2 \cdot 4^k\epsilon' < \lambda^{j'} + 4^{k+1}\epsilon'.$$

If P_{k+1} contains a new edge $e \in E(P_{k+1}) \setminus E(P_k)$, then by construction $c_{\pi^{l-k+1}}(e) \leq \lambda^{j'} + 4^{k+1}\epsilon'$. As a consequence the reduced costs of the edges in P_K with respect to the node potential $\pi^{j'}$ after the j'-th iteration fulfill $c_{\pi^{j'}}(e) = c_{\pi^{l-K}}(e) \leq \lambda^{j'} + 4^K\epsilon' < \lambda^{j'} + \epsilon$.

As ξ^j is sufficiently balanced, there were at least $|\Xi_1| \geq |V| - 1$ attempts to augment the current edge progression by an incoming edge as well as $|\Xi_2| \geq |V| - 1$ attempts to augment by an outgoing edge. If P_K contains a cycle, we are done.

Otherwise, there must have been decisions not to add an incoming edge and decisions not to add an outgoing edge, and P_K is an s-t-path with $s, t \in V$.

Let $k_1 \in \{0, \dots, K\}$ be the maximum index such that $\xi^{l-k_1} \in \Xi_1$, and let $s = v^{k_1-1,0}$ be the first vertex in P_{k_1-1}. By Lemma 5.20, $\overline{\pi}(s) - \pi^{l-k_1}(s) \leq 2 \cdot 4^{k_1-1}\epsilon' < 4^{k_1}\epsilon'$. We have to show that this potential difference cannot grow above $4^k\epsilon'$ for $k > k_1$. Assume $\overline{\pi}(s) - \pi^{l-k_2}(s) > 4^{k_2}\epsilon'$, for some $K \geq k_2 > k_1$, and let k_2 be smallest possible. Then $\overline{\pi}(s) - \pi^{l-(k_2-1)}(s) \leq 4^{k_2-1}\epsilon'$, and by combining both inequalities

$$\pi^{l-(k_2-1)}(s) - \pi^{l-k_2}(s) > 3 \cdot 4^{k_2-1}\epsilon'.$$

We show that this large increase would push the reduced costs $c_{\pi^{l-(k_2-1)}}(s, w)$ of the first edge (s, w) in P_{k_2} above $\lambda^{j'} + 4^{k_2-1}\epsilon'$, a contradiction. We need to find an upper bound for $\Delta\pi^{l-(k_2-1)}(w)$. The largest increase of $\pi(w)$ can occur if it is not limited by $\overline{\pi}(w)$:

$$\begin{aligned}
\Delta\pi^{l-(k_2-1)}(w) &= \tfrac{1}{2}(c^-_{\pi^{l-k_2}}(w) - c^+_{\pi^{l-k_2}}(w)) &&\leq \tfrac{1}{2}(c^-_{\pi^{l-k_2}}(w) - \lambda^{j'}) \\
&\leq \tfrac{1}{2}(c_{\pi^{l-k_2}}(s, w) - \lambda^{j'}) &&\leq \tfrac{1}{2}4^{k_2}\epsilon' &&= 2 \cdot 4^{k_2-1}\epsilon'.
\end{aligned}$$

With the bounds for the potential changes in s and w we obtain.

$$\begin{aligned}
c_{\pi^{l-(k_2-1)}}(s, w) &\geq c_{\pi^{l-k_2}}(s, w) + \pi^{l-(k_2-1)}(s) - \pi^{l-k_2}(s) - \xi^{l-k_2}\Delta\pi^{l-(k_2-1)}(w) \\
&> c_{\pi^{l-k_2}}(s, w) + 3 \cdot 4^{k_2-1}\epsilon' - 2 \cdot 4^{k_2-1}\epsilon' \\
&\geq \lambda^{j'} + 4^{k_2-1}\epsilon'.
\end{aligned}$$

It follows that $\overline{\pi}(s) - \pi^{j'}(s) \leq 4^K\epsilon' < \epsilon$.

Analogously, it is shown that for the last vertex t in P_K with $\pi^{j'}(t) - \underline{\pi}(t) \leq \epsilon$.

□

With this lemma Theorem 5.16 follows straight away:

Proof of Theorem 5.16: According to Lemma 5.21, for every $\epsilon > 0$ there is an iteration j where λ^j is within an ϵ-neighborhood from an upper bound, which is given either as the average cost of a cycle in G or by the average cost of an s-t-path which is optimally balanced between its potential bounds at the endpoints $\overline{\pi}(s)$ and $\underline{\pi}(t)$.

□

Nevertheless, the overall reduced cost distribution may not converge towards an optimum distribution as defined in the Slack Balance Problem, as the example in Figure 5.6 shows. Let the edge labels denote the reduced costs on each edge. In this scenario the local balancing would do nothing, because the loops with a reduced cost of -2 (magenta edges) prevent any node potential changes. In an optimally balanced solution the blue edges would be balanced as well and end up with a reduced cost of 0 on both edges, for instance by shifting the node potential of A by +1. The hybrid approach described in Section 5.3.5 overcomes this limitation.

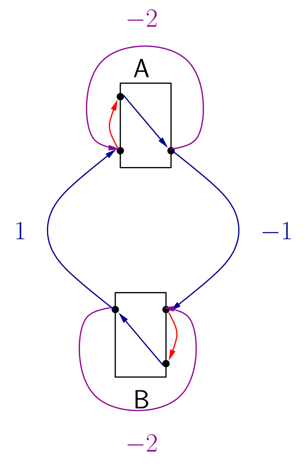


Figure 5.6: An unfavorable instance for iterative local balancing.

5.3.2 Convergence Rate

In general the convergence rate of Algorithm 8 is rather poor. For instance, when choosing the sufficiently balanced sequence $(\xi^j)_{j\in\mathbb{N}} = (\frac{1}{2})_{j\in\mathbb{N}}$ it cannot be better than linear. Recall the example from Figure 5.5, and consider the clock arrival time in B as fixed by upper and lower bounds. So that only A can be scheduled. Then the local shifts $\Delta\pi^j(\mathsf{A})$ of the clock input of A will be $\frac{1}{2^j}$, and the convergence rate is

$$\lim_{j\to\infty} \frac{|\lambda^* - \lambda^{j+1}|}{|\lambda^* - \lambda^j|} = \frac{1}{2}.$$

Kleff [2008] showed that with increasing cycle sizes this rate gets even worse. Let C_N be a cycle with N vertices and $\xi^j = \frac{1}{2}$ for all $j \in \mathbb{N}$. Then

$$\lim_{N\to\infty} \lim_{j\to\infty} \frac{|\lambda^* - \lambda^{j+1}|}{|\lambda^* - \lambda^j|} = 1.$$

However, in our experimental results in Section 5.4, we will see that the cycles in chip design are rather short, in the range of one to three vertices. In practice the worst slack is maximized very quickly.

5.3.3 Iterative Time Window Optimization

For the time window optimization, where the reduced costs of the early-late-separation edges is to be increased, the merge of early and late vertices must be reversed such that the set of nodes is $\{v_0, v_e^1, v_l^1, v_e^2, v_l^2, \dots, v_e^k, v_l^k\}$. Recall the notation of the early-late-separation edges $E_\eta = \{(v_l^i, v_e^i) \mid 1 \le i \le k\}$. The iterative time window optimization in Algorithm 9 first computes for each pair $(v_l^i, v_e^i) \in E_\eta$ the maximally allowed local increase of $c_\pi(v_l^i, v_e^i) = \pi(v_l^i) - \pi(v_e^i)$, in terms of a maximum feasible increase of $\pi(v_l^i)$ and and maximum feasible decrease of $\pi(v_e^i)$.

Then based on a damping factor $\xi \in [0, 1]$, these node potential changes are limited to globally feasible ones. Evidently this approach does not create a distribution of

Algorithm 9 Iterative Time Window Optimization

```
1: repeat
       /* Compute maximally allowed (local) time-window augmentations.*/
2:     for $(v_l, v_e) \in E_\eta$ do
3:         $c_\pi^-(v_l) \leftarrow \min_{\eta \in \{\text{early,late}\}} \min\{c_\pi(u, v_l) - \mathrm{S}_\mathrm{l}^\mathrm{tgt} \mid (u, v_l) \in \delta^-(v_l)\}$;
4:         $\Delta\pi(v_l) = \max\{0, c_\pi^-(v_l)\}$;
5:         $c_\pi^+(v_e) \leftarrow \min_{\eta \in \{\text{early,late}\}} \min\{c_\pi(v_e, w) - \mathrm{S}_\mathrm{e}^\mathrm{tgt} \mid (v_e, w) \in \delta^+(v_e)\}$;
6:         $\Delta\pi(v_e) = \max\{0, c_\pi^+(v_e)\}$;
7:     end for
       /* Carry out globally feasible time-window augmentations.*/
8:     for $(v_l, v_e) \in E_\eta$ do
9:         $\pi(v_l) = \pi(v_l) + \xi \cdot \Delta\pi(v_e)$;
10:        $\pi(v_e) = \pi(v_e) - (1 - \xi) \cdot \Delta\pi(v_e)$;
11:    end for
12: until $\sum_{v \in V(G)} |\Delta\pi(v)| < \epsilon$
```

time windows that is leximin maximal. But it converges towards a solution were no window can be widened further.

5.3.4 Implicit Implementation

The local shifts of a vertex potential $\pi(v)$ are computed based on the minimum reduced cost $c_\pi^-(v)$ of an incoming and the minimum reduced cost $c_\pi^+(v)$ of an outgoing edge (Algorithm 8 lines 5 and 6 and Algorithm 9 lines 3 and 5). Assume there are no (user-defined) tests between signals in the combinatorial logic. Furthermore, consider clock sinks to be contracted according to Remark 5.11 on page 94. Then $c_\pi^-(v)$ and $c_\pi^+(v)$ are given by the smallest slacks of the test arcs entering a clock pin v and the smallest slacks of the propagation arcs leaving the clock pin.

We do not need to construct the register graph explicitly but we can use the slacks on the arcs in the timing graph to compute the local shifts. Thereby only a small amount of data needs to be stored with each register. We give a slightly more flexible description that we will also use later in Section 5.3.5. The objects we consider are clusters and defined as follows.

Definition 5.22. *A cluster in iterative local clock skew scheduling consists of a set of pins and four edge sets (C, L_i, L_o, E_i, E_o), where*

- *$C \subset P_{cs}$ is the set of clocktree sinks,*
- *$L_i \subset E_t^{\mathcal{T}}$ is the set of test edges to compute $c_\pi^+(C)$ if $\eta = \text{late}$,*

- *$L_o \subset E_p^{\mathcal{T}}$ is the set of propagation edges to compute $c_\pi^-(C)$ if $\eta = \text{late}$,*
- *$E_i \subset E_t^{\mathcal{T}}$ is the set of test edges to compute $c_\pi^-(C)$ if $\eta = \text{early}$, and*
- *$E_o \subset E_p^{\mathcal{T}}$ is the set of propagation edges to compute $c_\pi^-(C)$ if $\eta = \text{early}$.*

The edges in $L_i \cup L_o \cup E_i \cup E_o$ are called cluster edge.

The worst reduced costs of an incoming edge and an outgoing edge are now computed implicitly through the worst slacks on the timing arcs. Recall the definitions (2.31), (2.32), and (2.33) in Section 2.4.8 of a timing arc slack $\text{slk}(p, \sigma_p, q, \sigma_q)$ of two signals $\sigma_p \in \mathfrak{S}(p), \sigma_q \in \mathfrak{S}(q), (p, q) \in E^{\mathcal{T}}$. Then the worst slack of an arc $(p, q) \in E^{\mathcal{T}}$ is $\min\{\text{slk}(p, \sigma_p, q, \sigma_q) \mid \sigma_p \in \mathfrak{S}(p), \sigma_q \in \mathfrak{S}(q)\}$.

Initially a cluster is created for each pin $p \in P_{cs}$. Some multi-sink clusters might be predefined. For instance, the two clock inputs of master-slave latches might form a common cluster (see Section 5.2.1 for a short description of master-slave latches).

L_o and E_o are the triggering propagation edges that are reached by a forward breadth-first or depth-first search in the propagation graph starting at the pins in C. This search is stopped whenever a triggering propagation segment is reached, which will be added to the sets L_o and E_o depending on the timing mode. Recall Section 2.4.2 for the definition of a triggering propagation segment. The input test arc sets L_i and E_i are the test arcs that enter some pin that was marked during the forward search. The set L_i consists of the late mode test arcs, where the head vertex is the early test end, while for E_i the head vertices are the late test end.

For a simple latch such as the one in Figure 2.5, this procedure would exactly create the sets $C = \{c\}$,
$L_o = \delta^-_{G^{\mathcal{T}}}(c) \cap \{e \in E_t^{\mathcal{T}} \mid e \text{ is labeled } (\text{late}, \zeta, \zeta')\}$,
$E_o = \delta^+_{G^{\mathcal{T}}}(c) \cap \{e \in E_p^{\mathcal{T}} \mid e \text{ is labeled } (\text{early}, \zeta, \zeta')\}$,
$L_i = \delta^+_{G^{\mathcal{T}}}(c) \cap \{e \in E_p^{\mathcal{T}} \mid e \text{ is labeled } (\text{late}, \zeta, \zeta')\}$, and
$E_i = \delta^-_{G^{\mathcal{T}}}(c) \cap \{e \in E_t^{\mathcal{T}} \mid e \text{ is labeled } (\text{early}, \zeta, \zeta')\}$.

During the course of the algorithm the local optimum schedules are computed based on the relevant edge sets. The arrival time changes $(\Delta\pi(C))$ are then applied to all signals in $\{\sigma \in \mathfrak{S}(c) \mid c \in C\}$. After all clusters were re-scheduled, a full timing analysis is performed, to obtain new valid slack values on the edges.

Remark 5.23. *For non-transparent flip-flops, the arrival times at the data output and the required arrival times at the data input depend only on the clock pin. In our implementation we measure simply the worst slacks at these pins instead of using the timing arcs.*

Note that this procedure measures the slack of loops, too. If the slack of a loop is the smallest slack of a cluster, the cluster will not be shifted.

Remark 5.24. *The algorithm balances the reduced costs of incoming and outgoing edges regardless of their values being already larger than zero. To improve the running time in practice, line 7 could be refined such that $\Delta\pi(v)$ is set to zero if $c_\pi^-(v) \geq 0$ and $c_\pi^+(v) \geq 0$. Thereby the node potential is unchanged if the worst reduced costs at the input and output are already above the slack target.*

User defined tests are not considered properly in this implicit implementation. To consider them correctly, they must be represented by explicit edges, as in the following hybrid balancing model.

5.3.5 Hybrid Balancing

The idea of the hybrid clock skew balancing is do combine iterative local balancing with the critical cycle contraction, and on-the-fly creation of the register graph edges. Algorithm 9 can be applied as a minimum mean cycle algorithm within the framework of Algorithm 7. Formally, we run Algorithm 8, where shift bounds are handled explicitly, until the worst slack is within an ϵ-neighborhood of the optimum worst slack. According to Lemma 5.21, the subgraph induced by the edges that are within an ϵ-neighborhood of the optimum consists of paths and cycles. Iteratively, each such path and cycle can be contracted into a new node $v^\star$ with initial node potential $\pi(v^\star) = 0$ and shift bounds

$$[\underline{\pi}(v^\star), \overline{\pi}(^\star)] := \bigcap_{v \in V(C)} [\underline{\pi}(v) - \pi(v), \overline{\pi}(v) - \pi(v)].$$

Costs of edges incident to $v^\star$ are adapted as in (5.6) and (5.7). Then Algorithm 8 is run to maximize the next worst slack.

Algorithm 10 Hybrid Local Minimum Balance Algorithm

Input: $(G, c), \eta \in \{\text{early}, \text{late}\}, \epsilon$.

```
repeat
    Run Algorithm 8 until worst slack is maximized;
    $\lambda = \min\{c_\pi \mid e \in E(\eta)\}$;
    while $\exists$ a path or cycle with an upper reduced cost bound below $(\lambda + \epsilon)$ do
        Contract that path or cycle into a new node $v^\star$;
    end while
until $\lambda \geq 0$
```

Theorem 5.25.
Let $G = (V, E_{\text{late}})$ be a graph with edge costs c and lower and upper bounds on the node potentials $\overline{\pi}, \underline{\pi} : V(G) \to \mathbb{R}$, and let $\pi^\star$ be an optimum solution with respect to the Slack Balance Problem, *which is defined as the* Minimum Balance Problem *on (G, c) extended by additional unparameterized arcs that model the hard lower and upper bounds on the node potential.*

Furthermore, let $\epsilon^\star > 0$ and π be the solution of the hybrid iterative local clock skew scheduling, running with $\epsilon < \frac{\epsilon^\star}{|V|}$. Then the reduced costs with respect to π are at most

$$c_\pi(e) \leq c_{\pi^\star}(e) + \epsilon^\star \text{ for all } e \in E_{\text{late}} \tag{5.32}$$

Proof. For paths and cycles that are contracted first (5.32) is certainly true. The sum of the reduced cost of the edges on any contracted sub-path of length k is at most $k\epsilon$. This error propagates into the adapted costs of edges incident to the new node according to (5.6) and (5.7).

By induction, the sum of reduced costs of each contracted subpath are within $\epsilon \cdot k'$, where k' is the path length in the uncontracted input graph. Thus in the end the maximum deviation on a path and thus edge is $\epsilon \cdot |V| < \epsilon^\star$.

□

As for the non-hybrid iterative local balancing algorithm this approach does not require the knowledge of the full register graph in advance. Instead, the adjacency edge-lists are needed only for the newly generated "contraction" nodes. In the implementation, the register graph is not given explicitly, but Definition 5.22 is extended to

Definition 5.26. *A hybrid cluster is a quintuple* (C, L_i, L_o, E_i, E_o), *where*

- $C \subset P_{cs}$ *is the set of clocktree sinks,*
- $L_i \subset E_t^{\mathcal{T}} \cup E^{\mathcal{R}}$ *is the set of test edges to compute* $c_\pi^-(C)$ *if* $\eta = \text{late}$,
- $L_o \subset E_p^{\mathcal{T}} \cup E^{\mathcal{R}}$ *is the set of propagation edges to compute* $c_\pi^+(C)$ *if* $\eta = \text{late}$,
- $E_i \subset E_t^{\mathcal{T}} \cup E^{\mathcal{R}}$ *is the set of test edges to compute* $c_\pi^-(C)$ *if* $\eta = \text{early}$, *and*
- $E_o \subset E_p^{\mathcal{T}} \cup E^{\mathcal{R}}$ *is the set of propagation edges to compute* $c_\pi^-(C)$ *if* $\eta = \text{early}$.

The edges in $L_i \cup L_o \cup E_i \cup E_o$ *are called hybrid cluster edges.*

Now, a new contraction node $v^\star$ is represented by a "hybrid" cluster, which is created by joining all clock sink sets into a new set $C^\star$, and replacing all edges by register graph edges from and to other clusters. For the new cluster we then have $L_i \subset E^{\mathcal{R}}$, $E_i \subset E^{\mathcal{R}}$, $L_o \subset E^{\mathcal{R}}$, and $E_o \subset E^{\mathcal{R}}$, while the edge sets of all other clusters remain unchanged. With this construction the cluster-internal slacks of paths between vertices in $C^\star$ are hidden, and $v^\star$ is scheduled with respect to the less critical slacks to its exterior.

5.4 Experimental Results

5.4.1 Memory Consumption

A big drawback of both graph model approaches is their huge memory demand. In case of the signal graph model it is still linear in the size of the static timing memory requirement, but this can already be too much. On large designs with more that 25 000 000 pins, and 4 signals for $\{\text{late}, \text{early}\} \times \{\text{rise}, \text{fall}\}$ per pin, this results already in 100 000 000 nodes. Even after pruning the graph (replacing paths by single edges and similar techniques) there would be about 20 000 000 nodes, with

10 000 000 clock nodes and 80 000 000 edges. Note that the number of nodes in the slack balance graph can double as a new node is inserted during each contraction in Algorithm 7. The register graph grows even more dramatically (quadratically in the number of registers in the worst case). In practice the number of edges tends to be 20–60 times the number of registers. Keeping the complete slack balance graph in memory is practically infeasible, besides the huge running time effort to compute all its edges.

The advantage of the iterative local scheduling method is that there is no need to generate a graph explicitly, but it can work on the timing graph directly without further memory overhead. Though its convergence rate is at most linear, it turns out that on clock scheduling instances the worst slack is maximized after a few iterations. In our experiments we observed that the algorithm usually converges faster than we can construct any slack balance graph. Table 5.1 shows a comparison of the memory consumption when balancing slacks combinatorially in the signal graph or with iterative local scheduling. For both cases the numbers include the memory demand of the chip data and timing engine, which was HalfTimer, as described in Schietke [1999], in the case of the signal graph and BonnTime, as described in Szegedy [2005b], for iterative balancing. In the iterative case this base memory is dominating the numbers, while the optimization structures require less then 2% of the memory consumption. The memory consumption of the signal graph implementation keeps a redundant intermediate data layer between the timing graph and the scheduling graph, so that the reported memory numbers can potentially be halved. However, the tendency is apparent. Based on these results the old implementations were

Chip	**# Circuits**	**Signal Graph**	**Iterative Balancing**
	in thousand	memory in GB	memory in GB
Hanno	433	3.39	1.18
Michael	693	6.26	1.53
Thomas	834	5.75	1.45
Bert	1032	10.00	2.34
Yvonne	1110	8.83	2.24
Salvina	1600	18.74	5.01
Josef	1678	24.38	3.70

Table 5.1: Memory Overhead of the signal graph model in comparison to the iterative local scheduling.

discarded some years ago, so that we do not have numbers from current designs.

5.4.2 Running Times

Table 5.2 shows the running times of the iterative implementations on the designs introduced in Section 2.7, obtained on a 2.93 GHz Intel Xeon E7220. The instances consist of placed netlists with optimized data paths. The iterative local balancing

Chip	**Iterative Local Balancing**			**Iterative Hybrid Balancing**		
	Late	Early	Clock	Late	Early	Clock
Minyi	2	4	3790	6	4	3832
Franz	8	5	209	9	5	65
Lucius	19	20	12	30	17	49
Julia	20	25	134	24	26	172
Tara	22	1201	211	27	2178	265
Felix	42	0	91	51	0	118
Fazil	49	6	44	103	7	33
Arijan	276	362	438	309	310	458
Maxim	617	26	315	1349	20	310
David	751	8678	279	1045	15434	307
Bert	826	372	47	1109	410	51
Ludwig	832	933	151	1063	1046	166
Karsten	1457	859	166	1749	880	192
Trips	3000	2341	251	3267	2357	252
Valentin	3672	1961	436	4189	1865	417

Table 5.2: Running Times in Seconds

was stopped after at most 250 iterations, when the maximum node-potential changes decreased below 1 ps, or when the average node-potential change decreased below $\frac{1}{1000}$ ps. The hybrid scheduling performed at most 250 contractions, which occurred seldom. The largest total running times occurred on David. The iterative local balancing takes 2 hours and 42 minutes, while the hybrid algorithm needs two more hours. It can be useful to call the late mode optimization incrementally within a late mode timing optimization flow. The running times for late mode optimization remained within one hour. On Valentin, where the largest late mode running time occurs, the worst slack was maximized already after half an hour, or 13 iterations of the algorithm. The average number of iterations until no worst slack register is scheduled any further is 9.5, with a maximum of 65 iterations on Fazil. Note that this is a sufficient condition for the worst slack maximization, which can already be maximized after a smaller number of iterations. However, we do not search for an explicit cycle after each iteration. On most instances the critical cycle contains only 1-3 registers and is found within 5 iterations.

5.4.3 Quality of the Slack Distribution

All approaches maximize the worst slack. In the signal graph model the slack distribution of the timing tests is optimized. The drawback of this approach is that in an optimum solution potentially all paths entering the critical cycle will have this same worst slack. Thus, the result can end up with many critical paths, which can degrade the sign-off timing results (see Section 2.5).

The register graph model creates a better distribution for sign-off timing, as register to register connections are optimized instead of timing tests. The hybrid iterative local scheduling does also optimize the slack distribution of register to register paths within an ϵ-error. In our experiments, it turned out that the benefit of the hybrid register graph generation over the non-hybrid local scheduling is negligible. Table 5.3 shows how the worst late slack (WS) and how the sum of negative late slacks (SNS), taken over all nets, are optimized by clock skew scheduling. The table shows the respective numbers for a (non-optimized) zero skew solution and after iterative local balancing (Algorithm 8) and after the iterative hybrid balancing (Algorithm 10).

Table 5.4 shows the corresponding numbers for early mode slacks. As early mode slacks might be impaired by late mode optimization, we added two additional columns showing the early mode numbers after late mode optimization just before early mode optimization. For the hybrid run these intermediate numbers look similar and are omitted in the table.

In our runs the slack targets were chosen as $\mathrm{S}_\mathrm{l}^\mathrm{tgt} = \mathrm{S}_\mathrm{e}^\mathrm{tgt} = 0.0$. The accuracy was chosen as the accuracy given by the timing rules, which is $\epsilon = 1$ picosecond. Between the non-hybrid and hybrid mode there is almost no difference. Only on Maxim and Trips the SNS number is slightly improved through the hybrid algorithm.

The effect of the hybrid optimization can better be measured when considering the slacks of the register graph edges. Figure 5.7 shows the 1 000 worst register edges colored by slack without clock schedule optimization, after the non-hybrid, and after the hybrid local scheduling for two of the instances. Apparently, there is hardly a difference between the non-hybrid and hybrid mode. Equivalent pictures could be drawn for any of the test cases. Observe the strongly connected critical components being nicely visible in the optimized pictures.

The reason for this is that situations like in Figure 5.6 (page 110) are rather hypothetical. In most cases there is only one or a few dominating clusters throughout the course of the hybrid algorithm. After a contraction is performed, the next critical cycle contains a single dominating cluster. This cycle will also be balanced optimally if the schedule of the dominating cluster is fixed, because all remaining registers on the cycle can move freely. The number of critical paths between two different clusters is usually very small. As a consequence it is sufficient to run the fast and memory efficient non-hybrid optimization in practice.

Early mode slacks might be significantly impaired by late mode scheduling. Table 5.4 shows that the sum of negative early mode slacks was worsened for most of the chips, compared to a zero skew schedule. For the chips Lucius, David, and Trips the worst early slack was worsened too.

This enforces the insertion of routing detours or repeater insertion for delay creation. However, this is very instance specific. To limit the worsening of early mode slacks, the balancing approach could be modified to disallow the decrease of early mode slacks below zero, as late constraints result in bounds during early mode optimization. Early mode slacks that are already below the target might be

Chip	**Zero Skew**		**Iterative Local Balancing**		**Iterative Hybrid Balancing**	
	WS	**SNS**	**WS**	**SNS**	**WS**	**SNS**
Franz	-0.130	-0	-0.109	-0	-0.109	-0
Minyi	-0.736	-0	-0.047	0	-0.047	0
Lucius	-0.640	-6	-0.638	-5	-0.638	-5
Julia	-2.066	-6	-0.055	-0	-0.055	-0
Tara	-0.745	-8	-0.121	-0	-0.121	-0
Felix	-0.891	-25	-0.858	-10	-0.858	-10
Fazil	-1.525	-17	-1.216	-11	-1.216	-11
Bert	-0.897	-33	-0.618	-3	-0.618	-3
Maxim	-1.564	-222	-1.131	-21	-1.131	-19
Arijan	-2.155	-25	-2.155	-4	-2.155	-4
Ludwig	-11.313	-622	-11.313	-223	-11.313	-223
David	-1.795	-77	-1.610	-17	-1.610	-17
Karsten	-4.465	-888	-3.463	-212	-3.463	-212
Trips	-2.356	-692	-1.988	-233	-1.988	-232
Valentin	-1.795	-1747	-1.625	-320	-1.625	-320

Table 5.3: Optimizing late mode slacks with WS in ns and SNS in μs (rounded)

Chip	**Zero Skew**		**Iterative Local Balancing**				**Iterative Hybrid Balancing**	
			After Late Opt		**After Early Opt**			
	WS	**SNS**	**WS**	**SNS**	**WS**	**SNS**	**WS**	**SNS**
Franz	-0.095	-0	-1.400	-5	-0.095	-0	-0.095	-0
Minyi	-0.115	-0	-1.035	-1	-0.037	-0	-0.037	-0
Lucius	-0.267	-1	-1.739	-38	-0.798	-4	-0.798	-4
Julia	-1.335	-0	-3.061	-10	-0.935	-0	-0.936	-0
Tara	-1.568	-42	-2.960	-60	-1.110	-35	-1.111	-36
Felix	0.103	0	-0.051	-0	0.000	-0	-0.001	-0
Fazil	0.128	0	-1.053	-0	-0.045	-0	-0.051	-0
Bert	-5.035	-15	-5.035	-60	-5.035	-17	-5.035	-17
Maxim	-0.146	-0	-0.146	-0	0.000	-0	0.000	-0
Arijan	-4.142	-0	-4.142	-68	-4.142	-0	-4.142	-0
Ludwig	-31.618	-23	-31.618	-617	-31.618	-133	-31.618	-133
David	-1.561	-151	-3.544	-685	-2.059	-149	-2.060	-149
Karsten	-5.063	-1	-5.063	-962	-5.063	-157	-5.063	-157
Trips	-0.562	-86	-3.454	-2127	-1.478	-138	-1.478	-139
Valentin	-24.985	-1	-24.985	-948	-24.985	-160	-24.985	-161

Table 5.4: Optimizing early mode slacks (WS in ns and SNS in μs)

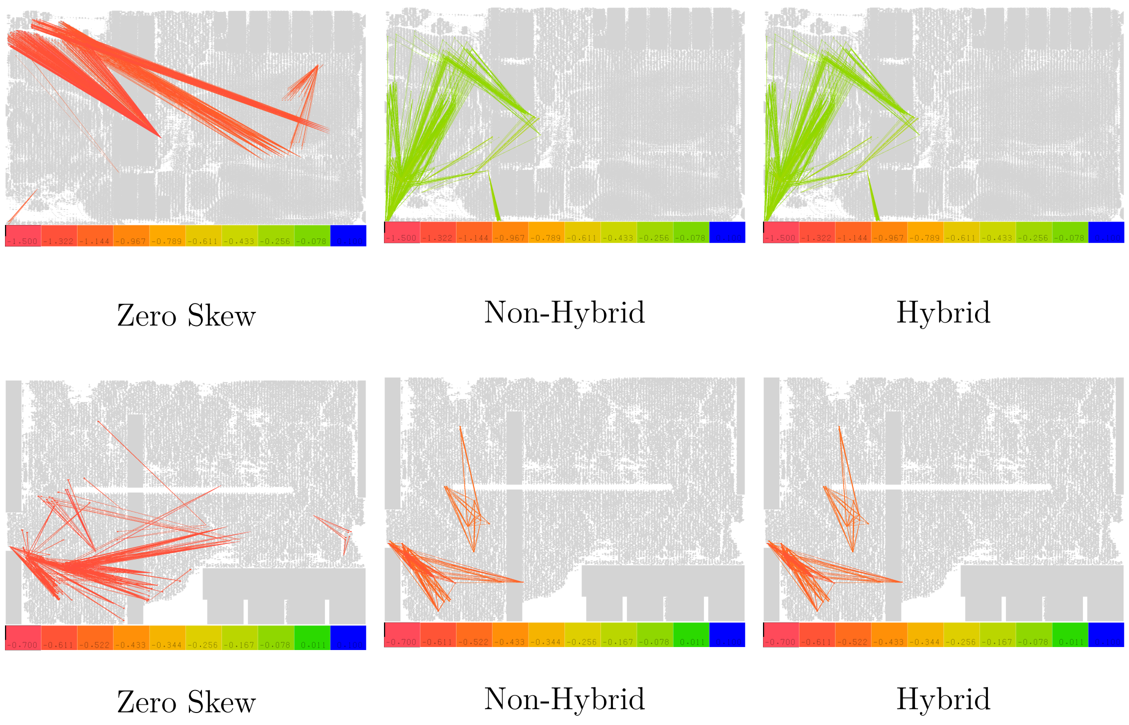


Figure 5.7: Improvement of late mode slacks of 1 000 worst register edges on Maxim (top) and Felix (bottom). Edges are colored by slack from red (critical) to blue (above the target).

decreased further, because delay buffer need to be inserted anyway. Note that most assertions emerge from a design stage, where early mode slacks were of no interest. Thus they may contain some inaccuracies, for instance on the chips Ludwig and Valentin.

5.4.4 Other Aspects

Another advantage of the iterative local scheduling approach is that it is tolerant to varying delays. As described in Section 2.4.4, the slews propagated to a pin, and thus the delays in the forward cone, vary with varying arrival times at the predecessors. Consequently, the arc delays are not constant when applying clock skew scheduling. The graph models do not account for this effect, and may end with worse results than expected. In contrast, the iterative local clock skew scheduling performs a new timing analysis after each iteration. Thus the delays and slacks are always up-to-date, and the next iteration automatically compensates the effect of varying delays.

Even more advantageous is the fact that the local scheduling can easily be called incrementally with other tools that optimize the delays, such as gate sizing or buffering. The graph models need to recompute the edge costs every time clock scheduling is invoked. Especially for the register graph model this corresponds to a re-computation of longest path delays between all connected register pairs, and thus a complete rebuild of the graph. The iterative local scheduling can simply continue based on its data structures given in Definition 5.22.

The advantage of the iterative time-window optimization in Algorithm 9 is that time windows of non-critical registers are increased from the first iteration on. In contrast, Algorithm 7 is wasting its running time in identifying and contracting the smallest time-windows, which stay small anyway.

5.5 Notes on Clocktree Synthesis

Once an optimum schedule is computed it needs to be realized by a clocktree. This problem is related to the Repeater Tree Problem, as in both cases a signal has to be distributed by a repeater tree. However, the construction of clocktrees is significantly more complicated as not only maximum required arrival times, but also minimum required arrival times need to be preserved. For the sake of completeness, we summarize existing approaches for clocktree construction in this section.

The input to clocktree construction is almost the same as the Repeater Tree Instance on page 35. The most important difference is that there is no slew dependent maximum required arrival time $\mathrm{rat}(s, \zeta, slew)$ for each sink $s \in S$, but a slew dependent arrival time window $[l_s(\zeta, slew), u_s(\zeta, slew)]$ for the required tree latency, $\zeta \in \{\mathrm{rise}, \mathrm{fall}\}, slew \in \mathbb{R}_+$. The windows for the individual sinks are defined relative to each other, which means that a signal starting at the tree root r must arrive at each sink $s \in S$ within its time window, after adjusting the start time at r by an adequate latency value. The larger a sink slew is, the smaller its time window will be, because the setup and hold times increase with the input slew, and thus the underlying timing tests are tightened. Often the time window constraint is restricted to a common window $[l_s, u_s]$ for rising and falling signals which is feasible for all slews of transition $\zeta \in \{\text{rise}, \text{fall}\}$ which are below some slew limit.

The goal is to synthesize a clocktree which ensures that all clock signals arrive within the specified time intervals. The robustness with respect to process variations, which is often measured in terms of the tree latency, and the power consumption are secondary and tertiary objectives. Depending on the application the order of the objectives may change or some weighted mean is chosen.

In a variant of the problem there can be several tree roots instead of a single root, for instance the endpoints of a toplevel H-tree or wires of a clock grid can be roots of bottom-level clocktrees.

An additional constraint is the clock pulse width, which is the latency difference between the rising and the falling signal. This difference must be kept small at every pin within the clocktree.

Traditionally the individual delay constraints were met by balancing wires as proposed by Chao et al. [1992]. This approach is known as the Deferred-Merge-Embedding Algorithm (DME). A huge amount of literature is based on the DME-Algorithm. Theoretically very exact delay targets can be met by tuning wire delay. The drawback is that it often requires a lot of wiring resources, because intended extra delay must be created by wiring detours. Furthermore, the prescribed wiring layouts are hard to achieve in detailed routing, and due to increasing wiring resistances in new technologies delays increase significantly.

Held et al. [2003] proposed a different approach, which assumes that all inserted nets will be routed with minimum length. Delays are balanced by the constructed tree topology and accurate cell sizing. We will explain this approach a bit more in detail.

As inverters typically yield smaller latencies, we will consider only inverters and no buffers. In a preprocessing step maximum inverter spacings as well as capacitance and slew targets are computed according to Section 3.3. Furthermore, a blockage grid is created as in Section 3.8.1. Then the approximate minimum delays to a source from every point on the chip are computed, taking into account that some macros can prevent the tree from going straight towards a source. This results in a shortest delay tree on the grid tiles, which will guide the topology construction towards the tree source.

The tree is constructed in bottom-up fashion summarized in Algorithm 11. While the topology of the tree is constructed, placeholder inverters are inserted and placed at every internal node. In contrast to Chapter 3, a preliminary topology T consists of a branching with arbitrary high out-degrees. Furthermore, there is a bijection between the internal vertices and the inverters in the final solution. However, the inverter sizes are not fixed before the complete topology is known. Instead a set of solution candidates is maintained at each vertex $v \in V(T)$.

Each solution candidate *sc* is associated with an inverter size, an input slew, a

Algorithm 11 BonnClock-Algorithm

```
1: Active vertices $V_{act} := S$;
2: while $V_{act}$ cannot be connected to $r$ do
       /* Clustering */
3:     Create a new vertex $v'$;
4:     Find a cluster $V' \subseteq V_{act}$ for $v'$;
5:     $V_{act} \leftarrow V_{act} \setminus V'$;

       /* Placement */
6:     Find the final placement $\mathrm{Pl}(v)$ for all $v \in V'$ within $\mathrm{Pl}_{prelim}(v)$;
7:     Refine the feasible predecessor areas $\mathrm{Pl}_{pred}(v)$ for all $v \in V'$;
8:     Compute the preliminary placement area $\mathrm{Pl}_{prelim}(v') := \bigcap_{v \in V'} \mathrm{Pl}_{pred}(v)$;
9:     Compute the predecessor placement area $\mathrm{Pl}_{pred}(v')$;

       /* Dynamic Programming Update */
10:    Refine the solution candidates for $v \in V'$ based on (the final) $\mathrm{Pl}(v)$;
11:    Compute the solution candidates for $v'$;
12: end while
13: Chose the solution candidate at $r$ that minimizes the objective function;
```

feasible arrival time interval $[l_{sc}, u_{sc}]$ for the input, and a solution candidate for each successor. Dominated candidates, whose time intervals are contained in the time intervals of other solution candidates with the same input slews, are pruned.

Given the set of solution candidates for each successor, a set of solution candidates for a newly inserted inverter is computed as follows. For each input slew at the successors we simultaneously scan the corresponding candidate lists in the natural order and choose maximal intersections of their time intervals. For such a non-dominated candidate set we try all inverter sizes and a discrete set of input slews and check whether they generate the required input slews at the successors. If so, a new candidate is generated. The input slew discretization enforces the acceptance of solutions, whose discretized input slew is some ϵ off the real final slew, where ϵ is the granularity of the discretization.

When an inverter (a new node v' in the notation of Algorithm 11) is inserted, its position $\mathrm{Pl}_{prelim}(v')$ is not fixed to a single location, in order to maintain a maximum freedom for the potential area $\mathrm{Pl}_{pred}(v')$ of its predecessor, which will be determined during a later clustering step. Instead $\mathrm{Pl}_{prelim}(v')$ is given as the intersection $\bigcap_{v \in V_{act}} \mathrm{Pl}_{pred}(v)$ of the predecessor areas of its successors. The area $\mathrm{Pl}_{pred}(v')$ is then defined as the set of octagons containing all feasible positions (Figure 5.8). From all points that are within the maximum inverter spacing (magenta bounded area) from the preliminary placement area $\mathrm{Pl}_{prelim}(v')$ of v' (blue area), unusable areas (for instance, those blocked by macros) and points that are too far away from the source are subtracted. The resulting area $\mathrm{Pl}_{pred}(v')$ (green area) can

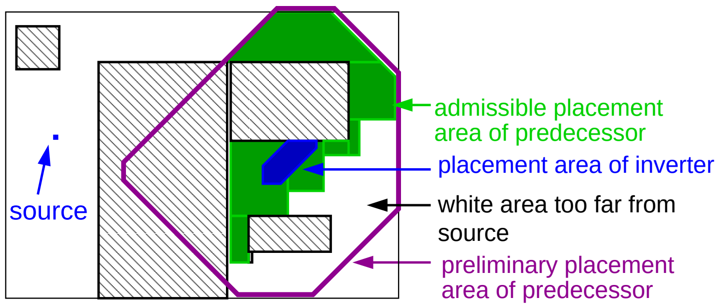


Figure 5.8: Computation of $\mathrm{Pl}_{pred}(v')$ *for the newly inserted inverter* $v' \in V_{act}$.

again be represented as a union of octagons.

The inverter sizes are selected when the final topology is found by choosing a solution candidate at the root. The best solution candidate with respect to

1. arrival time interval matching,
2. tree latency, and
3. power consumption

is chosen. Due to discretizing slews, assuming bounded RC delays, and legalization, the timing targets may be missed by a small amount, in the order of 20 *ps*. But this impacts the overall timing result only if the deviation occurs in opposite directions at the ends of a critical path.

For the clustering (lines 3–5 in Algorithm 11), a greedy style clustering (Algorithm 12) was proposed in Held et al. [2003]. For each vertex $v \in V_{act}$, the interval $[l_v, u_v]$ is defined as the bounding box of the time windows of its solution candidates. The greedy algorithm tries to find a cluster of active vertices $V' \subseteq V_{act}$ that maximizes the intersection of predecessor areas times the intersection of the time intervals:

$$\mathrm{volume}\left(\bigcap_{\tilde{v}\in V'\cup\{v\}} \mathrm{Pl}_{pred}(\tilde{v}) \times \bigcap_{\tilde{v}\in V'\cup\{v\}} [l_{\tilde{v}}, u_{\tilde{v}}]\right).$$

Maßberg and Vygen [2008] improved the greedy style clustering by modeling and solving it as a special facility location problem that partitions all active vertices from V_{act} into clusters. They proposed a constant factor approximation algorithm that has still a very fast running time bound of $O(|S| \log |S|)$. This global clustering improves the power consumption by 10–15 percent. It is especially effective in the leaf levels, which account for most of the power consumption within a clocktree.

Figure 5.9 shows a gigahertz clock on the chip Bert. It was designed for 900 Mhz but is used in hardware with 1033 Mhz. The speed of the chip was improved by more than 30% using clock skew scheduling in form of Algorithm 7 on the timing graph model, and by constructing the scheduled tree with BonnClock (Algorithm 11). Each net in the figure is represented by a star connecting the source to all sinks.

Algorithm 12 BonnClock—Greedy Clustering

```
$V' \leftarrow \emptyset$;
Chose $v \in V_{act}$ that has maximum $[l_v, u_v]$;
**while** $V' \cup \{v\}$ is feasible **do**
    $V' \leftarrow V' \cup \{v\}$;
    $V_{act} \leftarrow V_{act} \setminus \{v\}$;
    Chose $v \in V_{act}$ that
      1) has the same parity as vertices in $V'$ and
      2) maximizes
        $$\text{volume}\Big(\bigcap_{\tilde{v} \in V' \cup \{v\}} \text{Pl}_{pred}(\tilde{v}) \times \bigcap_{\tilde{v} \in V' \cup \{v\}} [l_{\tilde{v}}, u_{\tilde{v}}]\Big);$$
**end while**
```

Colors indicate the arrival times of the signals at each circuit. They range from very early times (blue) close to the source over green and yellow to very late times (red). The leaf level arrival times vary (intendedly) by 600 ps, while the tree latency is roughly 3 ns, which means that signals from three subsequent clock cycles travel through the tree simultaneously.

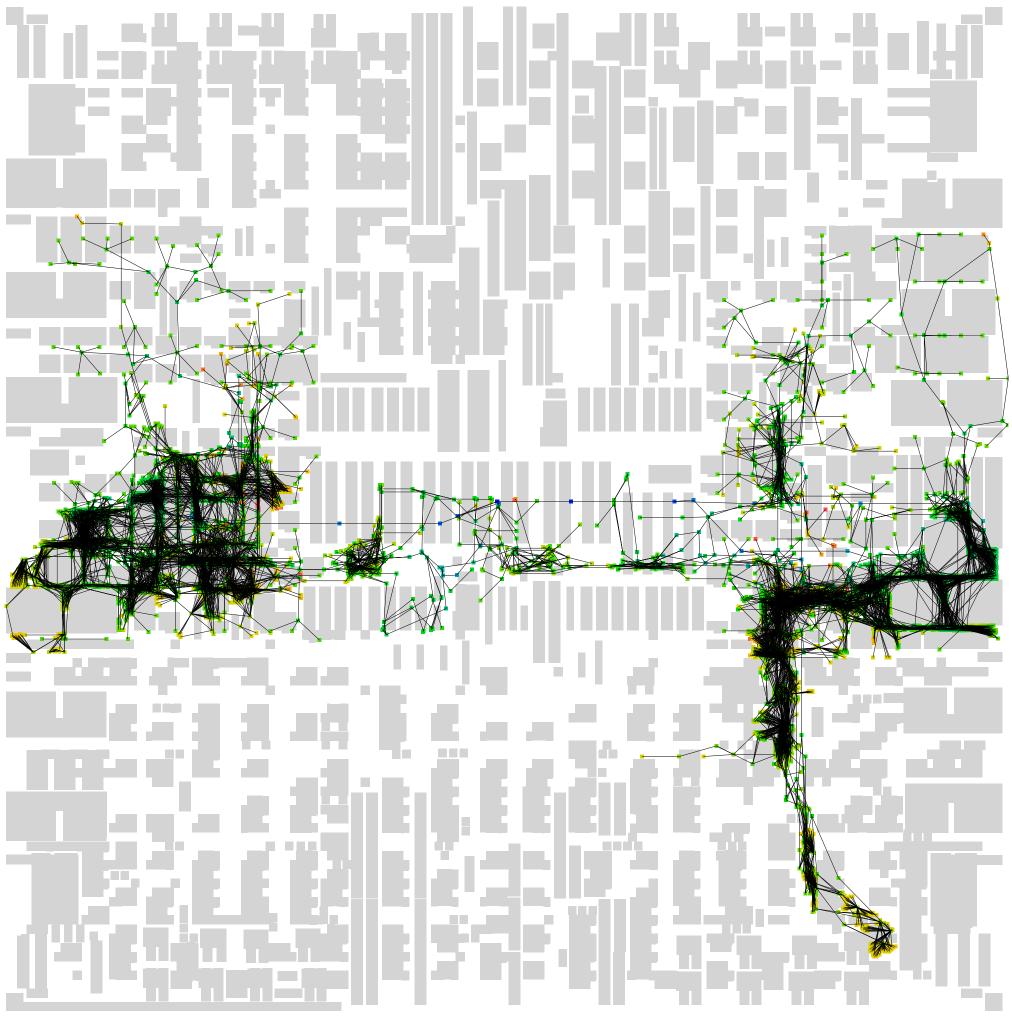


Figure 5.9: Gigahertz clock on the chip Bert.

6 Time-Cost Tradeoff Problem

In this section we present a new method to compute the time-cost-tradeoff curves of linear time-cost tradeoff instances in general graphs with cycles. It is a generalization of a method for acyclic instances by Phillips and Dessouky [1977].

We will apply this method as an approximation to discrete timing optimization operations such as threshold voltage optimization or plane assignment. The clou of this approach is that it integrates delay optimization and clock skew scheduling. Other operations, especially circuit sizing, influence many arcs and do not fit well into this scheme.

6.1 Problem Formulation

Traditionally linear acyclic time-cost tradeoff instances are given as partially ordered sets (*posets*) of jobs or activities. However, they are usually transformed into equivalent problem formulations on acyclic *activity-on-edge networks*.

Here, we consider the generalization to non-acyclic activity-on-edge networks, which due to the cycles do not have an underlying poset formulation. The instances we consider are already given as activity-on-edge networks. Therefore, we skip any order theoretic formulation and start directly with a network formulation.

Definition 6.1. *A time-cost tradeoff instance is a quadruple* $(G, \mathcal{T}, (c_e)_{e\in \mathrm{E_p}}, \mathrm{E_t})$ *with following properties.* G *is a directed graph whose edges represent certain jobs (activities). For each job* $e \in E(G)$ *there is a non-empty set* $\mathcal{T}(e) \subseteq \mathbb{R}$ *of possible execution times or delays and a strictly decreasing cost function* $c_e : \mathcal{T}(e) \to \mathbb{R}_{\geq 0}$.

Furthermore, there is a subset $\mathrm{E_t} \subseteq E(G)$ *of test edges. Every cycle* $C \subseteq G$ *with* $\sum_{e\in E(C)} \max_{\delta\in\mathcal{T}(e)} \delta > 0$ *must contain at least one test edge,* $|E(C) \cap \mathrm{E_t}| > 0$*. The edges from* $\mathrm{E_p} := E(G) \setminus \mathrm{E_t}$ *are called propagation edges.*

An instance is called *linear* if $\mathcal{T}(e) = [l_e, u_e]$, $l_e, u_e \in \mathbb{R}$ with $l_e \leq u_e$, and c_e is linear for all $e \in E(G)$. It is called *discrete* if $\mathcal{T}(e)$ is finite for all $e \in E(G)$.

The edges $(v, w) \in E(G)$ define some precedence relation with the meaning that an event in w must not start before an event started in v plus some given delay $\vartheta(v, w) \in \mathcal{T}(v, w)$:

Definition 6.2. *An assignment of start times* $\mathrm{at} : V(G) \to \mathbb{R}$ *and execution times* $\vartheta : E(G) \to \mathbb{R}$ *is feasible if* $\vartheta(e) \in \mathcal{T}(e)$ *and*

$$\mathrm{at}(v) + \vartheta(v, w) \leq \mathrm{at}(w) \tag{6.1}$$

for all $e = (v, w) \in E(G) \setminus \mathrm{E_t}$.

For test edges $(v,w) \in \mathrm{E_t}$, (6.1) may be violated. However, we are interested in their slack $\mathrm{slk}(v,w)$:

$$\mathrm{slk}(v,w) := \mathrm{at}(w) - \mathrm{at}(v) - \vartheta(v,w). \tag{6.2}$$

Given a feasible pair (at,ϑ) the worst slack $\mathrm{slk}(\mathrm{at},\vartheta)$ is given by

$$\mathrm{slk}(\mathrm{at},\vartheta) := \min_{e\in \mathrm{E_t}} \mathrm{slk}(e),$$

and the total cost $c(\mathrm{at},\vartheta)$ is given by

$$c(\mathrm{at},\vartheta) = c(\vartheta) = \sum_{e\in E(G)} c_e(\vartheta(e)).$$

The following two problems are usually considered on time-cost tradeoff instances: the Budget Problem and the Deadline Problem.

Budget Problem

Input: A time-cost tradeoff instance $(G,\mathcal{T},(c_e)_{e\in E(G)},\mathrm{E_t})$, and a budget $B \in \mathbb{R}_+$.

Output: Find a feasible pair (at,ϑ) that maximizes the worst slack among those whose cost is at most B. We denote this maximum possible worst slack by:

$$\begin{aligned} \mathrm{S}_{opt}(\mathrm{B}) &:= \max\ \mathrm{slk}(\mathrm{at},\vartheta) \\ \text{such that} & \\ (\mathrm{at},\vartheta) &\quad \text{is feasible and} \\ c(\vartheta) &\le \mathrm{B}\,. \end{aligned} \tag{6.3}$$

Deadline Problem

Input: A time-cost tradeoff instance $(G,\mathcal{T},(c_e)_{e\in \mathrm{E_p}},\mathrm{E_t})$, and a deadline or worst slack $\mathrm{S} \in \mathbb{R}$.

Output: Find a feasible pair (at,ϑ) that minimizes the costs among those whose worst slack is at least S. We denote this minimum cost by

$$\begin{aligned} \mathrm{B}_{opt}(\mathrm{S}) &:= \min\ c(\mathrm{at},\vartheta) \\ \text{such that} & \\ (\mathrm{at},\vartheta) &\quad \text{is feasible and} \\ \mathrm{slk}(\mathrm{at},\vartheta) &\ge \mathrm{S}\,. \end{aligned} \tag{6.4}$$

Obviously, both problems are linear programs if $(G,\mathcal{T},(c_e)_{e\in \mathrm{E_p}},\mathrm{E_t})$ is a linear instance. The following definition characterizes solutions whose slack can only be improved by increasing the cost, and whose cost can only be decreased by lowering the slack.

Definition 6.3. *A feasible realization* (at, ϑ) *is called time-cost optimum if*

$$\mathrm{slk}(\mathrm{at}, \vartheta) = \mathrm{S}_{opt}(c(\vartheta)) \tag{6.5}$$

and

$$c(\vartheta) = \mathrm{B}_{opt}(\mathrm{slk}(\mathrm{at}, \vartheta)). \tag{6.6}$$

The time-cost tradeoff problem is to find all time-cost optimum realizations:

Time-Cost Tradeoff Problem	
Input:	A time-cost tradeoff instance $(G, \mathcal{T}, (c_e)_{e \in E(G)}, \mathrm{E_t})$.
Output:	Determine the set of feasible time-cost optimum realizations.

In the linear case, the cost/slack pairs of time-cost optimum solutions are located on a piecewise linear so called *time-cost tradeoff curve*, within the compact interval $[\underline{\mathrm{S}}, \overline{\mathrm{S}}]$ for the slack, defined by the best possible slack $\underline{\mathrm{S}}$ of a lowest-cost solution and the best possible slack $\overline{\mathrm{S}}$ at any cost. This curve is equivalently defined by B_{opt} or S_{opt} respectively, as in the example in Figure 6.1. This property follows from the

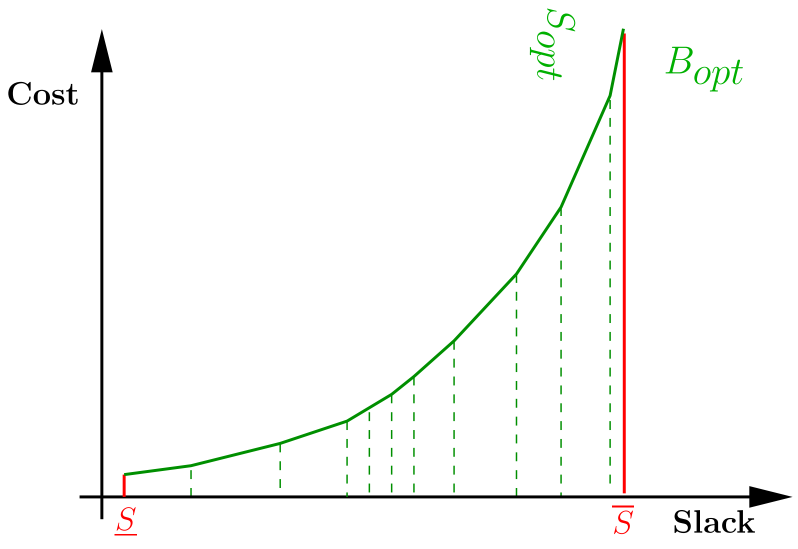


Figure 6.1: An example of a (linear) time-cost tradeoff curve. The points on the piecewise linear solid green line are the time-cost optimum solutions.

simple fact that both functions B_{opt} and S_{opt} are parametric linear programs. The *linear time-cost tradeoff problem* is to compute this curve.

6.2 Related Work

The traditional *time-cost tradeoff problem* was introduced by Kelley Jr. [1961] and Fulkerson [1961]. It consists of activities that are related by a partial order, possible execution times are closed intervals ($\mathcal{T}(e) = [l_e, u_e]$ for all $e \in E(G)$) and costs are linear functions. It was shown that they can be represented by an activity-on-edge diagram, whose edge number is polynomially bounded in input size of the partial order (see Skutella [1998] for details). Such an activity-on-edge diagram corresponds to a time-cost tradeoff instance where the propagation graph $(V(G), \mathrm{E_p})$ is acyclic and $|\,\mathrm{E_t}\,| = 1$. There is a one-to-one correspondence between the worst slack on the single test edge and the longest path delay through the acyclic propagation graph. The name Deadline Problem originates from a given deadline for the project finish time given by the longest path delay.

The linear deadline problem on acyclic graphs can be solved in polynomial time by a minimum cost flow algorithm (for example see Lawler [1976] chapter 4.13). This approach can be extended to cyclic graphs (see Levner and Nemirovsky [1994], or Vygen [2001] chapter 5.5). The fastest algorithm for piecewise linear convex costs was given by Ahuja et al. [2003].

Unfortunately a minimum cost flow formulation is only known for the Deadline Problem. The Budget Problem could be solved by a linear programming solver or by binary search over S and solving the Deadline Problem by a minimum cost flow algorithm for each decision.

The discrete versions of the acyclic problem, where $\mathcal{T}(e)$ is a discrete set of points, were shown to be strongly *NP*-hard by De et al. [1997]. Approximation algorithms are currently only available for some special cases of the Budget and Deadline Problem (see Skutella [1998]). Deineko and Woeginger [2001] considered the hardness of the bicriteria (cost and slack) approximation problem.

6.3 A Combinatorial Algorithm

We present the first combinatorial algorithm to solve the linear Time-Cost Tradeoff Problem for (linear) instances $(G, \mathcal{T}, (c_e)_{e\in \mathrm{E_p}}, \mathrm{E_t})$, where G may contain cycles. From now on we consider each instance to be linear unless the instance type is noted explicitly.

With this formulation we model and solve practical problems from VLSI-design. Here the graph represents the signal propagation through the netlist. Cycles are induced by registers whose switching times can be scheduled. Registers are represented by test arcs where we measure whether the signal arrives before the next cycle starts. Their delays will be negative as they contain a negative adjust by the cycle time.

Improving the speed, that is, increasing the worst slack, involves a higher power consumption. We want to stop improving speed if either a maximum budget is reached, or all timing restrictions are met, and thus we have found a feasible delay and start time assignment according to Definition 6.2.

Without loss of generality, we consider only instances where all test edges $e \in \mathrm{E_t}$ have a fixed delay $\mathcal{T}(e) = \{0\}$. General instances can be transformed by replacing each $e = (v, w) \in \mathrm{E_t}$ by a new vertex v' and two new edges $e' = (v, v') \in E(G) \setminus \mathrm{E_t}$

and $e'' = (v', w) \in \mathrm{E_t}$ with $\mathcal{T}(e') := \mathcal{T}(e), c_{e'} := c_e, \mathcal{T}(e'') := \{0\}$, and $c_{e''} := 0$.

The function B_{opt} can be determined by a parametric linear programming problem formulation, with S as parameter, and is therefore piecewise linear, convex, and continuous. Analogously the inverse function S_{opt} is piecewise linear, concave and continuous.

The algorithm combines ideas from Phillips and Dessouky [1977] with minimum ratio cycle algorithms by Megiddo [1983] and Young et al. [1991]. Note that in contrast to classic shortest path theory we deal with longest delay paths. Here edge delays are *feasible* if no positive delay cycle exists. We start with some useful Lemmata.

6.3.1 Preliminary Considerations

Given fixed delays, a start time assignment that maximizes the worst slack can be computed by a minimum ratio cycle computation:

Lemma 6.4. *Let $(G, \mathcal{T}, (c_e)_{e \in \mathrm{E_p}}, \mathrm{E_t})$ be a time-cost tradeoff instance, and let ϑ be a delay assignment. Then, the highest achievable worst slack $\mathrm{S}(\vartheta)$ for ϑ is given by the value of a minimum ratio cycle*

$$\min \left\{ \frac{\sum_{e \in E(C)} -\vartheta(e)}{|E(C) \cap \mathrm{E_t}\,|} \mid C \subseteq G \text{ cycle with } E(C) \cap \mathrm{E_t} \neq \emptyset \right\}.$$

It can be computed in $O(nm + n^2 \log n)$ time.

Proof. Define $p : E(G) \to \{0, 1\}$ by $p(e) = 1$ if and only if $e \in \mathrm{E_t}$. Then we look for the largest value s such that there exists arrival times at : $V(G) \to \mathbb{R}$, with

$$\mathrm{at}(v) + \vartheta(v, w) + s \cdot p(v, w) \leq \mathrm{at}(w) \quad \forall (v, w) \in E(G).$$

This is equivalent to

$$\sum_{e \in E(C)} \Big(\vartheta(e) + s \cdot p(e) \Big) \leq 0$$

for all cycles $C \in E(G)$ with $E(C) \cap \mathrm{E_t} \neq \emptyset$. Resolving all inequalities by s proves the statement. By Theorem 5.5 a minimum ratio cycle can be computed in $O(nm + n^2 \log n)$ as $w_{\max} = 1$.

□

With the above considerations we obtain the following corollary.

Corollary 6.5. *Let $(G, \mathcal{T}, (c_e)_{e \in \mathrm{E_p}}, \mathrm{E_t})$ be a time-cost tradeoff instance. Then the highest achievable worst slack $\overline{\mathrm{S}}$ is given by the minimum ratio delay of a cycle:*

$$\overline{\mathrm{S}} = \min \left\{ \frac{\sum_{e \in E(C)} -\min\{\delta \mid \delta \in \mathcal{T}(e)\}}{|E(C) \cap \mathrm{E_t}\,|} \mid C \subseteq G \text{ cycle with } E(C) \cap \mathrm{E_t} \neq \emptyset \right\} \tag{6.7}$$

The worst slack $\underline{\mathrm{S}}$ *of a lowest-cost solution is given by a the minimum ratio delay cycle:*

$$\underline{\mathrm{S}} = \min\left\{ \frac{\sum_{e\in E(C)} - \max\{\delta \mid \delta \in \mathcal{T}(e)\}}{|E(C) \cap \mathrm{E_t}\,|} \,\middle|\, C \subseteq G \textit{ cycle with } E(C) \cap \mathrm{E_t} \neq \emptyset \right\} \tag{6.8}$$

The domain $dom(\mathrm{B}_{opt}) = [\underline{\mathrm{S}}, \overline{\mathrm{S}}]$ *can be computed in* $O(nm + n^2 \log n)$ *time.*

□

The next lemma allows us to consider only the subgraph of minimum ratio cycles when we want to improve the worst slack.

Lemma 6.6. *Let* $(G, \mathcal{T}, (c_e)_{e\in \mathrm{E_p}}, \mathrm{E_t})$ *be a time-cost tradeoff instance and a let* ϑ *be the delay assignment of a time-cost optimum solution. Then the delay* $\vartheta(e)$ *of an arc* $e \in E(G)$ *that is not located on a minimum ratio cycle, according to Lemma 6.4, is assigned to its highest possible value, that is,* $\vartheta(e) = \max\{\,\delta \mid \delta \in \mathcal{T}(e)\}$.

Proof. Assume there is an arc $e' \in E(G)$ for which the statement does not hold ($\vartheta(e') < \max\{\delta \mid \delta \in \mathcal{T}(e)\}$) and e' is not located on a minimum ratio cycle. If $E(C') \cap \mathrm{E_t} = \emptyset$ for all cycles $C' \subseteq G$ with $e' \in E(C')$, $\vartheta(e')$ does not influence any slack. Therefore, $\vartheta(e')$ can be increased to $\max\{\delta \mid \delta \in \mathcal{T}(e)\}$.

Let us now assume $E(C) \cap \mathrm{E_t} \neq \emptyset$ for some cycle $C \in G$ with $e' \in E(C)$. Let $C' \subseteq G$ be a cycle with maximum ratio delay among those which contain e'. We set $\mathrm{slk}' := \frac{\sum_{e\in E(C')} - \vartheta(e)}{|E(C')\cap \mathrm{E_t}\,|} > \mathrm{S}(\vartheta)$. Now we can increase the delay of e' by

$$\min\{u_e - \vartheta(e'), \mathrm{slk}' - \mathrm{S}(\vartheta)\} > 0,$$

without increasing the ratio delay of any cycle beyond $-\,\mathrm{S}(\vartheta)$. As c_e is a strictly decreasing function we have found a solution of lower cost with the same worst slack. Thus, ϑ cannot be the delay assignment of a time-cost optimum solution.

□

6.3.2 Modifying Delays

In this section the right-sided derivative of a time-cost optimum solution (at, ϑ) with $\mathrm{slk}(\mathrm{at}, \vartheta) < \overline{\mathrm{S}}$

$$B'_{opt}(\mathrm{slk}(\mathrm{at}, \vartheta), 1) := \lim_{h\searrow 0} \frac{B_{opt}(\mathrm{slk}(\mathrm{at}, \vartheta) + h) - B_{opt}(\mathrm{slk}(\mathrm{at}, \vartheta))}{h}$$

is computed. Obviously $B'_{opt}(\mathrm{slk}(\vartheta), 1)$ is induced by underlying arrival time and delay changes. In fact, we will compute $B'_{opt}(\mathrm{slk}(\vartheta), 1)$ implicitly by computing optimum arrival time and delay assignment changes. We will later use these assignment changes to transform the current time-cost optimum solution into a time-cost optimum solution with a higher worst slack.

A worst slack improvement can only be achieved by decreasing the total delay on each maximum ratio delay cycle, due to Lemmata 6.4 and 6.6. All remaining edge delays must stay at their current maximum possible value. We therefore can restrict the search for delays that must be modified to the *critical subgraph* which is defined next.

Definition 6.7. *Given a time-cost tradeoff instance* $(G, \mathcal{T}, (c_e)_{e\in \mathrm{E_p}}, \mathrm{E_t})$ *and a solution* (at, ϑ), *we define* $G' \subseteq G$ *by* $V(G') = V(G)$ *and*

$$\begin{aligned} E(G') = \;& \{(v,w) \in \mathrm{E_p} \;\mid \mathrm{at}(v) + \vartheta(v,w) = \mathrm{at}(w)\} \;\dot\cup \\ & \{(v,w) \in \mathrm{E_t} \;\mid \mathrm{at}(v) + \mathrm{slk}(\mathrm{at},\vartheta) = \mathrm{at}(w)\}. \end{aligned}$$

The critical (sub-) graph $G_{crit} \subseteq G'$ *is the union of strongly connected components in* G'. *For shorter notation we set*

$$\begin{aligned} E_{crit} &:= E(G_{crit}), \\ V_{crit} &:= V(G_{crit}), \\ E_u &:= \{e \in E_{crit} \cap \mathrm{E_p} \mid \vartheta(e) = u_e\}, \text{ and} \\ E_l &:= \{e \in E_{crit} \cap \mathrm{E_p} \mid \vartheta(e) = l_e\}. \end{aligned}$$

$E_u \cap E_l$ contains all edges $e \in E_{crit} \setminus \mathrm{E_t}$ with fixed delays ($l_e = u_e$). We will formulate the problem of optimum delay changes as a linear program. As we consider an infinitesimal slack increase and therefore infinitesimal delay changes, we can ignore relaxed delay constraints, and need to consider only tight delay constraints of the type $\vartheta(e) = l_e$ or $\vartheta(e) = u_e$. We provide a linear program formulation for a specific worst slack improvement of value one. But its solution can be scaled to arbitrary positive slack improvements. For a sufficiently small slack improvement all side constraints that were neglected in the linear program will be fulfilled, and a time-cost optimum solution with a higher worst slack will be found.

We define costs for all $e \in E_{crit} \cap \mathrm{E_p}$ by

$$\tilde{c}_e := \begin{cases} -\frac{\partial c_e}{\partial \vartheta}\Big|_{(l_e,u_e)} & \text{if } l_e < u_e \\ 0 & \text{if } l_e = u_e, \end{cases} \tag{6.9}$$

and introduce variables y_e for all $e \in \mathrm{E_p} \cap E_{crit}$, and z_v for all $v \in V_{crit}$. The y_e-variables refer to the delay decrease of edge $e \in \mathrm{E_p} \cap E_{crit}$ per unit worst slack increase. If $y_e > 0$, e will be accelerated. If $y_e < 0$, e will be delayed. The new delay will be given by $\vartheta(e) - \epsilon y_e$ for some appropriate $\epsilon > 0$. The z_v-variables refer to the change in the start time per unit worst slack increase. The new start times will be given by $\mathrm{at}(v) - \epsilon z_v$ for the same $\epsilon > 0$.

We want to minimize the total cost of decreasing delays, while increasing the worst slack by one. This can be formulated as following linear program:

$$\begin{array}{lll}
\min \sum\limits_{e \in E_{crit} \cap \mathrm{E_p}} & \tilde{c}_e y_e & \\
y_e & \geq 0 & \forall e \in E_u, \\
y_e & \leq 0 & \forall e \in E_l, \\
z_t - z_s & \geq 1 & \forall (t,s) \in E_{crit} \cap \mathrm{E_t}, \\
y_e + z_v - z_w & \geq 0 & \forall e = (v,w) \in E_{crit} \cap \mathrm{E_p}\,.
\end{array} \tag{P}$$

We denote the vector of arrival time changes by $z := (z_v)_{v \in V_{crit}}$ and the vector of delay changes by $y := (y_e)_{e \in E_{crit} \cap \mathrm{E_p}}$. An optimum solution (z, y) for the worst slack improvement of 1 can be transformed into an optimum solution for a worst slack improvement of $\epsilon \geq 0$ by multiplying all variables in (P) by ϵ.

Lemma 6.8. *Let* $(G, \mathcal{T}, (c_e)_{e \in \mathrm{E_p}}, \mathrm{E_t})$ *be a time-cost tradeoff instance and* (at, ϑ) *be a time-cost optimum solution with* $\mathrm{slk}(\mathrm{at}, \vartheta) < \overline{\mathrm{S}}$. *Then the linear program (P) has a finite optimum solution with positive value.*

Proof. Due to our assumption—$\mathrm{slk}(\mathrm{at}, \vartheta) < \overline{\mathrm{S}}$—there exist feasible arrival-time- and delay-change variables y', z' that improve the worst slack in G_{crit} by some value $\epsilon > 0$. Then $(\frac{1}{\epsilon} y', \frac{1}{\epsilon} z')$ is a feasible solution of (P).

Now assume (P) has a solution (y, z) with non-positive objective. Then we can scale all values by a sufficiently small multiplier $\epsilon > 0$ such that $\vartheta(e) - y_e \in [l_e, u_e]$ and obtain solution with less or equal cost that is faster by ϵ. This contradicts the time-cost optimality of (at, ϑ).

□

By linear programming duality the dual problem has a finite optimum. To obtain the dual program we introduce dual variables $\mu_e, e \in E_u$, for the first set of inequalities, $\lambda_e, e \in E_l$, for the second set of inequalities, and $f_e, e \in E_{crit}$, for the third and fourth set of inequalities. Then the dual is given by

$$\begin{array}{lll}
\max \sum\limits_{e \in E_{crit} \cap \mathrm{E_t}} f_e & & \\
\sum\limits_{e \in \delta^+_{E_{crit}}(v)} f_e - \sum\limits_{e \in \delta^-_{E_{crit}}(v)} f_e & = 0 & \forall v \in V_{crit} \\
f_e & = \tilde{c}_e & \forall e \in E_{crit} \setminus (E_u \cup E_l), \\
f_e + \mu_e & = \tilde{c}_e & \forall e \in E_u \setminus E_l, \\
f_e - \lambda_e & = \tilde{c}_e & \forall e \in E_l \setminus E_u, \\
f_e + \mu_e - \lambda_e & = \tilde{c}_e & \forall e \in E_u \cap E_l, \\
f_e & \geq 0 & \forall e \in E_{crit}, \\
\mu_e & \geq 0 & \forall e \in E_u, \\
\lambda_e & \geq 0 & \forall e \in E_l.
\end{array}$$

After elimination of the slack variables μ_e and λ_e, and inverting the objective function, the dual program can be written as:

$$\begin{array}{rll}
\min \quad \sum_{e \in E_{crit} \cap \mathrm{E_t}} & -f_e & \\
\sum_{e \in \delta^+_{E_{crit}}(v)} f_e - \sum_{e \in \delta^-_{E_{crit}}(v)} f_e & = 0 & \forall v \in V(G_{crit}) \\
f_e & = \tilde{c}_e & \forall e \in E_{crit} \setminus (E_u \cup E_l), \\
f_e & \leq \tilde{c}_e & \forall e \in E_u \setminus E_l, \\
f_e & \geq \tilde{c}_e & \forall e \in E_l \setminus E_u, \\
f_e & \geq 0 & \forall e \in E_{crit}.
\end{array} \tag{D}$$

The dual program (D) is, in fact, a minimum-cost-flow problem. An optimum flow f can be found by an efficient minimum-cost-flow algorithm, for instance in strongly polynomial time $O(m \log m(m + n \log n))$ as described by Orlin [1993].

6.3.3 Choosing the Step Length

In the previous section we have computed the optimum delay change y per unit worst slack increase, which induces a descent direction of $B'_{opt}(\mathrm{slk}(\vartheta), 1)$ with a minimum cost increase. In the LP-formulation, we have ignored the constraints induced by the feasible delay ranges $[l_e, u_e]$ for all $e \in G_{crit}$, arc-constraints of type (6.1) in $\mathrm{E_p} \setminus E_{crit}$, and slack constraints in $\mathrm{E_t} \setminus E_{crit}$. In this subsection we determine the maximum feasible step length ϵ of the slack improvement based on these constraints. We look for the largest $\epsilon > 0$ for which $(\vartheta(e) - \epsilon \cdot y_e)$, $e \in \mathrm{E_p}$ is the delay assignment of a time-cost optimum solution, where y is extended to all propagation edges $\mathrm{E_p}$ by $y_e := 0$ for all $e \in \mathrm{E_p} \setminus E_{crit}$.

First, the delays have to stay within the allowed intervals: $(\vartheta(e) - \epsilon \cdot y_e) \in [l_e, u_e]$ for all $e \in \mathrm{E_p}$. This defines two upper bounds ϵ_1 and ϵ_2 on ϵ:

$$\epsilon_1 := \min \left\{ \frac{\vartheta(e) - u_e}{y_e} \mid e \in E_{crit} \cap \mathrm{E_p}, y_e < 0 \right\}, \tag{6.10}$$

$$\epsilon_2 := \min \left\{ \frac{\vartheta(e) - l_e}{y_e} \mid e \in E_{crit} \cap \mathrm{E_p}, y_e > 0 \right\}. \tag{6.11}$$

Second, we have to satisfy arc-constraints not only in G_{crit} but in G. Let ϵ_3 be the largest number such that there is a feasible arrival time assignment at' for delays ϑ' defined by $\vartheta'(e) := \vartheta(e) - \epsilon_3 \cdot y_e, e \in \mathrm{E_p}$, which achieves a worst slack of $\mathrm{slk}(\mathrm{at}', \vartheta') = \mathrm{slk}(\mathrm{at}, \vartheta) + \epsilon_3$. Then ϵ_3 is an upper bound on the feasible step length. The next lemma summarizes the three types of bounds.

Lemma 6.9. *The maximum ϵ by which we may increase the worst slack and modify delays according to the subgradient $(y_e)_{e \in E_{crit}}$, while maintaining time-cost-optimality is given by*

$$\epsilon := \min\{\epsilon_1, \epsilon_2, \epsilon_3\}, \tag{6.12}$$

where

$$\epsilon_1 := \min\left\{\frac{\vartheta(e)-u_e}{y_e} \mid e \in E_{crit} \setminus \mathrm{E_t}, y_e < 0\right\}, \tag{6.13}$$

$$\epsilon_2 := \min\left\{\frac{\vartheta(e)-l_e}{y_e} \mid e \in E_{crit} \setminus \mathrm{E_t}, y_e > 0\right\}, \tag{6.14}$$

$$\epsilon_3 := \max\{\epsilon \mid \exists \textit{ feasible } \mathrm{at}' \textit{ with } \mathrm{slk}(\mathrm{at}', \vartheta - \epsilon y) = \mathrm{slk}(\mathrm{at}, \vartheta) + \epsilon\} \tag{6.15}$$

It turns out that ϵ_3 can again be computed by a minimum ratio cycle algorithm. We define variable delays with argument ϵ and slopes $p : E(G) \to \mathbb{R}$ by:

$$p(e) = \quad y_e \qquad \forall e \in E_{crit} \setminus \mathrm{E_t} \tag{6.16}$$

$$p(e) = \quad 0 \qquad \forall e \in E(G) \setminus (E_{crit} \cup \mathrm{E_t}) \tag{6.17}$$

$$p(e) = -1 \qquad \forall e \in \mathrm{E_t} \tag{6.18}$$

And fixed delays $d : E(G) \to \mathbb{R}$ by:

$$d(e) = \vartheta(e) \qquad \forall e \in E(G) \setminus \mathrm{E_t} \tag{6.19}$$

$$d(e) = \mathrm{slk}(\mathrm{at}, \vartheta) \qquad \forall e \in \mathrm{E_t} \tag{6.20}$$

Recall that we assumed fixed delays of $\vartheta(e) = 0$ for test edges $e \in \mathrm{E_t}$, and note that we want to guarantee at least the worst slack $\mathrm{slk}(\mathrm{at}, \vartheta)$ of the current solution (at, ϑ). Now ϵ_3 is the value of following linear program:

$$\begin{array}{rl} \max & \epsilon \\ \mathrm{at}'(v) + d(v,w) - \epsilon \cdot p(v,w) & \leq \mathrm{at}'(w) \quad \forall (v,w) \in E(G). \end{array} \tag{6.21}$$

This linear program is equivalent to a minimum ratio cycle problem:

Lemma 6.10. *The largest ϵ that fulfills (6.21) equals the value of a minimum ratio cycle:*

$$\epsilon_3 = \min\left\{\frac{\sum_{e\in E(C)} -d(e)}{\sum_{e\in E(C)} p(e)} \;\middle|\; C \subseteq G \textit{ cycle }, \sum_{e\in E(C)} p(e) < 0\right\}.$$

Assuming $p(e) \in \mathbb{Z}$ for all $e \in E(G)$, it can be computed in pseudo-polynomial time $O(y^{max}(nm + n^2 \log n))$, where $y^{max} := \max\{|p(e) - p(e')|; e, e' \in E(G)\}$ and, or in strongly polynomial time

$$O(n^3 \log n + \min\{nm, n^3\} \cdot \log^2 n \log\log n + nm \log m).$$

If $\sum_{e\in E(C)} p(e) \geq 0$ for all cycles $C \in G$, ϵ will be unbounded.

Proof. Equation (6.21) holds if and only if

$$\sum_{e\in E(C)} d(e) \leq \epsilon \cdot \sum_{e\in E(C)} p(e) \text{ for all cycles } C \subseteq G. \tag{6.22}$$

For cycles C with $\sum_{e\in E(C)} p(e) \geq 0$, this is true for all $\epsilon \geq 0$, as the current solution (at, ϑ) is feasible. Therefore, (6.22) can be replaced by

$$\epsilon \leq \frac{\sum_{e\in E(C)} d(e)}{\sum_{e\in E(C)} p(e)} \text{ for all cycles } C \subseteq G, \text{ with } \sum_{e\in E(C)} p(e) < 0. \tag{6.23}$$

Note, $\sum_{e\in E(C)} p(e) < 0$ can only occur on cycles $C \subset G$, $C \not\subset G_{crit}$, because no cycle in G_{crit} is delayed. Therefore, also the numerator $\sum_{e\in E(C)} d(e)$ must be negative, and the fraction is positive.

Given a feasible lower bound of $\epsilon \geq 0$ on the final value, this minimum ratio cycle problem is solvable with the algorithms and running times used in Theorem 5.5.

□

In the next subsection we will see that we always can obtain a bounded integral solution for (P) and provide $\tilde{p} = p$.

The overall algorithm to solve the time-cost tradeoff problem is summarized in Algorithm 13. It considers a slack target $\mathrm{S}_\mathrm{l}^{\mathrm{tgt}} \in \mathbb{R}$, and computes the time-cost tradeoff curve on the interval $[\max\{\underline{\mathrm{S}}, \mathrm{S}_\mathrm{l}^{\mathrm{tgt}}\}, \min\{\overline{\mathrm{S}}, \mathrm{S}_\mathrm{l}^{\mathrm{tgt}}\}]$.

Algorithm 13 Time-Cost Tradeoff Curve Computation

Input: A linear time-cost tradeoff instance $(G, \mathcal{T}, (c_e)_{e\in \mathrm{E_p}}, \mathrm{E_t})$,
a slack target $\mathrm{S}_\mathrm{l}^{\mathrm{tgt}}$.
Output: Support points $\mathrm{S}_0, \mathrm{S}_1, \ldots, \mathrm{S}_K$ and values $\mathrm{B}_0, \mathrm{B}_1, \ldots, \mathrm{B}_K$ of B_{opt}

```
 1: Compute $\overline{\mathrm{S}}$;
 2: Set all delays $\vartheta(e)$ to the lowest cost value;
 3: Compute time-cost optimum solution $(\mathrm{at}, \vartheta)$;
 4: $i = 0$; $\mathrm{S}_0 = \mathrm{slk}(\mathrm{at}, \vartheta)$;$\mathrm{B}_0 = \mathrm{c}(\vartheta)$;
 5: while $\mathrm{S}(\vartheta) < \min\{\overline{\mathrm{S}}, \mathrm{S}_\mathrm{l}^{\mathrm{tgt}}\}$ do
 6:     Min-Cost-Flow in $G_{crit} \Longrightarrow (y_e)_{(e\in E_{crit})}$;
 7:     $\epsilon_1 := \min\{\frac{\vartheta(e)-u_e}{y_e} \mid e \in E_{crit} \setminus \mathrm{E_t}, y_e < 0\}$;
 8:     $\epsilon_2 := \min\{\frac{\vartheta(e)-u_e}{y_e} \mid e \in E_{crit} \setminus \mathrm{E_t}, y_e < 0\}$;
 9:     Minimum-Ratio-Cycle in $G \Longrightarrow \epsilon_3$, $(z_v)_{(v\in V)}$;
10:     $\epsilon := \min\{\epsilon_1, \epsilon_2, \epsilon_3\}$;
11:     Modify $(\mathrm{at}, \vartheta)$ according to $\epsilon \cdot y$ and $\epsilon \cdot z$;
12:     $i := i + 1$; $\mathrm{S}_i = \mathrm{slk}(\mathrm{at}, \vartheta)$;$\mathrm{B}_i = \mathrm{c}(\vartheta)$;
13: end while
```

6.3.4 Bounding the Running Time

The running time of a while-loop iteration (lines 6–12) in Algorithm 13 is dominated by the minimum cost flow computation (line 6) and minimum ratio cycle computation (line 9). For an analysis of the overall algorithm the maximum number of such iterations must be determined. The problem is that ϵ may converge towards 0 and the algorithm may get stuck.

First, it is shown that the LP-solutions in line 6 are integral and all variables can be bounded by $|V(G)|$. Then, the recurrence of a solution vector y is suppressed. Together this yields the termination of the algorithm.

Lemma 6.11. *For integral $\tilde{c}_e \in \mathbb{Z}, e \in E_p$, the linear program (P) has an optimum solution (y,z) with $|y_e|, z_v \in \{0,\dots,|V|-1\}$ for all $e \in \mathrm{E_p}, v \in V$.*

Proof. Hoffman and Kruskal [1956] proved that minimum cost flow problems have integral optimum solutions for integral capacities, and the optimum solution of the dual problem is integral for integral costs (see also Korte and Vygen [2008], Section 5.4 on totally unimodular matrices). Here, in (P) and (D) capacities correspond to $\tilde{c}$ and costs are zero-one vectors. Therefore, both (P) and (D) have integral optimum solutions for $\tilde{c}_e \in \mathbb{Z}, e \in E_p$.

We show how to transform an optimum solution (z,y) of (P) into a solution $(\hat{z},\hat{y})$ in which the maximum difference of arrival time changes is limited by the number of vertices in V_{crit}:

$$\max\{z_v \mid v \in V_{crit}\} - \min\{z_v \mid v \in V_{crit}\} < |V_{crit}|. \tag{6.24}$$

After shifting all z-variables by the same value $(-\min\{z_v \mid v \in V_{crit}\})$ there is always a solution with $\min\{z_v \mid v \in V_{crit}\} = 0$ and $\max\{z_v \mid v \in V_{crit}\} < |V_{crit}|$. Given the limits of the z-variables, the limits of the y-variables follow immediately from the fourth set of inequalities in (P) and the objective to minimize costs.

Assume (6.24) to be violated, and let the vertices $v \in V_{crit}$ be sorted by increasing value z_v: $z_{v_1} \le z_{v_2} \le \dots \le z_{v_{|V_{crit}|}}$. Then there must be two subsequent vertices $v_k, v_{k+1} \in V_{crit}$ whose z-difference is at least two: $z_{v_{k+1}} - z_{v_k} \ge 2$.

We split the set V_{crit} into the two sets $V_{crit} = V' \dot{\cup} V''$ (see Figure 6.2) with

$$V' := \{v \in V_{crit} \mid z_v \ge z_{v_{k+1}}\} \text{ and}$$
$$V'' := \{v \in V_{crit} \mid z_v \le z_{v_k}\}.$$

We claim that we can increment all values $z_v, v \in V''$ by one, while adapting some y-variables, and obtain a feasible solution of (P) with equal costs. For this purpose we categorize the edges in the cut $\delta_{G_{crit}}(V')$, which are indicated by the red and blue arcs in Figure 6.2.

1. Red test edges: $\{(v',v'') \in \delta^+_{G_{crit}}(V') \cap \mathrm{E_t}\}$. Here $z_{v'} - z_{v''} \ge 2$ and the corresponding constraint in (P) will not be violated after increasing $z_{v''}$ by one.

2. Tight red propagation edges:
$\mathrm{E_p^{red}} := \{(v,w) \in \delta^+_{G_{crit}}(V') \cap \mathrm{E_p}; y_{(v,w)} + z_v - z_w = 0\}$.
Here $y_e \le -2$. Therefore, the first two inequalities in (P) allow to increase y_e by at least 2, and to decrease y_e arbitrarily $(e \in \mathrm{E_p^{red}})$.

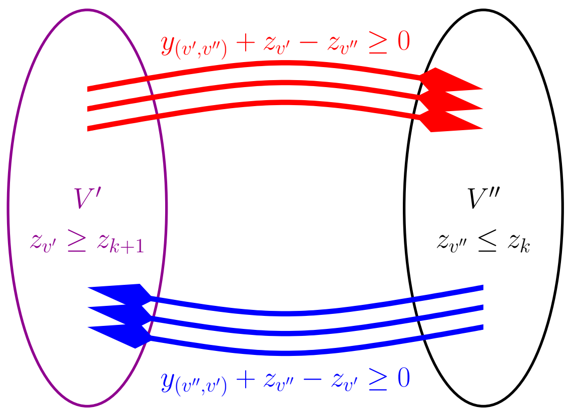

$y_{(v',v'')} + z_{v'} - z_{v''} \geq 0$
V'
$z_{v'} \geq z_{k+1}$
V''
$z_{v''} \leq z_k$
$y_{(v'',v')} + z_{v''} - z_{v'} \geq 0$

Figure 6.2: Limiting the solution size of (P).

3. Relaxed red propagation edges: $\{(v,w) \in \delta^+_{G_{crit}}(V') \cap \mathrm{E_p}; y_{(v,w)} + z_v - z_w > 0\}$. Here y_e must be bounded from below. Therefore, $y_e = 0$ and $y_{(v,w)} + z_v - z_w \geq 2$ for all $e \in \{(v,w) \in \delta^+_{G_{crit}}(V') \cap \mathrm{E_p}; y_{(v,w)} + z_v - z_w > 0\}$. These edges stay valid when increasing $z_w, w \in V''$ by up to 2. We do not consider these edges during the increase step.
4. Blue propagation edges: $\mathrm{E_p^{blue}} := \{(v'',v') \in \delta^-_{G_{crit}}(V') \cap \mathrm{E_p}\}$. Here $y_{(v'',v')} \geq z'_v - z''_v \geq 2$, and by cost minimization $y_{(v'',v')} = z'_v - z''_v$. Therefore, the first two inequalities in (P) allow to decrease $y_{(v'',v')}$ by at least 2, and to increase $y_{(v'',v')}$ arbitrarily.
5. There can be no blue test edges:
$\{(v,w) \in \delta^-_{G_{crit}}(V') \cap \mathrm{E_t}\} = \emptyset$ as $z_v - z_w \leq -2 < 0$ for all $(v,w) \in \delta^-_{G_{crit}}(V') \cap \mathrm{E_t}$.

We claim

$$\sum_{e \in \mathrm{E_p^{red}}} \tilde{c}_e = \sum_{e \in \mathrm{E_p^{blue}}} \tilde{c}_e.$$

Assume $\sum_{e \in \mathrm{E_p^{red}}} \tilde{c}_e > \sum_{e \in \mathrm{E_p^{blue}}} \tilde{c}_e$. Then a feasible solution with less cost can be found by decreasing z_w and $y_{(v,w)}$ by 1 for all $w \in V''$ and all $(v,w) \in \mathrm{E_p^{red}}$, and by increasing $y_{(v,w)}$ by 1 for all $(v,w) \in \mathrm{E_p^{blue}}$.

Analogously if $\sum_{e \in \mathrm{E_p^{red}}} \tilde{c}_e < \sum_{e \in \mathrm{E_p^{blue}}} \tilde{c}_e$ a feasible solution with less cost can be found by increasing z_w and $y_{(v,w)}$ by one for all $w \in V''$ and all $(v,w) \in \mathrm{E_p^{red}}$, and by decreasing $y_{(v,w)}$ by one for all $(v,w) \in \mathrm{E_p^{blue}}$.

Therefore, we can obtain a feasible solution of the same cost by increasing z_w and y_e by one for all $w \in V''_{crit}$ and all $e \in \mathrm{E_p^{red}}$, and by decreasing y_e by 1 for all $e \in \mathrm{E_p^{blue}}$.

□

Corollary 6.12. *There is an optimum solution for (P) in which $|y_e| < |V_{crit}|$ for all $e \in E_{crit} \cap \mathrm{E_p}$. This solution can be found by a minimum cost flow algorithm with the same running time bound as (P).*

Proof. Lemma 6.11 shows the existence of a bounded optimum solution. When adding constraints $-|V_{crit}|+1 \leq y_e \leq |V_{crit}|-1$ for all $e \in E_{crit} \cap \mathrm{E_p}$ to (P), we obtain a new linear program (P'):

$$\begin{array}{lll}
\min \sum\limits_{e \in E_{crit} \setminus \mathrm{E_t}} & \tilde{c}_e y_e & \\
y_e & \geq 0 & \forall e \in E_u, \\
y_e & \leq 0 & \forall e \in E_l, \\
y_e & \leq |V_{crit}|-1 & \forall e \in E_{crit} \cap \mathrm{E_p}, \\
y_e & \geq -|V_{crit}|+1 & \forall e \in E_{crit} \cap \mathrm{E_p}, \\
z_t - z_s & \geq 1 & \forall (t,s) \in E_{crit} \cap \mathrm{E_t}, \\
y_e + z_v - z_w & \geq 0 & \forall (v,w) \in E_{crit} \cap \mathrm{E_p}\,.
\end{array} \tag{P'}$$

This program is again the dual of a minimum cost flow problem.

□

Note that the bounds on the variable sizes are independent from the costs. Thus, we can scale fractional costs to integers and obtain a bounded integral solution. As we apply strongly polynomial algorithms for the minimum cost flow problem and the representation sizes of the scaled costs are bounded linearly in the number of edges times the input sizes of the costs, this does not affect the running times.

The recurrence of a solution vector y can be prevented by a technique known from preventing cycling in the simplex method for linear programming (see Charnes [1952]): a slight cost perturbation. This cost perturbation will create unique costs for every solution vector y. As the time-cost tradeoff curve is convex this prevents a solution vector y of (P) to recur. As the entries in $|y|$ are be bounded by $|V(G)|$, the number of different y-vectors is limited by $(2|(VG)|-1)^{|\mathrm{E_p}|}$.

The cost perturbation will only be applied when solving (P'). Assume that the costs $\tilde{c}_e, e \in \mathrm{E_p}$ are already scaled to integers. Let the edges be arranged in arbitrary but fixed order $e_1, e_2, \ldots, e_{|E(G)|}$.

Lemma 6.13. *Number the propagation edges $E_p = \{e_1, e_2, \ldots, e_{|E(G)|}\}$. For $K \geq 2|V(G)|$ any optimum solution y of (P') with respect to the perturbed costs $\hat{c}_{e_i} := \tilde{c}_{e_i} + K^{-i}, e_i \in \mathrm{E_p}$, is also optimum with respect to the original costs $\tilde{c}_e, e \in \mathrm{E_p}$. Furthermore, different integral solutions y and y' have different values with respect to $\hat{c}$:*

$$\sum_{e \in \mathrm{E_p}} \hat{c}_e y_e \neq \sum_{e \in \mathrm{E_p}} \hat{c}_e y'_e \qquad \textit{for all } y \neq y'. \tag{6.25}$$

Proof. Let y, y' be two feasible integral solutions of (P'). If they have different total cost with respect to $\tilde{c}$, this difference will be integral: $\sum_{e \in \mathrm{E_p}} \tilde{c}_e (y_e - y'_e) \in \mathbb{Z}$. Furthermore, two different solutions $y \neq y'$ have different perturbation cost and the absolute perturbation cost difference is less than one:

$$
\begin{aligned}
0 < \left| \sum_{e_i \in \mathrm{E_p}} K^{-i}(y_{e_i} - y'_{e_i}) \right| &\leq \sum_{e_i \in \mathrm{E_p}} K^{-i}(2|V(G)| - 1) \\
&< \frac{2|V(G)| - 1}{K - 1} \\
&\leq 1 \qquad\qquad \text{for } y \neq y'.
\end{aligned}
$$

It follows immediately that two integral solutions $y \neq y'$ have different objective values with respect to $\hat{c}$. Assume, there is an optimum solution y with respect to $\tilde{c}$ that is not optimum with respect to $\hat{c}$ and let y' be optimum with respect to $\hat{c}$. Then

$$
\begin{aligned}
\sum_{e_i \in \mathrm{E_p}} \hat{c}_{e_i}(y_{e_i} - y'_{e_i}) &= \sum_{e_i \in \mathrm{E_p}} \tilde{c}_{e_i}(y_{e_i} - y'_{e_i}) + \sum_{e_i \in \mathrm{E_p}} K^{-i}(y_{e_i} - y'_{e_i}) && (6.26) \\
&< -1 + 1 = 0. && (6.27)
\end{aligned}
$$

This contradicts the optimality of y' with respect to $\hat{c}$ and completes the proof.

□

After scaling the perturbated cost of each propagation edge to an integer by multiplication with $K^{|\mathrm{E_p}|}$, the size of its representation increases from $\log \tilde{c}_{e_i}$ to $O(\log \tilde{c}_{e_i} + |\mathrm{E_p}| \log K)$. Thus, the perturbation increases the total representation size of the costs by $O(|\mathrm{E_p}|^2 \log |V(G)|)$. Again, this will not influence the worst case running time, when applying strongly polynomial algorithms for the minimum cost flow and the minimum ratio cycle computations.

Theorem 6.14. *Given a linear time-cost tradeoff instance $(G, \mathcal{T}, (c_e)_{e \in \mathrm{E_p}}, \mathrm{E_t})$, the time-cost tradeoff curve B_{opt} can be computed in $O(T(2n-1)^p)$ time, where $n = |V(G)|, p = |\mathrm{E_p}|$, and T is the running time of a minimum cost flow plus a minimum ratio cycle computation. The currently best known bound for T is $O(\min\{m \log m(m + n \log n), n^2 m + n^3 \log n\})$ with $m = |E(G)|$.*

Proof. The theorem follows from the previous considerations including scaling and perturbating the slopes of the cost functions. Each y-variable can take at most $2n - 1$ different values. Thus, there are at most $(2n-1)^p$ different delay modification vectors. The running time bound for a minimum cost flow computation is $O(m \log m(m + n \log n))$ (Orlin [1993]). By Lemma 6.11 the absolute values of the entries in the (integral) vector y are bounded by $|V| - 1$. Therefore, a minimum ratio cycle computation can be done in $O(n^2 m + n^3 \log n)$ according to Theorem 5.5 and Remark 5.8.

□

6.3.5 Piecewise Linear Convex Cost Functions

Algorithm 13 can be extended to handle piecewise linear convex costs instead of linear costs. Let $(G, \mathcal{T}, (c_e)_{e \in \mathrm{E_p}}, \mathrm{E_t})$ be a time-cost tradeoff instance, and let

$l_e = p_e^0 < p_e^1 < \cdots < p_e^{k_e} = u_e$ be the support points of the piecewise linear cost convex function c_e of edge e. See Figure 6.3 for an example.

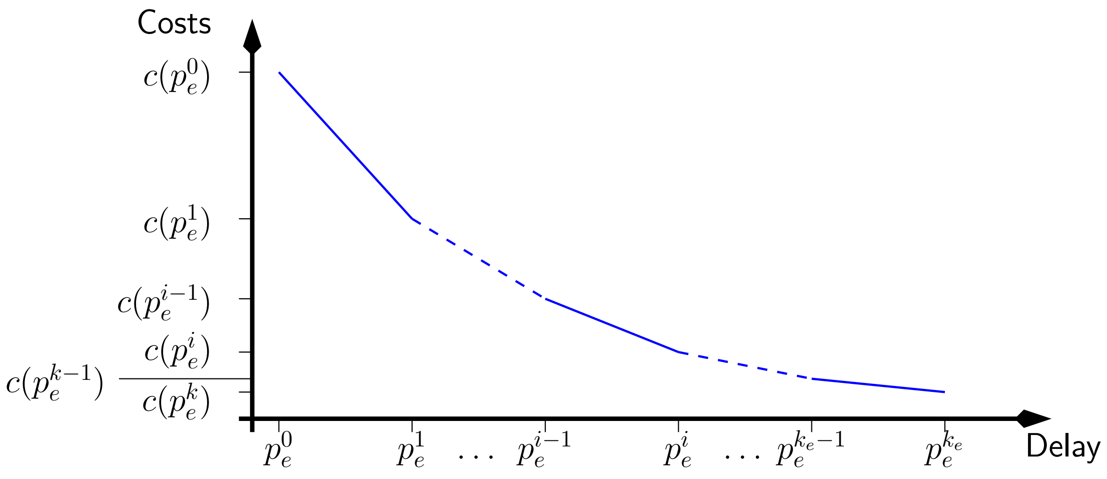


Figure 6.3: A piecewise linear convex cost function

Now define a new instance $(G', \mathcal{T}^{G'}, (c_{e'})_{e' \in E(G') \setminus \mathrm{E_t}}, \mathrm{E_t})$ with linear cost functions. The graph G' is constructed from G by replacing each arc $e \in \mathrm{E_p}$ by a path $P_e \subset G'$ with $(k_e + 1)$ edges $e'^0_e, e'^1_e, \dots, e'^{k_e}_e$. The cost function of an edge $e'^i_e \in E(G')$ is denoted by c_e^i.

The edge e'^0_e represents the minimum possible delay, $\mathcal{T}^{G'}(e'^0_e) := \{p_e^0\}$, with minimum possible costs $c_e^0(p_e^0) = c_e(u_e)$. The other edges e'^i_e, $i \in \{0, \dots, k_e - 1\}$, represent a segment of the piecewise linear function c_e: the delay range is $\mathcal{T}(e'^i_e) := [0, p_e^i - p_e^{i-1}]$, with linear decreasing costs $c_e^i : \mathcal{T}(e_e^i) \to [0, c_e(p_e^{i-1}) - c_e(p_e^i)]$.

Due to the convexity of c_e the (negative) slopes of the linear pieces are increasing: $\frac{\partial c_e^1}{\partial \vartheta} < \frac{\partial c_e^2}{\partial \vartheta} < \cdots < \frac{\partial c_e^{k_e}}{\partial \vartheta}$. Due to this ordering, the minimum cost delay assignment of the path P_e with total delay equal to a given delay $\vartheta(e)$ on the original edge $e \in \mathrm{E_p}$ has following property. There exists an index $i^\star \in \{1, \dots, k_e\}$ such that

$$\begin{aligned} \vartheta(e_e^i) &= 0 && \text{for all } i > i^\star, \\ \vartheta(e_e^i) &= p_e^i - p_e^{i-1} && \text{for all } i < i^\star, \text{ and} \\ \vartheta(e_e^{i^\star}) &= \vartheta(e) - p_e^{i^\star - 1} && (\text{implying } \vartheta(e_e^{i^\star}) \in [0, p_e^{i^\star} - p_e^{i^\star - 1}]). \end{aligned} \tag{6.28}$$

Therefore, there is a one-to-one correspondence between time-cost optimum solutions in $(G, \mathcal{T}, (c_e)_{e \in \mathrm{E_p}}, \mathrm{E_t})$ and $(G', \mathcal{T}^{G'}, (c_{e'})_{e' \in E(G') \setminus \mathrm{E_t}}, \mathrm{E_t})$. The choice of $i^\star$ is not unique: if $\vartheta(e'^{i^\star}_e) = 0$ (or $\vartheta(e'^{i^\star}_e) = p_e^{i^\star} - p_e^{i^\star - 1}$ and $i^\star < k_e$,), $i^\star$ could be decremented (incremented) by one.

The extended instance $(G', \mathcal{T}^{G'}, (c'_{e'})_{e' \in E(G') \setminus \mathrm{E_t}}, \mathrm{E_t})$ demonstrates that the piecewise linear convex problem can be transformed into an equivalent problem with linear costs. Nevertheless the implementation, especially the solution of (D), can be found without explicitly using the extended graph. During one iteration of the while-loop in Algorithm 13, at most one edge delay on the path $P_e \subset G'$ representing $e \in \mathrm{E_p}$ will be modified, due to the convexity of c_e. To compute these delay changes we have to solve the dual minimum cost flow problem (D). The flow-constraints induced by each path P_e can be represented equivalently by constraints on a single edge:

In the case $\vartheta(e) \in (p_e^{i-1}, p_e^i)$ for some $i \in \{1, \dots, k_e\}$, all delays on edges $e'^j_e \in E(P_e)$ with $j \neq i$, will be unchanged and can be neglected by contraction. Otherwise, $\vartheta(e) = p_e^i$ for some $i \in \{0, \dots, k_e\}$, either $\vartheta(e'^{i+1}_e)$ will be increased or $\vartheta(e'^i_e)$ will be decreased in the current iteration, while all other edge-delays in P_e remain unchanged. As the delay reduction variable $y_{e'^{i+1}_e}$ is not bounded from below, and $y_{e'^i_e}$ is not bounded from above in (P), we have $e'^i_e \in E_u \setminus E_l$ and $e'^{i+1}_e \in E_l \setminus E_u$. In the dual problem this results in flow-constraints $f_{e'^i_e} \leq \tilde{c}^i_e$ and $f_{e'^{i+1}_e} \geq \tilde{c}^1_e$. These two constraints on two consecutive edges in P_e can equivalently be considered as two constraints on a single edge.

It follows that (P) can solved by a minimum cost flow computation in the original graph G. The computation of ϵ_1, ϵ_2 must be adapted such that no delay exceeds a support interval, and the computation of ϵ_3 is unchanged, because it does not depend on the cost functions. However, this implementation detail does not affect the worst case running time. This is given by the worst case running time for the extended instance $(G', \mathcal{T}^{G'}, (c'_{e'})_{e' \in E(G') \setminus \mathrm{E_t}}, \mathrm{E_t})$. From Theorem 6.14 we derive the following corollary.

Corollary 6.15. *Given a time-cost tradeoff instance $(G, \mathcal{T}, (c_e)_{e \in \mathrm{E_p}}, \mathrm{E_t})$ with piecewise linear convex costs, the time-cost tradeoff curve B_{opt} can be computed in $O(T(2n-1)^K)$ time, with $n = |V(G)|$, $m = E(G)$ and $K := \sum_{e \in \mathrm{E_p}} k_e$, and $T = O(\min\{m \log m (m + n \log n), n^2 m + n^3 \log n\})$ is the running time bound for a minimum cost flow plus a minimum ratio cycle computation.*

□

6.3.6 Optimizing Weighted Slacks

In some applications the individual test edges might be of different importance, for example if we want to minimize the reference cycle time as in Section 5.2.3. Let us assume that a non-negative weight is assigned to each test edge by $\mathrm{w} : \mathrm{E_t} \to \mathbb{R}_+$, and the worst weighted slack is to be maximized. Here the worst weighted slack $\mathrm{slk^w}(\mathrm{at}, \vartheta)$ for a feasible pair (at, ϑ) is defined as

$$\mathrm{slk^w}(\mathrm{at}, \vartheta) := \min_{e \in \mathrm{E_t}} \frac{\mathrm{slk}(e)}{\mathrm{w}(e)}. \tag{6.29}$$

The Weighted Budget Problem, the Weighted Deadline Problem, and the Weighted Time-Cost Tradeoff Problem arise from the non-weighted definitions on pages 128 and 129, by replacing every occurrence of slk by $\mathrm{slk^w}$.

In presence of weights, Lemma 6.4 translates to following lemma.

Lemma 6.16. *Let $(G, \mathcal{T}, (c_e)_{e\in \mathrm{E_p}}, \mathrm{E_t}, \mathrm{w})$ be a weighted time-cost tradeoff instance, and let ϑ be a delay assignment. Then the highest achievable worst weighted slack for ϑ is given by a minimum ratio negative-delay cycle:*

$$\min\left\{\frac{\sum_{e\in E(C)} -\vartheta(e)}{\sum_{e\in E(C)\cap \mathrm{E_t}} \mathrm{w}(e)} \mid C \subseteq G,\ \textit{cycle}, E(C)\cap \mathrm{E_t} \neq \emptyset\right\}.$$

By Theorem 5.5 it can be computed in strongly polynomial time. The delay modification linear program (P) transforms to

$$\begin{array}{lll} \min \sum\limits_{e\in E_{crit}\cap \mathrm{E_p}} & \tilde{c}_e y_e & \\ y_e & \geq 0 & \forall e \in E_u, \\ y_e & \leq 0 & \forall e \in E_l, \\ z_t - z_s & \geq \mathrm{w}(t,s) & \forall (t,s) \in E_{crit}\cap \mathrm{E_t}, \\ y_e + z_v - z_w & \geq 0 & \forall e = (v,w) \in E_{crit}\cap \mathrm{E_p}\,. \end{array} \tag{6.30}$$

Furthermore, the minimum ratio cycle computations in lines 3 and 9 of Algorithm 13 need to be modified. Equation (6.18) needs to be changed in line 9. The variable delay of -1 on test-edges needs to be replaced by

$$p(e) = -\,\mathrm{w}(e) \quad \forall e \in \mathrm{E_t}\,. \tag{6.31}$$

The worst case running time of Algorithm 13 can increase because the bounds on the optimum solutions y must be increased under weights.

Furthermore, the computations of the minimum ratio cycles may become more costly. In Theorem 6.14 the bounds on the y-variables were used to yield a strongly polynomial running time for the parametric shortest path algorithm. For arbitrary weights w, a strongly polynomial running time can only be guaranteed using Megiddos minimum ratio cycle algorithm as in Lemma 6.10.

Theorem 6.17. *Given a time-cost tradeoff instance $(G, \mathcal{T}, (c_e)_{e\in \mathrm{E_p}}, \mathrm{E_t})$ together with weights $\mathrm{w} : E_t \to \mathbb{N}$, the weighted time-cost tradeoff curve $\mathrm{B}^{\mathrm{w}}_{opt}$ can be computed in $O(T(2n\,\mathrm{w_{max}} - 1)^p)$ time, where $n = |V(G)|, p = |\,\mathrm{E_p}\,|, w_{\max} = \max\{\mathrm{w}(e) \mid e \in \mathrm{E_t}\}$, and T is the running time of a minimum cost flow computation plus a minimum ratio cycle computation.*

Proof. The proof can be carried out as the proof of Theorem 6.14. The main difference is that the values $|y_e|, e \in \mathrm{E_p}$, are bounded by $|y_e| < |V_{crit}|\,\mathrm{w_{max}}$ (instead of $|y_e| < |V_{crit}|$). The proof of Lemma 6.11 needs to be adapted by replacing $|V_{crit}|$ by $|V_{crit}|\,\mathrm{w_{max}}$ (unless it specifies the index of the vertex with maximum z-value $z_{v_{|V_{crit}|}}$), and replacing 2 by $(\mathrm{w_{max}}+1)$ (unless it acts of an subtractor of a vertex index).

$\square$

In the weighted case the parametric shortest path algorithm provides a worst case running time of $O(\mathrm{w_{max}}(nm + n^2 \log n))$ for computing a minimum ratio cycle, which is not strongly polynomial in contrast to the non-weighted case. Valid strongly polynomial running time bounds for the minimum ratio cycle computation can be found in Theorem 5.5 on page 88.

6.3.7 Optimizing the Slack Distribution

In many practical applications we are given both a slack target $\mathrm{S}_\mathrm{l}^{\mathrm{tgt}} \in \mathbb{R}$ and a budget $\mathrm{B} \in \mathbb{R}$, and we are looking for a time-cost optimum solution (at, ϑ) that either reaches the slack target $\mathrm{S}_\mathrm{l}^{\mathrm{tgt}}$ or consumes the complete budget B. This problem can be solved with Algorithm 13, when stopping the while-loop as soon as the slack target is reached or the budget is consumed.

However, in most VLSI-scenarios this point is unreachable, because $\mathrm{slk}(\mathrm{at}, \vartheta) < \mathrm{S}_\mathrm{l}^{\mathrm{tgt}}$, and $c(\mathrm{at}, \vartheta) < \mathrm{B}$ for a time-cost optimum solution with $\mathrm{slk}(\mathrm{at}, \vartheta) = \overline{\mathrm{S}}$. If this is the case other potentially more resource or running time consuming optimization routines, such as logic restructuring or replacement, have to be applied afterwards. The workload for more costly routines is to be kept small. Therefore, not only the maximization of the worst slack is of interest, but the computation of a minimum cost vector $(\mathrm{slk}(e))_{e \in \mathrm{E_t}}$ that is leximin maximal with respect to $\mathrm{S}_\mathrm{l}^{\mathrm{tgt}}$. However, it is not clear how to optimize the slack distribution on uncritical parts at minimum cost, as the example in Figure 6.4 shows.

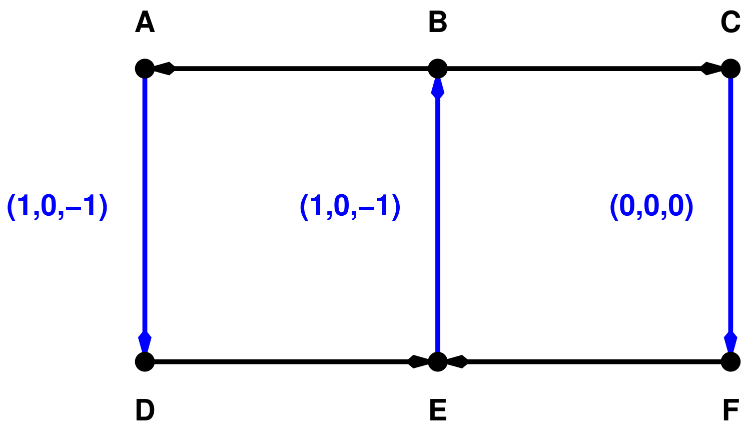


Figure 6.4: An example showing that a time-cost tradeoff optimum delay assignment might not yield a distribution-cost optimum solution.

Assume vertical (blue) edges to be propagation edges and horizontal (black) edges to be test edges. The labels on the propagation edges specify $(u_e, l_e, \tilde{c}_e)$. In the beginning $A \rightarrow D \rightarrow E \rightarrow B \rightarrow A$ is the worst slack cycle with a slack of -1. This slack can be improved at minimum cost by speeding up (A, D) to $\vartheta(A, D) = 0$. Afterwards all slacks would be -0.5. However, when choosing (E, B) instead of (A, D) the slack of the other simple cycle $B \rightarrow C \rightarrow F \rightarrow E \rightarrow B$ would have a slack of 0. Therefore, the choice of (B, E) would lead to a better slack distribution at equal cost.

Nevertheless, Algorithm 13 can be modified in the following way. When the worst slack is reached, the second worst slack can be maximized while keeping the worst slack unchanged. Just like the slack balance algorithm (Algorithm 7 in Section 5.2), this can be done by fixing all slacks on those worst slack cycles whose edge delays are all at their minimum possible value. All delays on these cycles are fixed and all test edges are transferred from $\mathrm{E_t}$ to $\mathrm{E_p}$ with a fixed delay of $\mathrm{slk}(\vartheta)$.

Formally, given a time-cost optimum solution (at, ϑ), the subgraph G^{fix}_{crit} of non-improvable worst slack cycles is the union of the strongly connected subgraphs of $(V_{crit}, \{e \in E_{crit} | \, \vartheta(e) = l_e\})$. Algorithm 14 describes how the slack distribution can be optimized. The modified time-cost tradeoff instance maintains all worst slacks,

Algorithm 14 Slack Distribution Time-Cost Tradeoff Algorithm

```
Input:  A linear time-cost tradeoff instance $(G, \mathcal{T}, (c_e)_{e\in \mathrm{E_p}}, \mathrm{E_t})$,
        a slack target $\mathrm{S_l^{tgt}}$.
1: Run Algorithm 13 $\Rightarrow (\mathrm{at}, \vartheta)$;
2: while $\mathrm{slk}(\mathrm{at}, \vartheta) < \mathrm{S_l^{tgt}}$ do
3:     Compute $G^{fix}_{crit}$;
4:     $\vartheta(e) \leftarrow \mathrm{slk}(e)$ for all $e \in E(G^{fix}_{crit}) \cap \mathrm{E_t}$;
5:     $\mathrm{E_p} \leftarrow \mathrm{E_p} \cup (E(G^{fix}_{crit}) \cap \mathrm{E_t})$;
6:     $\mathrm{E_t} \leftarrow \mathrm{E_t} \setminus E(G^{fix}_{crit})$;
7:     Run Algorithm 13 $\Rightarrow (\mathrm{at}, \vartheta)$;
8: end while
9: return $(\mathrm{at}, \vartheta)$;
```

and Algorithm 13 can be continued until there is a next worst slack cycle that cannot be improved any more. This approach results in a sequence of solutions that yield an optimum slack distribution on the subgraph of cycles, which are completely assigned to the minimum delays.

Theorem 6.18. *Given a time-cost tradeoff instance $(G, \mathcal{T}, (c_e)_{e\in \mathrm{E_p}}, \mathrm{E_t})$, a slack target $\mathrm{S_l^{tgt}}$, an output (at, ϑ) of Algorithm 14, let G^{fix}_{crit} be defined with respect to (at, ϑ). Then (at, ϑ) is a minimum cost solution such that the vector $(\mathrm{slk}(e))_{(e\in E(G^{fix}_{crit}))} \cap \mathrm{E_t})$ is leximin maximal and $\mathrm{slk}(e) \geq \mathrm{S_l^{tgt}}$ for all $e \in \mathrm{E_t} \setminus E(G^{fix}_{crit})$.*

Proof. By construction the slack distribution in G^{fix}_{crit} is equivalent to a slack distribution computed by Algorithm 7 in G^{fix}_{crit}, when all delays in G^{fix}_{crit} are lowest possible.

□

6.3.8 Notes on the Acyclic Case

In the acyclic case, there is only a single test arc. The minimum cost flow instance reduces to a maximum flow instance with upper and lower capacitances, and the minimum ratio cycle instance reduces to a longest path computation. The worst-case running time of a while-loop iteration is significantly smaller:

The maximum flow problem can be solved in strongly polynomial time $O(nm \log_{(2+m/(n \log n))} n)$ (King et al. [1994]), or in

$O(\min\{m^{1/2}, n^{2/3}m\} m \log(n^2/m) \log C)$ (Goldberg and Rao [1998]), where $n = |V(G)|, m = |E(G)|$, and $C = ||\tilde{c}||_\infty$. A longest path computation in an acyclic graph runs in $O(m)$ time. More details on the acyclic case and its applications to threshold voltage assignment in VLSI-design can be found in Schmedding [2007].

6.3.9 Infeasible Maximum Delays

In the definition of a time-cost tradeoff instance, we required any positive delay cycle to contain at least one test edge. This condition can be relaxed, to a condition where just every cycle $C \subseteq G$, with $\sum_{e \in E(C)} \min_{\delta \in \mathcal{T}(e)} \delta > 0$, must contain at least one test edge. But finding an initial time-cost optimum solution of minimum cost becomes more difficult. However, once it is found, the time-cost tradeoff curve can be computed by Algorithm 13.

First, we compute the minimum cost feasible solution (at, ϑ) that might not be time-cost optimum.

Lemma 6.19. *Given a time-cost tradeoff instance* $(G, \mathcal{T}, (c_e)_{e \in \mathrm{E_p}}, \mathrm{E_t})$, *a feasible solution* (at, ϑ) *of minimum cost can be found in* $O(m \log m(m + n \log n))$ *time.*

Proof. We formulate the problem as a linear program that ignores all slacks. Costs $(\tilde{c}_e)_{(e \in E(G))}$ are defined as in (6.9) by the slopes of the linear cost functions. There are variables y_e for all $e \in \mathrm{E_p}$ and z_v for all $V \in V(G)$, where y_e describes a delay reduction yielding a final delay of $\vartheta(e) = u_e - y_e$, and z_v determines the arrival times $at(v) = z_v$. Now we can formulate the problem as:

$$\begin{aligned} \min \sum_{e \in E_{crit} \cap \mathrm{E_p}} & \quad \tilde{c}_e y_e \\ u_e - l_e \geq \; y_e & \quad \geq 0 \quad \forall e \in \mathrm{E_p}, \\ z_w - z_v + y_e & \quad \geq 0 \quad \forall e = (v, w) \in \mathrm{E_p} \end{aligned} \tag{6.32}$$

This is the dual of a minimum cost flow problem, which again can be solved in $O(m \log m(m + n \log n))$ time (Orlin [1993]).

□

This gives a tight lower bound $\underline{\mathrm{B}} := \sum_{e \in \mathrm{E_p}} c_e(\vartheta(e))$ for $dom(\mathrm{S}^{opt})$, but another delay assignment ϑ' of equal cost might offer a higher worst slack. Therefore, we have to solve the BUDGET PROBLEM with budget $\underline{\mathrm{B}}$. So far no combinatorial algorithm to this problem is known. Thus, we rely on a general purpose LP-solver, such as the polynomial time algorithm by Karmarkar [1984].

6.4 Applications in Chip Design

In this section we model certain optimization techniques in chip design as time-cost tradeoff problems. The general framework generates a set of alternative "local" layouts, for example for a circuit or net. With each layout a delay and a cost is associated. The cost is induced by the underlying resource consumption, for example power or space. The goal is to find, with limited resources, a layout assignment that yields a leximin maximum distribution of slacks with respect to a slack target.

The time-cost tradeoff problem is especially applicable where timing optimization is bounded by limited resources, and where the optimization operations have only a "local" influence on a small set of edges in the timing graph. This characteristic holds for many optimization techniques, especially for threshold voltage assignment, plane assignment, and repeater tree insertion, where a layout change influences the delay through a single circuit or (buffered) net. Furthermore, the effect of an operation is independent from other operations on a path and the delay changes on the path can simply be added.

In contrast, changing the size of a circuit does not only influence the delay through that circuit but also through its predecessor circuits in the opposite direction.

Algorithm 13 on page 137 serves as a heuristic for the discrete problem on non-linear delay models. Starting from a least resource consuming layout it iteratively selects a set of layout changes called *operations* that improves the slack distribution of the design. Whenever a set of time-cost efficient operations is to be found, the problem is linearly relaxed, and the minimum-cost flow as well as the minimum-ratio cycle problem are solved as in a while-loop iteration of Algorithm 13. Guarded by the delay modification y-vector, underlying operations are applied whenever their y-value is non-zero. An iteration is finished by reoptimizing the arrival time assignment.

One feature of Algorithm 13 is that the whole instance graph G need not be created explicitly. The minimum cost flow computation is performed only on G_{crit}. The minimum ratio cycles can be computed by the incremental parametric shortest path algorithm by Albrecht [2006]. The minimum ratio cycle computation would first start constructing its structures on G_{crit} only and the incrementally add the most critical parts of G through callbacks. If $|E(G_{crit})| \ll |E(G)|$ this can yield an significant speed-up even for the Budget Problem, which could be solved by a single minimum cost flow computation in G. In our VLSI-applications G will not be created explicitly.

Another advantage of Algorithm 13 is its iterative nature. The error that is introduced with every linearization can be (rewarded) in subsequent iterations. Furthermore all local layout changes create small delay changes in the physical and logical vicinity of the modified parts too. In practice these functions can be reapproximated after each iteration during the course of the algorithm. This yields an overall stable behavior and ensures that the final result matches with reality. In a single linear programming optimization of the Budget Problem or Deadline Problem the approximation error can get very large if delay-cost functions are estimated only once at the beginning. Especially many electrical violations can make the results useless.

A similar model was proposed in Schietke [1999] for the acyclic case. There, the delay modification vector is computed heuristically by a minimum cut approach, but without enabling simultaneous decreasing and increasing of delays. Thus, it does not guarantee optimality even in the case of linear problems.

6.4.1 Delay Optimization Graph

The delay optimization graph $G^{\mathcal{O}}$, which corresponds to G_{crit}, is defined such that layout changes through a circuit or net can be identified by a single edge. Note that we cannot use the timing graph $G^{\mathcal{T}}$ directly because a layout change usually affects several timing edges in $E^{\mathcal{T}}$. Such a coupling of jobs is not modeled in the time-cost tradeoff problem. Instead of maintaining real edge delays we estimate only the delay changes caused by layout changes.

For every circuit $c \in \mathcal{C}$ there is one vertex for every input pin from $P_{in}(c)$, one vertex for every output pin from $P_{out}(c)$, and two *internal* vertices v_c^{in} and v_c^{out}. There is an edge (v_c^{in}, v_c^{out}) that represents the layout changes of c.

For every net $N \in \mathcal{N}$ there are $|P_{out}(N)|$ edges as in the timing graph. Let $p \in P_{out}(c) \cap P_{in}(N)$ be the source of N, and let v_p be the vertex that represents p. Layout changes of n are represented by the edge (v_c^{out}, v_p). Figure 6.5 shows the model of a two bit full adder and its output nets.

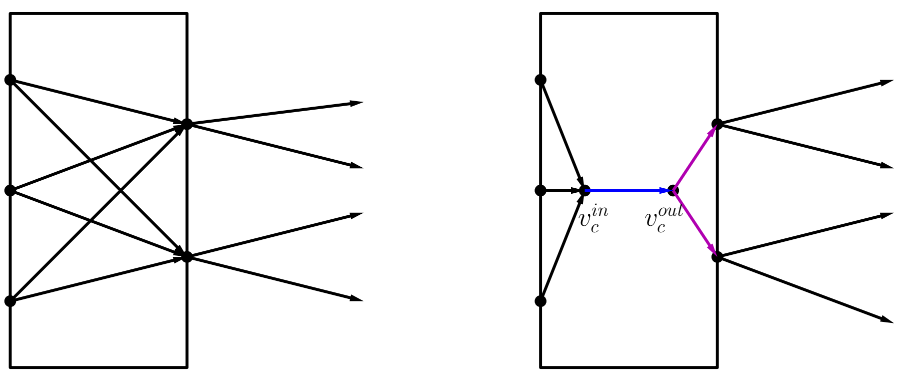


Figure 6.5: An example of the delay optimization graph of a two bit adder. In the left the complete bipartite timing graph of the adder plus two outgoing net timing edges for each output pin are given. In the right the corresponding delay optimization graph with a blue edge, representing circuit layout changes, and violet edges representing net layout changes is shown. With black edges no operations is associated. They are used to represent timing constraints only.

6.4.2 Operations

We call layout changes based on the current layout *operations*. For every $e \in E(G^{\mathcal{O}})$ that represents layout changes, a set $\mathcal{O}(e)$ of potential operations is computed. With every operation $o \in \mathcal{O}(e)$ we associate a local slack change $-\Delta \operatorname{slk}_e^o$ and a cost change Δc_e^o. The local slack change of $(v, w) \in E(G^{\mathcal{O}})$ is the difference of the worst slack among all pins in $w \cup v \cup \Gamma^-_{G^{\mathcal{O}}}(v) \cdots \cup \Gamma^{-k}_{G^{\mathcal{O}}}(v)$, where $k = 0$ for net operations and $k = 2$ for circuit operations. Here $\Gamma^-_G(V') := \{v \in V(G) \setminus V' | (v, v') \in E(G)\}$ are the predecessor nodes of $V' \subset V(G)$, and Γ^{-k}_G is defined recursively, by applying the predecessor operator k times.

If operations would modify exactly one edge delay in $G^{\mathcal{O}}$, a local slack increase is equivalent to a delay decrease. In practice it is more flexible to consider rather slack changes, as this allows for accounting minor delay changes in the predecessor cone and its successor cone.

We now give a list of operations that are in particular applicable, because they effect the delays through the predecessors only marginally.

Threshold Voltage Assignment

As mentioned in the introduction of Chapter 4, circuits can be realized with varying threshold voltages retaining the footprint. Given a book $b \in \mathcal{B}$, let $[b]_{V_t} = \{b_1, b_2, \dots, b_{\max_{[b]_{V_t}}}\} \subset [b]$ be the set of equivalent books that vary only in their threshold voltage, sorted from highest to lowest threshold voltage. The number $i \in \{1, 2, \dots, \max_{[b]_{V_t}}\}$ is called *V_t-level*.

Furthermore, let the function $\text{power}_{\text{static}} : \mathcal{C} \times [\beta(c)] \to \mathbb{R}_+$ specify the static power consumption of a circuit $c \in \mathcal{C}$ when being realized by a book $b \in [\beta(c)]_{V_t}$. The power consumption is translated into a cost in the mathematical model.

As the pin shapes are equal for all $b \in [b]_{V_t}$, the input pin capacitances are almost the same. We assume $\text{pincap}(p_1) = \text{pincap}(p_2)$ for any pair of equivalent pins $p_1 \in b_1 \in [b]_{V_t}$ and $p_2 \in b_2 \in [b]_{V_t}$. In practice the pin capacitances increase by up to 10% from level to level due to varying gate material and thickness, but we ignore this fact here. If the input pin capacitances remain the same, the delay and slew propagations through the predecessor circuits are not affected when assigning a circuit to another V_t-level.

Now each V_t-level has assigned a delay and cost. The static power consumption grows exponentially when lowering the threshold voltage (see Rao et al. [2004]). Thus, when interpolating the discrete delay/cost pairs for each V_t-level, the relaxed cost function is piecewise linear and convex and we can apply Algorithm 13 in the adjusted version from Section 6.3.5.

We conclude this subsection with a short overview on other approaches for the threshold voltage optimization. Having their emphasis on the power analysis of operations, Dobhal et al. [2007] adjust single circuits with the best power delay tradoff. A linear programming formulation where the voltage threshold variables automatically snap to discrete values was proposed by Shah et al. [2005]. Gao and Hayes [2005] formulate a mixed integer linear program. Chou et al. [2005] incorporate threshold voltage optimization into a geometric programming gate sizing approach. Similarly, Mani et al. [2007] and Bhardwaj and Vrudhula [2008] combine both problems while considering statistical power and timing optimization. Engel et al. [2006] fomulate the problem of optimum leakage reduction as a k-cutset problem and solve this *NP*-hard problem heuristically.

Plane Assignment

The electrical properties of metal planes on a computer chip are significantly different. Figure 6.6 shows minimum wire widths of the ten metal planes on an IBM Power6

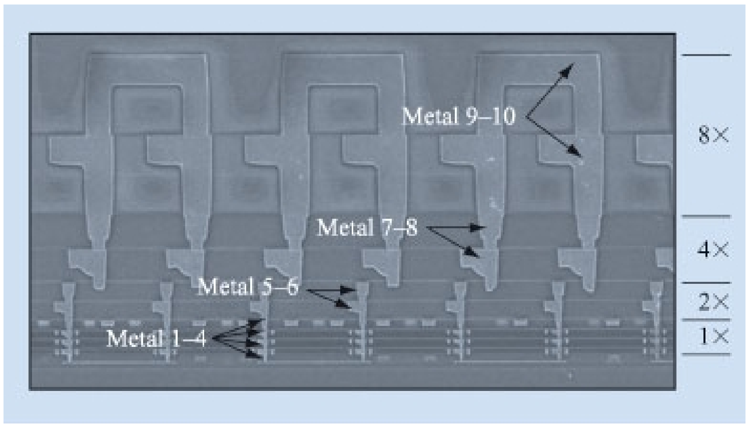


Figure 6.6: Metal layers of the Power6 Microprocessor (Berridge et al. [2007]).

processor. This is a typical wire width distribution for a 65 nm design. The higher the plane is the wider are the wires. In the figure the wire widths in the highest planes are eight times wider than those in the lowest planes. There are basically two reasons for increasing wire widths. First, due to the unavoidable inexact production process, with every additional plane its ground becomes more irregular. It would be impossible to realize a minimum width (1×) wire on the tenth plane. Second, lower planes have small widths to enable better routability, especially when accessing the pins. The drawback is that these wires with a small diameter suffer from high resistances and therefore higher delays. The fat toplevel wires have a much smaller resistance while the effective capacitance between metal and substrate remains about the same, because the distance increases.

The plane assignment problem is to assign each net to a pair (z_x, z_y) of wiring planes $1 \leq z_x, z_y \leq z_{\max}$ such that routability is assured and the slack distribution is optimized. The assignment of individual net-segments is not considered here, as this would introduce an immoderate complication of the physical design process. An assignment (z_x, z_y) will later be transformed into preference routing layers within $\{z_x, , \dots, z_{\max}\}$ for x-segments and $\{z_y, \dots, z_{\max}\}$ for y-segments. Here higher faster layers are still allowed to relax the routing constraints. Thereby, the lower the assigned planes are, the higher is the freedom for routing.

A plane assignment operation either assigns a net to a higher plane pair, where at least one of z_x or z_y is increased, or assigns it to a lower pair. The routability is estimated using the global routing graph $G^{\mathcal{GR}}$ (see Section 2.3.3). The costs should reflect the existing layer assignments. Thus, when a net is assigned to (z_x, z_y), a two-dimensional Steiner tree in the plane is computed and then projected to the overlaying (z_x, z_y)-edges in $G^{\mathcal{GR}}$. A particular assignment is realizable if all used global routing edges have sufficient free capacities. In a simple implementation costs could be defined, depending on the Steiner tree length, and as ∞ if the assignment cannot be realized.

In an advanced application plane assignment could be applied in interaction with a multi-commodity flow global routing algorithm such as in Vygen [2004]. In that framework the costs could be derived from the dual variables in the linear multi-commodity flow program.

Buffering

Instead of assigning single nets, one can also rebuild complete repeater trees for nets. By varying the performance parameter ξ in Section 3.2, alternative realizations can be generated. Each alternative has a certain worst slack at the root and a certain resource allocation cost.

As announced in Section 3.6, the post-optimization of a given repeater tree topology can be modeled as a Budget Problem. Here the task is to optimize the resource allocation for each individual edge in the topology. The wire delay parameter d_{wire} depends on the assumed value of ξ. By varying ξ between 0 and its initial value, which was used to construct the topology, different buffered wire delays at different resource allocation costs can be generated for each edge. The task is to find a minimum cost solution that maximizes the slack at the root, which is known from the initial topology.

6.4.3 Results in Threshold Voltage Optimization

We present results for the threshold voltage optimization problem in the acyclic case with a fixed clock schedule. The experimental results are based on a joint work with Schmedding [2007], who also implemented the algorithm.

Table 6.1 shows the results of voltage threshold optimization on those chips in our testbed which allow for several threshold voltages. The numbers were obtained during the timing refinement step (Section 7.3) of our full timing closure flow and reflect intermediate results of Table 7.1 on page 165. The table shows the worst slack (WS), the sum of negative slacks (SNS), the power consumption before any threshold voltage optimization, after threshold voltage optimization, and after some postoptimization, and finally the running time (CPU Time) of the voltage threshold optimization. The running times were obtained on an Intel Xeon E7220 processor with 2.93 GHz. Note that the sum of negative slacks is computed with respect to the slack target of 0.250 nanoseconds. That is why some SNS numbers are negative although the worst slack is strictly positiv (but below the target).

The postoptimization first performs local search gate sizing (Algorithm 6 on page 74). Then, it iterates once over all circuits to increase the threshold voltages if the worst slack of the circuit remains above the slack target of 250 picoseconds. The power column shows the estimate static power consumption in watt.

For the instances where the total power consumption is marked with a star no accurate power estimates were available. Here the numbers were chosen proportional

Chip	**Before Optimization**			**VT Optimization**			**Postoptimization**			**CPU**
	WS	SNS	P	WS	SNS	P	WS	SNS	P	Time
Franz	0.10	−1	0.01	0.25	0	0.01	0.25	0	0.01	0:00:32
Lucius	−1.71	−96	0.42⋆	−0.69	−22	3.21⋆	−0.56	−22	3.18⋆	0:10:13
Julia	−1.02	−15	0.00	−0.22	−0	0.00	−0.19	−0	0.00	0:01:39
Minyi	−0.89	−11	0.49⋆	0.15	−0	0.54⋆	0.15	−0	0.53⋆	0:03:35
Maxim	−1.67	−337	0.68⋆	−0.55	−137	5.84⋆	−0.52	−139	5.79⋆	0:45:06
Tara	−0.54	−10	2.37⋆	0.03	−0	2.47⋆	0.04	−0	2.46⋆	0:02:05
Bert	−0.67	−73	3.17	−0.63	−30	3.29	−0.63	−30	3.29	0:26:14
Ludwig	−6.02	−18	0.50⋆	−3.25	−3	0.53⋆	−3.25	−3	0.53⋆	0:28:22
Arijan	−2.84	−357	1.36	−2.09	−11	1.43	−2.07	−11	1.43	2:05:57
David	−1.76	−296	5.93	−0.83	−7	6.16	−0.81	−7	6.15	3:17:31
Valentin	−3.48	−1755	2.88	−3.44	−777	3.02	−3.43	−782	3.02	5:22:01
Trips	−0.92	−815	2.97	−0.47	−168	3.39	−0.43	−163	3.39	5:43:23

WS = worst slack in ns, SNS = sum of negative slacks in ms, P = power in watt, times in h:m:s, $S_l^{tgt} = 0.25ns$.

⋆ inaccurate estimates not in watt (see Section 6.4.3).

Table 6.1: Threshold Voltage Optimization Results.

to $area(c) \times 10^{-vt\_level(c)}$, where $area(c)$ is the area of a circuit $c \in \mathcal{C}$ and $vt\_level(c)$ its threshold voltage level. The number of available threshold voltage levels varied between 2 and 3 on all instances.

In general the reported slacks are higher than the slacks would be when setting all circuits to their lowest possible threshold voltage. This is, because a higher threshold voltage is accompanied by slightly higher input pin capacitance. A higher input pin capacitance would slow down more critical predecessors.

As the underlying model is highly non-linear and we are dealing with huge instances, we are not able to compute good lower bounds for the required power consumption. In Table 6.1 the power consumption before optimization gives a weak lower bound. The worst deviation among those rows with reliable power estimates is less than 15% on the chip Trips.

Figure 6.7 shows a typical slack-power tradeoff curve. The curve was obtained on David. It shows a worst slack of less than $\underline{S} \approx -3$ nanoseconds when all circuits are at their highest threshold voltage. The worst slack is improved by increasing the power until the blue line marked with $\overline{S}$ is reached.

At this point, the worst slack is optimized, and the algorithm continues to optimize less critical paths, as described in Section 6.3.7. From that point on, the minimum improvable slack is represented by the curve. The exponential growth of the power consumption is caused by the increasing number of paths becoming critical.

The slightly fuzzy structure of the curve is caused by our succesive relaxation approach. Based on the linear approximation and relaxation we compute continuous variable changes which are then mapped back to discrete changes. This can lead to small temporary slack degradations.

Figure 6.8 shows the slack distribution of the circuits on David before and after optimization. The circuits in the placement plots are colored according to the worst

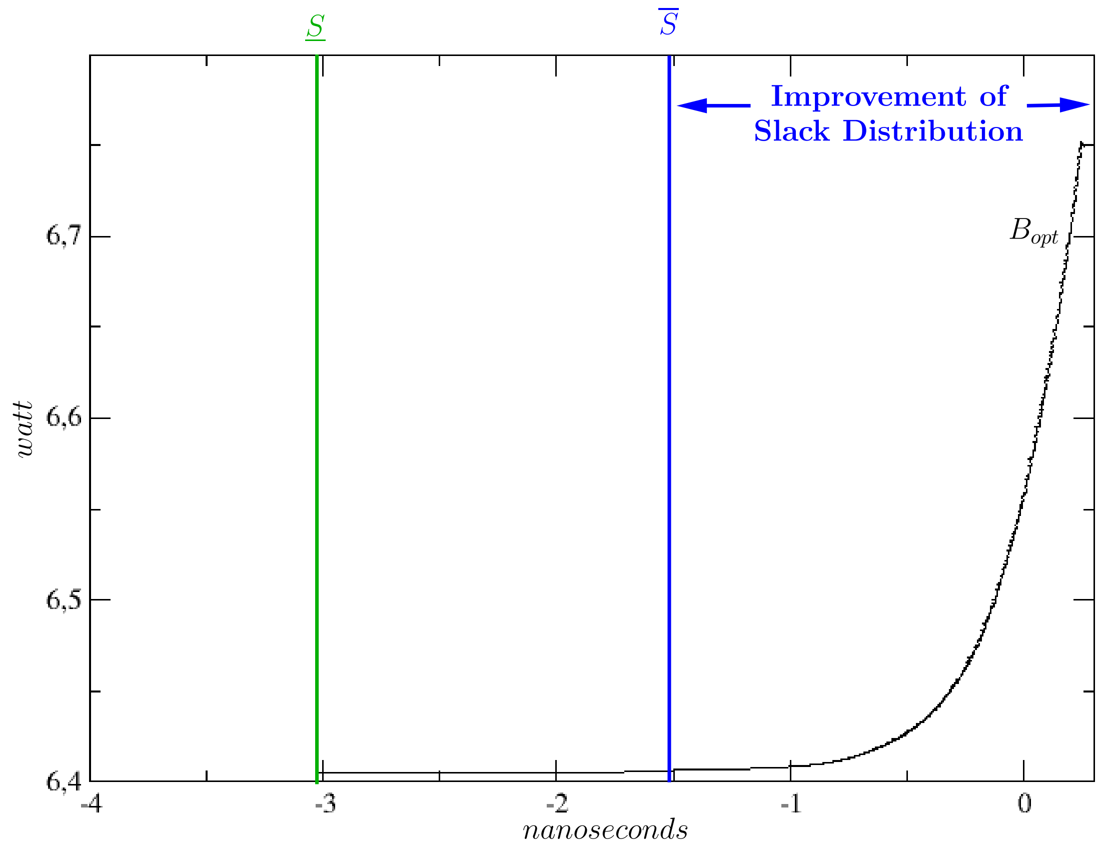
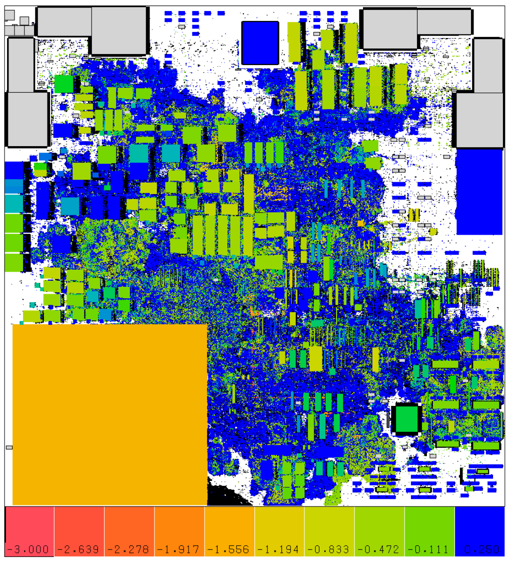
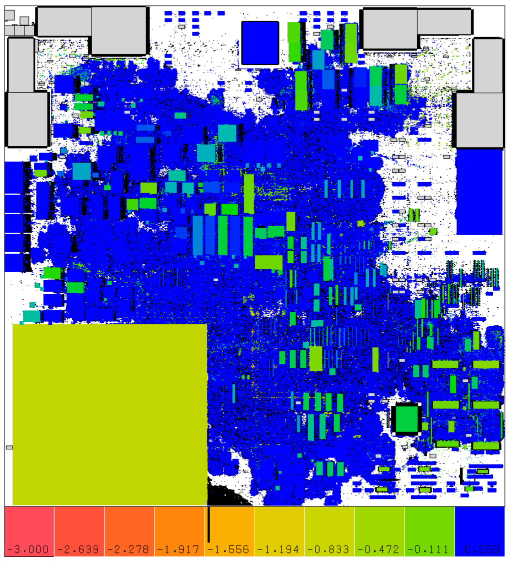


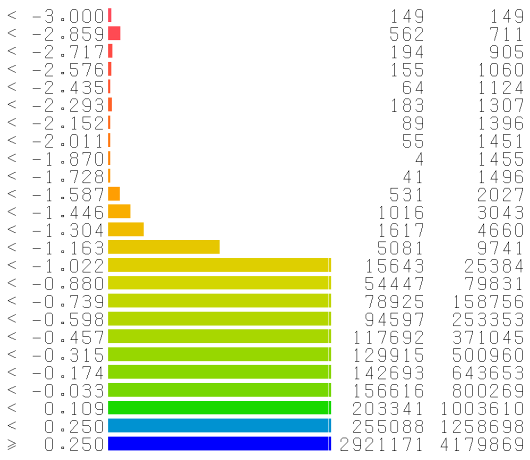
Figure 6.7: Time-Cost Tradeoff Curve on the Chip David.

slack at any of their pins. The colors in the plot and histograms are corresponding.

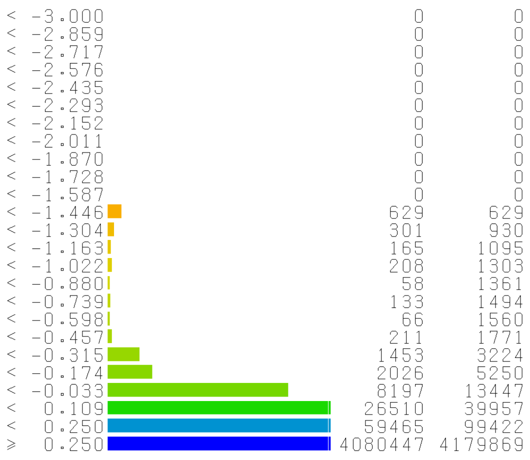
Note that this curve and the slack diagrams were obtained from a common run, but independently from the timing closure run and the numbers in Table 6.1. That is why the slacks differ from those in the table.







<	-3.000	149	149
<	-2.859	562	711
<	-2.717	194	905
<	-2.576	155	1060
<	-2.435	64	1124
<	-2.293	183	1307
<	-2.152	89	1396
<	-2.011	55	1451
<	-1.870	4	1455
<	-1.728	41	1496
<	-1.587	531	2027
<	-1.446	1016	3043
<	-1.304	1617	4660
<	-1.163	5081	9741
<	-1.022	15643	25384
<	-0.880	54447	79831
<	-0.739	78925	158756
<	-0.598	94597	253353
<	-0.457	117692	371045
<	-0.315	129915	500960
<	-0.174	142693	643653
<	-0.033	156616	800269
<	0.109	203341	1003610
<	0.250	255088	1258698
≥	0.250	2921171	4179869



<	-3.000	0	0
<	-2.859	0	0
<	-2.717	0	0
<	-2.576	0	0
<	-2.435	0	0
<	-2.293	0	0
<	-2.152	0	0
<	-2.011	0	0
<	-1.870	0	0
<	-1.728	0	0
<	-1.587	0	0
<	-1.446	629	629
<	-1.304	301	930
<	-1.163	165	1095
<	-1.022	208	1303
<	-0.880	58	1361
<	-0.739	133	1494
<	-0.598	66	1560
<	-0.457	211	1771
<	-0.315	1453	3224
<	-0.174	2026	5250
<	-0.033	8197	13447
<	0.109	26510	39957
<	0.250	59465	99422
≥	0.250	4080447	4179869

Slacks before optimization Slacks after optimization

Figure 6.8: Effect of Threshold Voltage Optimization on David.

7 Timing Driven Loop

In this final chapter we show how to combine the algorithms from the previous chapters efficiently into a timing closure flow (Algorithm 15). The literature on timing closure flows is sparse. A complete physical design flow also including logic re-synthesis, clock insertion, and routing can be found in Trevillyan et al. [2004]. A common gate sizing and placement was described by Chen et al. [2000]. Vujkovic et al. [2004] proposed a flow where transistor sizing plays a central role. An approach that first splits the design into good physical hierarchies was given by Cong [2002].

We present a new design flow for timing closure, which is especially designed to handle large multi-million circuit instances. Special care is taken to keep the overall running time small, without compromising the quality. In the top level the flow is separated into two main blocks.

First, the circuits are placed and data paths are optimized. As placement and timing optimization are called repeatedly within this phase, we call it the *timing driven (placement) loop*. Second, after data paths are optimized and the locations of the registers are known, the clocktrees are implemented. This step starts with the three step clock skew optimization for late mode, early mode, and time window constraints, as described in Section 5.2.2.

Algorithm 15 Timing Closure Flow

```
1: Timing Driven (Placement) Loop;
2: (Re-) Optimize clock schedule and build clocktrees;
```

During the timing driven loop, when clocktrees are not yet implemented, the timing analysis within the preliminary trees is meaningless. Instead, we assume an ideal clocktree with idealized clock arrival times at the registers. Note that clock skew scheduling could be performed incrementally within the timing driven loop. It then defines an ideal and optimized schedule.

The timing driven loop is described in Algorithm 16. The main components are a placement algorithm that minimizes the weighted sum of quadratic netlengths (Section 7.1), and the fast timing optimization (Section 7.2). The main purpose of the timing optimization is to obtain a predictable timing evaluation of the given placement and to guide the netweighting (Section 7.4), which penalizes timing critical nets. While the timing optimization in line 6 consists simply of buffering and gate sizing, other techniques like threshold voltage optimization and plane assignment are performed during the timing refinement and legalization in line 8.

When the algorithm first enters the loop, reasonable timing information is not available, and the placement is targeting minimum netlength. Nets in the combina-

Algorithm 16 Timing Driven Loop

```
1: Rip out repeaters;
2: Insert repeaters based on a QP solution;
3: repeat
4:     Netweighting; (Section 7.4)
5:     Placement; (Section 7.1)
6:     Timing optimization; (Algorithm 17, Section 7.2)
7: until Satisfying timing reached
8: Refine timing & legalize; (Section 7.3)
```

torial logic will be assigned a common netweight and clocktree nets will be weighted as explained in Section 7.4.2.

After timing optimization in line 6 it may turn out that the timing constraints are not achievable with the current placement, because the critical paths cover too long distances. In such a case the loop is repeated and the next placement is driven by increased netweights on the critical data paths, as we will describe in Section 7.4.1.

In line 5, the placement need not run all the way through legalization to obtain sufficient information on the timing criticality of the paths. To reduce the running time, we terminate the placement as soon as circuit locations are bounded to small regions of approximately 30×30 square micrometers. Final spreading and legalization are done only in the timing refinement and legalization step (Section 7.3).

In line 2 resource efficient repeaters are inserted based on a single quadratic programming (QP) placement solution, before any overlap removal is performed. Thus their number is usually a lower bound for the final number of repeaters in the design. Their purpose is to reserve space for later repeater insertion. Using the QP information, the topologies respect the intended positions of the sinks and perturb the placement net model marginally.

7.1 Quadratic Placement

For our tests, we used the BonnPlace program, which was developed at the Research Institute for Discrete Mathematics at the University of Bonn (Brenner et al. [2008]). The placement is done in a series of levels. Each level begins with a quadratic placement, in which the total squared interconnect length of pin pairs from a common net is minimized. This minimization can be performed independently for x and y coordinates. Exemplarily, we give the problem formulation for the x-coordinates:

$$\min \sum_{N \in \mathcal{N}} \frac{w(N)}{|N|-1} \sum_{p,q \in N} (x(p) + x_{offs}(\gamma(p)) - x(q) - x_{offs}(\gamma(q)))^2$$

where $w(N) \in \mathbb{R}_{\geq 0}$ is a netweight, $x(\gamma(p))$ is the x-coordinate of $\gamma(p) \in \mathcal{C} \dot{\cup} \mathcal{I}$, and $x_{offs}(p)$ is the offset of p relative to $\gamma(p)$, $p \in P$.

After weighted quadratic netlength minimization, the placement area is partitioned, and the circuits are assigned to the resulting subregions minimizing the total movement (Brenner and Struzyna [2005]). Now the subregions are processed recursively, while circuits are constrained into their assigned subregions. The algorithm is able to handle routing congestion (Brenner and Rohe [2003]) and many additional constraints that can be useful in a physical design process. Nets with high weights will be kept particularly short by the algorithm.

Compared to a pure (linear) netlength minimization, the quadratic netlength minimization has several advantages. First, in the presence of preplaced circuits or IO-ports, the quadratic formulation has a unique solution for most circuits, which allows to deduce more information for partitioning the circuits for overlap removal. Second, the solution is almost invariant under small netlist changes (Vygen [2007]), which might be introduced by the timing optimization or by external netlist corrections. Third, the wire delays of unbuffered wires grow also quadratically with the netlength. Finally, it can be solved extremely fast by the conjugate gradient method.

One difficulty with the quadratic minimization is that the total quadratic netlength of a long wire depends on the number of inserted repeaters. Adding a repeater in the middle of a two-point connection halves its quadratic netlength. Repeaters are particularly inserted into timing critical nets. Therefore, in a second placement run the effective linear netlength of such a net would increase and lead to even larger delays.

This problem can be overcome by different means. First, the decrease in quadratic netlength can be compensated by increasing the weight of these nets. This is what we applied in our experimental runs later in this chapter. Second, the repeaters could be ripped-out before placement, so that unbuffered netlengths are optimized. However, the area allocation for the necessary repeaters becomes difficult if repeaters are missing.

Finally, one could not only minimize the quadratic lengths of all nets but additionally for each repeater tree the quadratic distances between source and sinks. The quadratic length of these source-sink connections would not depend on the number of inserted repeaters. Furthermore, the repeaters themselves would be spread evenly between source and sinks due to the quadratic netlength minimization.

Within legalization the netlengths are ignored, but the total weighted quadratic movement of the circuits from their current location to a legalized one is minimized (Brenner and Vygen [2004]). Netweights that qualify the criticality of a net are now translated into circuit weights. Thereby the displacement of timing critical circuits is penalized and reduced.

For avoidance of routing congestion, the global routing is estimated by a fast statistical global routing algorithm, after each placement level. Then circuit sizes are virtually increased proportionally to the routing capacity overload in their proximity. Such congested circuits will be spread in subsequent placement levels, and fewer nets will pass a congested region. In the timing driven loop, the virtual circuit sizes from a previous placement run can be reused from the first level on, as an initial good guess for the virtual circuit sizes.

7.2 Timing Optimization

After global placement, we optimize slacks by repeater insertion and gate sizing in line 6 of Algorithm 16. Both components work globally on the complete netlist. Algorithm 17 gives a detailed view. Only in line 5 the algorithm does not work globally but only the most critical paths are improved by local search gate sizing. The primary purpose of this timing optimization is to generate realistic slacks for the netweight generation. Therefore they must be both fast and of high quality.

Algorithm 17 Timing Optimization (Section 7.2)

```
1: Power efficient repeater insertion;
2: Fast gate sizing; (Algorithm 3 on page 67)
3: Slack efficient repeater insertion;
4: Fast gate sizing; (Algorithm 3 on page 67)
5: Local search gate sizing; (Section 4.5)
```

For repeater insertion, we use the topology generation from Chapter 3 combined with a fast buffering routine shortly described in Section 3.7 (see also Bartoschek et al. [2006, 2007b]). Both components—topology generation and buffering—allow seamless scaling between power and performance. The scaling can be controlled by a parameter $\xi \in [0,1]$ as described in Chapter 3. In line 1, the topology is chosen close to a minimum Steiner tree topology and the capacitance limit is relaxed. Here we have chosen $\xi = 0.1$ in our experiments to avoid extreme daisy chains as they would occur with $\xi = 0.0$. The parameter λ that controls the possibility to distribute delay to the two branches in a topology (Section 3.4.1) was chosen as $\lambda = \frac{1}{4}$.

The inserted repeaters are sized roughly, leaving the actual sizing to the subsequent gate sizing. During the repeater insertion, time-consuming timing updates are suppressed. Timing is updated only non-incrementally within the fast gate sizing and incrementally in the local search.

After the first fast gate sizing all capacitance and slew constraints are met. Thus the computed slacks are meaningful when building the first slack efficient trees.

In line 3, faster topologies are generated and the fastest capacitance limit is used for trees with negative slacks. However, we do no drive the topologies to the extreme, as the current slacks are only a snapshot and not updated incrementally after a tree has been rebuild. In this step we have chosen $\xi = 0.5$ for topology generation and $\xi = 1.0$ for buffer insertion in our experiments.

7.3 Timing Refinement and Legalization

Within the final refinement and legalization in line 8 of the timing driven loop (Algorithm 16), we first perform the last placement levels and legalize the design to account for displacements of circuits that do not fit into narrow macro alleys to which they may have been assigned. For this purpose netweights are updated adaptively according to Section 7.4. Then, we remove capacitance and slew limit violations that have been created during legalization. Note that Algorithm 17 does not leave such violations and they can only be caused by placement changes. Therefore their number is small at this stage and occurs mainly on non-critical nets. Running the circuit assignment as described in Section 4.4.5 removes most violations quickly. Remaining violations are fixed by repeater insertion.

To improve the worst slacks further, several slack refinement steps are performed. First, we rebuild the most critical repeater trees (approximately 1‰) with an exhaustive dynamic programming repeater insertion based on a slack driven topology ($\xi = 0.7$ in our experiments). Second, if the chip allows several threshold voltages, we compute an optimum threshold voltage assignment according to 6.4.2 with respect to a fixed clock schedule (Section 6.3.8). Third, we assign long two-point connections to higher faster planes, while using an optimum repeater spacing for the corresponding wiring mode. In addition we assign individual nets to higher planes. Thereby we do not allocate more than 25% of the wiring resources, in order to reserve space for later clocktree insertion. We applied a simple greedy strategy for plane assignment, which always assigns a net the biggest time gain per cost (resource allocation) ratio, among those nets where the worst slack occurs, to a higher plane. The software-integration of the plane assignment into our time-cost tradeoff framework, described in Chapter 6, is an ongoing work.

Between these three steps we run the local search gate sizing, to achieve a resource efficient slack improvement, based on more globally changed delays and slews. The timing refinement and legalization phase is closed with a final placement legalization.

In this phase it would also make sense to perform local logic optimizations, such as pin swapping or De Morgan transforms (see also Trevillyan et al. [2004]), or more complex optimizations such as the worst path resynthesis proposed in Werber et al. [2007].

7.4 Netweights

Although used for decades, netweighting is barely understood. Sometimes the initially critical paths are well shortened, but previously uncritical paths become critical after timing-driven placement and optimization. Sometimes critical paths are not shortened sufficiently. Such cases can be overcome by iterating the timing driven loop and computing adaptive netweights: Let w_n be the vector of netweights, as it was used in the (n+1)-th iteration of Algorithm 16 (the n-th iteration with intended weights on data nets) and let w be the vector of netweights computed at the beginning of iteration $n + 2$. Adaptive netweights are of the form

$$w_{n+1} := \theta_{old} \cdot w_n + \theta_{new} \cdot w$$

where $\theta_{old}, \theta_{new} \in \mathbb{R}^+$ with $\theta_{old} + \theta_{new} \geq 1$ and $\theta_{old} > \theta_{new}$. Usually $\theta_{old} + \theta_{new} = 1$. If the sum is bigger than one, the total weighting tends to increase from iteration to iteration. This can be useful when the loop shall start with moderate weights and is iterated until satisfying weights are generated. We compute adaptive netweights also during placement and legalization at the end of the loop. In our experiments we used $\theta_{old} = \frac{2}{3}$ and $\theta_{new} = \frac{1}{3}$.

7.4.1 Data Netweights

On the large placement areas of multi-million circuit designs, the most common purpose of netweights is to shorten timing-critical connections that cover several millimeters. In extreme cases it is necessary to generate netweights that are up to 30 times the default netweight. Such weights heavily influence the total netlength and routability. Thus it is important to control the maximum netweight and maximum average netweight.

In our experience, simple weighting strategies often create very good results by tuning parameters like the maximum absolute weight and the maximum average weight.

Weighting by Percentile

Here the nets are sorted by non-decreasing slack. Then, given the maximum absolute weight, the nets are weighted by the percentile-rank of their slack, but not by the actual slack value. The percentile weights are adjusted such that the most critical nets obtain the maximum absolute weight and the average weight is met. With this method, the total wire length tends to increase moderately. Satisfying timing results are achieved after at most three or four loop iterations in Algorithm 16, that are two or three iterations with weighted data nets. Often a single weighted iteration is sufficient. This approach is quite tolerant to poor timing assertions that create a few meaningless, extremely negative slacks. More advanced algorithms that use the slack values more directly often become less successful due to preliminary timing assertions. Another advantage is that the total amount of weighting is independent from the slack range. In the adaptive netweighting scheme this ensures that netweights are not decreasing from iteration to iteration, because the slacks improve. We used this method in our experiments in Section 7.5.

Weighting by Value

Here the slacks are mapped to netweights by a monotonically decreasing function, for instance by $(\alpha \cdot \exp^{\mathrm{S}-\ \mathrm{slk}/\beta})$, where S is the worst slack in the design and slk the slack of the current net. By choosing α and β appropriately, maximum absolute and average netweights can be controlled. If the timing assertions are reliable, this method mostly gives slightly better results in a single iteration than the percentile weighting. When the loop is iterated this advantage usually vanishes. In case of some paths with unreliable assertions these path may dominate the netweighting, and thus require user interaction to hide those path slacks.

On individual test instances, advanced methods that account for the number of critical paths that traverse a net (see Kong [2002]) achieve superior timing results, though the increase in net length tends to be hardly controlled. We did not compare our method with sophisticated approaches that try to anticipate the effect of net weights and also consider driver sensitivities (Ren et al. [2005]), because driver strengths may not correlate with the criticality. Critical driver circuits can be weak to speed up a more critical predecessor. In a slightly different context, they could be made much stronger. However, we expect that such an approach can be beneficial during legalization.

7.4.2 Clock Netweights

Todays ASICs contain hundreds of larger and smaller clock domains. In our flow clocktrees are inserted after placement and optimization, just before detailed routing. It is extremely important for a successful and resource efficient clocktree insertion to bound the diameter of small clock domains. To achieve this, clock nets must not be ignored during placement. Instead, we assign netweights to clock nets. The value is chosen inversely proportional to the cycle time. The weighting range should not exceed the average netweight in the placement too much. Otherwise the attached registers will collapse and degrade the overall result. The quadratic objective function already helps to prevent registers from being placed far away from others. Small high-speed domains, which can often be found in the IO logic, are kept particularly small. Large nets with more than 10,000 pins are ignored by BonnPlace to keep the algorithm fast.

7.5 Timing Driven Loop Results

The tools we developed and implemented are integrated into the IBM design automation tools ChipBench™ and PDS™. For timing analysis we used the IBM EinsTimer™ program, which is also integrated into ChipBench™ and PDS™. Also note that we used the scan optimization program within ChipBench™ to reoptimize scan paths after the global placement. Because of the arbitrary and optimizable order by which registers are linked in the scan chains, connections to scan input pins at the registers are ignored during placement.

We ran the timing-driven loop with netweights set by percentile rank on all of our test instances from Section 2.7. To limit placement perturbation we set the maximum weight to at most 15 times the default netweight of 1 and the average net weight to at most 1.3 times the default netweight. We ran two iterations of Algorithm 16, one iteration without data netweights and one weighted iteration. After 3–5 iterations of fast gate sizing, the worst path delay is usually within 5% of its final value. Therefore we restrict the number of iterations in line 2 of Algorithm 17 to at most 6 and in line 4, where it restarts on a preoptimized input, to at most 5.

For all designs a slack target of 0.25 ns was specified. The worst slack (WS), the sum of negative slacks (SNS) with respect to the slack target, the standard circuit area in design units, and the running time are given in Table 7.1 for both iterations, and the refinement and legalization step. All running times were obtained on a 2.93 GHz Intel Xeon E7220 processor. Some designs contain large reset trees with several hundred thousand sinks. Their construction is contained in the reported numbers.

The Tables 7.2, 7.3, and 7.4 show the running time contributions of the main subprograms within each loop iteration, and the legalization and refinement step. Note that the placement was running in parallel on 4 processors, while all other steps were running sequentially. About one half of the running time of a loop iteration is spent in placement and the other half in timing optimization. The placement portion is smaller for small chips and larger for large chips. The scan optimization, of which the implementation is beyond our scope, was added to the tables because it accounts for a significant amount of running time. In this point the flow might be improved by calling the scan optimization only once after the last iteration of the loop. We call it in every iteration because the number of inserted repeaters would grow when unnecessarily long scan nets are buffered.

The fast gate sizing within the timing refinement and legalization step was used for electrical correction only. The running times in this last step are dominated by threshold voltage optimization and plane assignment. As they are called only once in the design flow this is tolerable. One could of course use faster heuristics for these tasks during global optimization in Algorithm 17.

The chip design Valentin is a special instance with extraordinary high running times, especially within placement. It is a preliminary netlist that has been regarded as absolutely unroutable. After running the timing driven loop with congestion-aware placement, routability was achieved easily. The second placement iteration uses the congestion information from the first iteration from the very beginning. Therefore a generally better routability is achieved than by running a single congestion-driven placement. Much running time is required in placement to remove routing congestion. Furthermore, the netlist contains many high-fanout nets of 100–200 sinks. These were not buffered frequently because of loose timing constraints. Their Steiner tree estimations and wire delay calculations had a bad impact on the running times of the timing analysis during gate sizing.

Using a preliminary version of Algorithm 17, the running time of timing optimization on the chip David was reduced from 3 days spent by an industrial tool to 6 hours. This speed-up enabled a significant design time reduction, and enabled the engineers at IBM to perform several iterations of the timing driven loop to achieve a better timing aware placement. Final netlist version of the chips Lucius, Tara, Arijan, Ludwig, David, Valentin and many other chips not listed here were and are currently designed for actual tape-out using 2–4 iterations of our timing driven loop.

Chip	**1st Iteration**				**2nd Iteration**				**Refine & Legalize**			
	WS	**SNS**	**Area**	**Runtime**	**WS**	**SNS**	**Area**	**Runtime**	**WS**	**SNS**	**Area**	**Runtime**
	(ns)	(ms)		(h:m)	(ns)	(ms)		(h:m)	(ns)	(ms)		(h:m)
Fazil	−1.221	−14	610	0:03:49	−1.051	−14	609	0:03:14	−0.828	−14	612	0:04:50
Franz	−0.076	−2	754	0:04:21	−0.014	−2	771	0:03:17	0.225	−0	765	0:12:15
Felix	−0.678	−20	809	0:04:32	−0.673	−21	815	0:03:59	−0.459	−20	812	0:07:51
Lucius	−0.853	−43	384	0:05:01	−0.943	−47	399	0:04:42	−0.369	−17	396	0:20:15
Julia	−0.832	−6	3092	0:12:51	−0.634	−5	3062	0:11:42	−0.184	−0	3069	0:11:44
Minyi	−1.370	−17	4866	0:28:46	−0.948	−11	4819	0:18:04	0.182	−0	4840	0:47:56
Maxim	−2.179	−285	6127	0:27:50	−1.783	−296	6482	0:29:59	−0.350	−109	6539	1:46:30
Tara	−0.576	−29	6421	0:45:08	−0.645	−14	6132	0:41:15	0.223	−0	6234	0:29:31
Bert	−0.845	−61	19485	1:20:59	−0.742	−58	19552	1:08:11	−0.624	−22	19617	1:25:32
Karsten	−3.113	−59	39731	4:01:19	−2.324	−41	39339	3:43:40	−1.634	−285	38841	2:36:45
Ludwig	−6.998	−80	42793	3:35:24	−6.053	−20	42696	2:48:45	−0.854	−1	42894	6:12:10
Arijan	−4.476	−471	67296	5:42:11	−2.836	−370	67415	4:24:53	−1.239	−10	67701	7:01:31
David	−4.426	−521	66226	6:19:34	−1.810	−300	65900	5:45:45	−0.353	−4	66353	9:47:03
Valentin	−4.193	−1841	97384	11:38:18	−3.490	−1766	96982	9:41:01	−1.812	−785	89821	12:36:44
Trips	−1.379	−853	76837	8:35:56	−1.322	−735	77304	8:47:53	−0.051	−51	77531	13:29:21

Table 7.1: Timing Driven Loop Results

Chip	Placement	Scan Opt.	Repeater Trees	Gate Sizing Fast	Gate Sizing Local	Total
Fazil	0:00:54	0:00:09	0:01:10	0:00:47	0:00:19	0:03:49
Franz	0:01:09	0:00:02	0:01:11	0:01:10	0:00:22	0:04:21
Felix	0:00:56	0:00:04	0:01:31	0:01:04	0:00:28	0:04:32
Lucius	0:01:16	0:00:01	0:01:24	0:01:21	0:00:27	0:05:01
Julia	0:04:53	0:00:24	0:03:13	0:01:56	0:00:21	0:12:51
Minyi	0:09:15	0:00:32	0:08:28	0:05:45	0:00:38	0:28:46
Maxim	0:08:02	0:00:01	0:08:59	0:06:07	0:00:59	0:27:50
Tara	0:19:05	0:00:10	0:12:59	0:08:10	0:01:02	0:45:08
Bert	0:27:27	0:05:06	0:18:58	0:15:20	0:02:08	1:20:59
Karsten	1:18:30	0:26:30	0:59:45	0:47:33	0:03:00	4:01:19
Ludwig	1:10:33	0:10:14	0:57:00	0:42:41	0:02:44	3:35:24
Arijan	2:19:39	0:04:11	1:27:52	1:14:14	0:06:17	5:42:11
David	2:24:35	0:05:33	1:35:47	1:32:42	0:08:23	6:19:34
Valentin	5:54:20	0:06:34	2:24:28	2:16:59	0:08:11	11:38:18
Trips	3:13:12	0:44:18	2:00:00	1:28:08	0:12:05	8:35:56

Table 7.2: Running times of the first iteration (hh:mm:ss)

Chip	Placement	Scan Opt.	Repeater Trees	Gate Sizing Fast	Gate Sizing Local	Total
Fazil	0:00:50	0:00:01	0:00:47	0:00:44	0:00:25	0:03:14
Franz	0:01:04	0:00:01	0:00:36	0:00:55	0:00:21	0:03:17
Felix	0:00:58	0:00:03	0:00:58	0:01:01	0:00:30	0:03:59
Lucius	0:01:40	0:00:01	0:01:08	0:01:18	0:00:18	0:04:42
Julia	0:05:54	0:00:15	0:01:35	0:01:50	0:00:16	0:11:42
Minyi	0:08:23	0:00:30	0:02:20	0:03:45	0:00:33	0:18:04
Maxim	0:09:59	0:00:03	0:06:41	0:08:34	0:02:09	0:29:59
Tara	0:19:14	0:00:10	0:07:18	0:10:22	0:01:00	0:41:15
Bert	0:25:55	0:05:08	0:10:05	0:15:15	0:01:58	1:08:11
Karsten	1:31:29	0:22:53	0:32:03	0:51:47	0:02:37	3:43:40
Ludwig	1:12:23	0:04:24	0:23:35	0:35:51	0:05:26	2:48:45
Arijan	2:03:22	0:01:56	0:40:42	1:06:10	0:04:09	4:24:53
David	2:42:53	0:03:18	0:51:25	1:30:27	0:07:08	5:45:45
Valentin	5:29:07	0:05:52	1:16:08	1:56:31	0:17:52	9:41:01
Trips	4:10:13	0:42:52	1:14:10	1:34:38	0:13:06	8:47:53

Table 7.3: Running times of the second iteration (hh:mm:ss)

Chip	**Placement**	**Repeater Trees**	**Gate Sizing Fast**	**Gate Sizing Local**	**V_t-Opt.**	**Plane Opt.**	**Total**
Fazil	0:00:35	0:00:44	0:00:12	0:02:07	0:00:00	0:00:08	0:04:50
Franz	0:00:34	0:04:30	0:00:40	0:04:09	0:00:32	0:00:05	0:12:15
Felix	0:00:40	0:01:10	0:00:19	0:02:06	0:00:00	0:02:27	0:07:51
Lucius	0:00:56	0:00:49	0:00:08	0:02:19	0:10:13	0:04:38	0:20:15
Julia	0:02:15	0:02:26	0:00:41	0:00:57	0:01:39	0:00:14	0:11:44
Minyi	0:06:53	0:19:32	0:03:13	0:03:52	0:03:35	0:00:10	0:47:56
Maxim	0:06:58	0:03:48	0:01:05	0:12:42	0:45:06	0:28:53	1:46:30
Tara	0:07:30	0:06:29	0:01:32	0:02:27	0:02:05	0:00:27	0:29:31
Bert	0:10:35	0:11:42	0:04:29	0:07:20	0:26:14	0:04:21	1:25:32
Karsten	0:31:59	0:31:51	0:13:18	0:16:54	0:00:00	0:06:54	2:36:45
Ludwig	1:14:11	1:27:39	1:05:08	0:23:08	0:28:22	0:06:52	6:12:10
Arijan	0:40:47	1:16:10	0:47:42	0:28:39	2:05:57	0:10:02	7:01:31
David	0:57:17	1:15:39	1:06:58	0:45:00	3:17:31	0:19:16	9:47:03
Valentin	1:00:49	1:41:11	1:02:11	1:05:57	5:25:55	0:09:38	12:36:44
Trips	1:04:44	0:44:40	0:25:06	0:45:15	5:43:23	2:56:03	13:29:21

Table 7.4: Running times in timing refinement and legalization (hh:mm:ss)

Notation Index

$\mathbb{N}$	set of natural numbers	
$\mathbb{Z}$	set of integral numbers	
$\mathbb{Q}$	set of rational numbers	
$\mathbb{R}$	set of real numbers	
$V(G)$	set of vertices of a graph G	
$E(G)$	set of edges of a graph G	
$\delta^-(X)$	incoming edges $\{(u,v) \in E(G) \mid u \in V(G) \setminus X, v \in X\}$ of a vertex set X in a graph G	
$\delta^+(X)$	outgoing edges $\{(v,w) \in E(G) \mid w \in V(G) \setminus X, v \in X\}$ of a vertex set X in a graph G	
$\mathcal{S}(x)$	set of shapes of object x	7
$\mathcal{I}$	chip image	8
$P(\mathcal{I})$	primary input or output pins	8
$\mathcal{C}$	set of circuits	9
P	set of pins	9
$\mathcal{N}$	set of nets	9
$Pl(x)$	placement location of object x	9
$P(o)$	pins of object $o \in \mathcal{B} \cup \mathcal{C} \cup \mathcal{N} \cup \mathcal{I}$	9
$P_{in}(o)$	input pins of object $o \in \mathcal{B} \cup \mathcal{C} \cup \mathcal{N} \cup \mathcal{I}$	9
$P_{out}(o)$	output pins of object $o \in \mathcal{B} \cup \mathcal{C} \cup \mathcal{N} \cup \mathcal{I}$	9
$\mathcal{B}$	library of circuit definitions/books	8
$\mathcal{P}(B)$	pin definitions of circuit definition $B \in \mathcal{B}$	8
β	Mapping of circuits to Books ($\beta : \mathcal{C} \to \mathcal{B}$)	9
$G^{\mathcal{GR}}$	global routing graph	10
PLL	phase locked loop	10
η	timing mode (*early* or *late*)	11
at	arrival time	11
slew	slew	11
$G_{\mathcal{T}_B}$	model timing graph of book $B \in \mathcal{B}$	12
$\text{downcap}(p,\eta)$	downstream capacitance of pin $p \in P$ in mode $\eta \in \{early, late\}$	12
$\text{wirecap}(N,\eta)$	wire capacitance of net $N \in \mathcal{N}$ and $\eta \in \{early, late\}$	12
$\text{pincap}(p,\eta)$	input pin capacitance of a sink pin $p \in P$ and $\eta \in \{early, late\}$	13
$\text{caplim}(p)$	load capacitance limit at a source pin p	13
$\text{slewlim}(p)$	input slew limit at a sink pin p	13

ϑ	delay function	13
λ	slew function	13
$V^{\mathcal{T}}$	set of timing nodes	15
$E_p^{\mathcal{T}}$	set of propagation edges	15
$G_p^{\mathcal{T}}$	timing propagation graph	15
$V^{\mathcal{T}^{start}}$	signal start nodes	16
$\mathfrak{S}(p)$	signal set of a measurement point p	16
σ	a signal	16
$E_t^{\mathcal{T}}$	set of test edges	21
$E^{\mathcal{T}}$	set of timing edges ($E^{\mathcal{T}} = E_p^{\mathcal{T}} \dot{\cup} E_t^{\mathcal{T}}$)	21
$G^{\mathcal{T}}$	timing graph ($G^{\mathcal{T}} = (V^{\mathcal{T}}, E^{\mathcal{T}})$)	21
rat	required arrival time	22
$G^{\mathfrak{S}}$	signal graph	24
$V^{\mathfrak{S}}$	signal graph vertices	24
$c^{\mathfrak{S}}$	signal graph costs	24
$E_p^{\mathfrak{S}}$	signal graph propagation edges	25
$E^{\mathfrak{S}}$	signal graph edges	25
$\mathrm{S}_{\mathrm{l}}^{\mathrm{tgt}}$	late slack target	30
V_t	threshold voltage	61
P_{cs}	set of clock sink pins	92
V_{cs}	clock sink vertices in the slack balance graph	92
$G^{\mathcal{R}}$	register graph	95

Bibliography

Agarwal, A., Chopra, K., Blaauw, D., and Zolotov, V. [2005]: Circuit optimization using statistical static timing analysis. Proceedings of the 42nd Annual Conference on Design Automation, 321–324, 2005.

Ahuja, R. K., Hochbaum, D. S., and Orlin, J. B. [2003]: Solving the Convex Cost Integer Dual Network Flow Problem. Management Science 49 (2003), 950–964.

Albrecht, C. [2001]: Optimierung der Zykluszeit und der Slackverteilung und globale Verdrahtung. Dissertation, University of Bonn, 2001.

Albrecht, C. [2006]: Efficient incremental clock latency scheduling for large circuits. Proceedings of the Conference on Design, Automation and Test in Europe, 1091–1096, 2006.

Albrecht, C., Korte, B., Schietke, J., and Vygen, J. [1999]: Cycle Time and Slack Optimization for VLSI-Chips. Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, 232–238, 1999.

Albrecht, C., Korte, B., Schietke, J., and Vygen, J. [2002]: Maximum Mean Weight Cycle in a Digraph and Minimizing Cycle Time of a Logic Chip. Discrete Applied Mathematics 123 (2002), 103–127.

Alpert, C. J., and Devgan, A. [1997]: Wire segmenting for improved buffer insertion. Proceedings ACM/IEEE Design Automation Conference, 588–593, 1997.

Alpert, C. J., Gandham, G., Hrkic, M., Hu, J., Kahng, A. B., Lillis, J., Liu, B., Quay, S. T., Sapatnekar, S. S., and Sullivan, A. J. [2002]: Buffered Steiner trees for difficult instances. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 21 (2002), 3–14.

Alpert, C. J., Gandham, G., Hrkic, M., Hu, J., and Quay, S. T. [2003]: Porosity aware buffered steiner tree construction. Proceedings of the ACM International Symposium on Physical Design, 158–165, 2003.

Alpert, C. J., Gandham, G., Hu, J., Neves, J. L., Quay, S. T., and Sapatnekar, S. S. [2001]: A Steiner tree construction for buffers, blockages, and bays. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 20 (2001), 556–562.

Bartoschek, C., Held, S., Rautenbach, D., and Vygen, J. [2006]: Efficient Generation of Short and Fast Repeater Tree Topologies. Proceedings of the International Symposium on Physical Design, 120–127, 2006.

Bartoschek, C., Held, S., Rautenbach, D., and Vygen, J. [2007a]: Efficient algorithms for short and fast repeater trees. I. Topology Generation. Technical Report 07977, Research Institute for Discrete Mathematics, University of Bonn, 2007.

Bartoschek, C., Held, S., Rautenbach, D., and Vygen, J. [2007b]: Efficient algorithms for short and fast repeater trees. II. Buffering. Technical Report 07978, Research Institute for Discrete Mathematics, University of Bonn, 2007.

Berridge, R., Averill, R. M., Barish, A. E., Bowen, M. A., Camporese, P. J., DiLullo, J., Dudley, P. E., Keinert, J., Lewis, D. W., and Morel, R. D. [2007]: IBM POWER6 microprocessor physical design and design methodology. IBM Journal of Research and Development 51 (2007), 685–714, `http://www.research.ibm.com/journal/rd/516/berridge.html`.

Bhardwaj, S., and Vrudhula, S. [2008]: Leakage Minimization of Digital Circuits Using Gate Sizing in the Presence of Process Variations. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 27 (2008), 445–455.

Blaauw, D., Zolotov, V., Sundareswaran, S., Oh, C., and Panda, R. [2000]: Slope Propagation in static timing analysis. Proceedings of the IEEE International Conference on Computer-Aided Design, 338–343, 2000.

Boyd, S. P., Kim, S.-J., Patil, D. D., and Horowitz, M. A. [2005]: Digital Circuit Optimization via Geometric Programming. Operations Research 53 (2005), 899–932.

Brenner, U., and Rohe, A. [2003]: An effective congestion-driven placement framework. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 22 (2003), 387–394.

Brenner, U., and Struzyna, M. [2005]: Faster and better global placement by a new transportation algorithm. Proceedings of the 42nd Annual Conference on Design Automation, 591–596, 2005.

Brenner, U., Struzyna, M., and Vygen, J. [2008]: BonnPlace: Placement of Leading-Edge Chips by Advanced Combinatorial Algorithms. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (accepted for appearance) (2008).

Brenner, U., and Vygen, J. [2004]: Legalizing a placement with minimum total movement. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 23 (2004), 1597–1613.

Burger, D., Keckler, S. W., and McKinley, K. S. e. a. [2004]: Scaling to the End of Silicon with EDGE Architectures. IEEE Compute 37 (2004), 44–55.

Chao, T.-H., Hsu, Y.-C., and Ho, J. M. [1992]: Zero Skew Clock Net Routing. Proceedings ACM/IEEE Design Automation Conference, 518–523, 1992.

Charnes, A. [1952]: Optimality and degeneracy in linear programming. Econometrica 20 (1952), 160–170.

Chen, C.-P., Chu, C. C. N., and Wong, D. F. [1999]: Fast and exact simultaneous gate and wire sizing by Lagrangian relaxation. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 18 (1999), 1014–1025.

Chen, H. Y., and Kang, S. M. [1991]: iCOACH: a circuit optimization aid for CMOS high-performance circuits. Integration, the VLSI Journal 10 (1991), 185–212.

Chen, W., Hseih, C.-T., and Pedram, M. [2000]: Simultaneous Gate Sizing and Placement. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 19 (2000), 206–214.

Chou, H., Wang, Y.-H., and Chen, C. C.-P. [2005]: Fast and effective gate-sizing with multiple-Vt assignment using generalized Lagrangian Relaxation. Proceedings of the 2005 Conference on Asia South Pacific Design Automation, ACM, New York, NY, USA, 381–386, 2005.

Chu, C. C. N. [2004]: FLUTE: Fast Lookup Table Based Wirelength Estimation Technique. Proceedings of the IEEE International Conference on Computer-Aided Design, 696–701, 2004.

Chu, C. C. N., and Wong, D. F. [1997]: Closed form solution to simultaneous buffer insertion/sizing and wire sizing. Proceedings of the ACM International Symposium on Physical Design, 192–197, 1997.

Cochet-Terrasson, J., Cohen, G., Gaubert, S., McGettrick, M., and Quadrat, J. [1998]: Numerical computation of spectral elements in max-plus algebra. IFAC Conf. on Syst. Structure and Control, 1998.

Cong, J. [2002]: Timing closure based on physical hierarchy. Proceedings of the international symposium on Physical design, 170–174, 2002.

Cong, J., Leung, K.-S., and Zhou, D. [1993]: Performance-driven interconnect design based on distributed RC delay model. Proceedings of the 30th International Conference on Design Automation, 606–611, 1993.

Cong, J., and Yuan, X. [2000]: Routing tree construction under fixed buffer locations. Proceedings ACM/IEEE Design Automation Conference, 379–384, 2000.

Conn, A. R., Coulman, P. K., Haring, R. A., Morrill, G. L., Visweswariah, C., and Wu, C. W. [1998]: JiffyTune: circuit optimization using time-domain sensitivities. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 17 (1998), 1292–1309.

Conn, A. R., Elfadel, I. M., W. W. Molzen, J., O'Brien, P. R., Strenski, P. N., Visweswariah, C., and Whan, C. B. [1999]: Gradient-based optimization of custom circuits using a static-timing formulation. Proceedings of the 36th ACM/IEEE Conference on Design Automation, 452–459, 1999.

Dai, Z.-J., and Asada, K. [1989]: MOSIZ: a two-step transistor sizing algorithm based on optimal timing assignment method for multi-stage complex gates. IEEE Custom Integrated Circuits Conference (1989), 17.3.1–17.3.4.

Dasdan, A., Irani, S., and Gupta, R. K. [1999]: Efficient Algorithms for Optimum Cycle Mean and Optimum Cost to Time Ratio Problems. Proceedings ACM/IEEE Design Automation Conference, 37–42, 1999.

De, P., Dunne, E. J., Ghosh, J. B., and Wells, C. E. [1997]: Complexity of the Discrete Time-Cost Tradeoff Problem for Project Networks. Operations Research 45 (1997), 302–306.

Dechu, S., Shen, Z. C., and Chu, C. C. N. [2005]: An efficient routing tree construction algorithm with buffer insertion, wire sizing, and obstacle considerations. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 24 (2005), 600–608.

Deineko, V. G., and Woeginger, G. J. [2001]: Hardness of approximation of the discrete time-cost tradeoff problem. Operations Research Letters 29 (2001), 207–210.

Deokar, R. B., and Sapatnekar, S. S. [1995]: A Graph–Theoretic Approach to Clock Skew Optimization. Proceedings of the IEEE International Symposium on Circuits and Systems, 407–410, 1995.

Dhar, S., and Franklin, M. A. [1991]: Optimum buffer circuits for driving long uniform lines. IEEE Journal of Solid-State Circuits 26 (1991), 32–40.

Dijkstra, E. W. [1959]: A Note on two Problems in connexion with graphs. Numerische Mathematik 1 (1959), 269–271.

Dobhal, A., Khandelwal, V., Davoodi, A., and Srivastava, A. [2007]: Variability Driven Joint Leakage-Delay Optimization Through Gate Sizing with Provable Convergence. 20th International Conference on VLSI Design, 571–576, 2007.

Elmore, W. C. [1948]: The transient response of damped linear networks with particular regard to wide-band amplifiers. Journal of Applied Physics 19 (1948), 55–63.

Engel, K., Kalinowski, T., Labahn, R., Sill, F., and Timmermann, D. [2006]: Algorithms for Leakage Reduction with Dual Threshold Design Techniques. International Symposium on System-on-Chip, 1–4, 2006.

Fishburn, J. P. [1990]: Clock skew optimization. IEEE Transactions on Computers 39 (1990), 945–951.

Fishburn, J. P., and Dunlop, A. E. [1985]: TILOS: A posynomial programming approach to transistor sizing. Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, 326–328, 1985.

Fulkerson, D. R. [1961]: A network flow computation for project cost curves. Management Science 7 (1961), 167–178.

Gao, F., and Hayes, J. P. [2005]: Total power reduction in CMOS circuits via gate sizing and multiple threshold voltages. Proceedings of the 42nd Annual Conference on Design Automation, 31–36, 2005.

Garey, M. R., and Johnson, D. S. [1977]: The rectilinear Steiner tree problem is *NP*-complete. SIAM Journal on Applied Mathematics 32 (1977), 826–834.

Ghiasi, S., Bozorgzadeh, E., Huang, P.-K., Jafari, R., and Sarrafzadeh, M. [2006]: A Unified Theory of Timing Budget Management. IEEE Transactions on computer-aided design of integrated circuits and systems 25 (2006), 2364–2375.

van Ginneken, L. P. P. P. [1990]: Buffer placement in distributed RC-tree networks for minimal Elmore delay. Proceedings of the IEEE International Symposium of Circuits and Systems, 865–868, 1990.

Goldberg, A. V., and Rao, S. [1998]: Beyond the flow decomposition barrier. Journal of the ACM 45 (1998), 783–797.

Golumbic, M. C. [1976]: Combinatorial Merging. IEEE Trans. Comput. 25 (1976), 1164–1167.

Hanan, M. [1966]: On Steiner's Problem with Rectilinear Distance. SIAM Journal on Applied Mathematics 14 (1966), 255–265.

Hathaway, D., Alvarez, J. P., and Belkbale, K. P. [1997]: Network timing analysis method which eliminates timing variations between signals traversing a common circuit path. United States patent $5,636,372$, 1997.

Held, S. [2001]: Algorithmen für Potential-Balancierungs-Probleme und Anwendungen im VLSI-Design. Diplomarbeit, University of Bonn, 2001.

Held, S., Korte, B., Maßberg, J., Ringe, M., and Vygen, J. [2003]: Clock Scheduling and Clocktree Construction for High Performance ASICs. Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, 232–239, 2003.

Hentschke, R. F., Narasimham, J., Johann, M. O., and da Luz Reis, R. A. [2007]: Maze routing steiner trees with effective critical sink optimization. ISPD, 135–142, 2007.

Heusler, L. S., and Fichtner, W. [1991]: Transistor sizing for large combinational digital CMOS circuits. Integration, the VLSI Journal 10 (1991), 155–168.

Hitchcock, R. B., Smith, G. L., and Cheng, D. D. [1982]: Timing Analysis of Computer Hardware. IBM Journal of Research and Development 26 (1982), 100–105.

Hoffman, A., and Kruskal, J. B. [1956]: Integral boundary points of convex polyhedra. Linear Inequalities and Related Systems, Princeton University Press, 38, 223–246, 1956.

Hrkic, M., and Lillis, J. [2002]: S-Tree: a technique for buffered routing tree synthesis. Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, 578–583, 2002.

Hrkic, M., and Lillis, J. [2003]: Buffer tree synthesis with consideration of temporal locality, sink polarity requirements, solution cost, congestion, and blockages. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 22 (2003), 481–491.

Hu, S., Alpert, C. J., Hu, J., Karandikar, S., Li, Z., Shi, W., and Sze, C. N. [2006]: Fast algorithms for slew constrained minimum cost buffering. Proceedings of the 43rd Annual Conference on Design Automation, 308–313, 2006.

Hu, S., Ketkar, M., and Hu, J. [2007]: Gate sizing for cell library-based designs. Proceedings of the 44th Annual Conference on Design Automation, 847–852, 2007.

Hwang, F. K. [1976]: On steiner minimal trees with rectilinear distance. SIAM Journal of Applied Mathematics 30 (1976), 104–114.

Karmarkar, N. [1984]: A new polynomial-time algorithm for linear programming. Proceedings of the sixteenth annual ACM symposium on Theory of computing, 302–311, 1984.

Karp, R. M. [1972]: Reducibility among combinatorial problems. R. E. Miller, J. W. T., ed., Complexity of Computer Computations, Plenum Press, New York, 85–103, 1972.

Karp, R. M. [1978]: A Characterization of the Minimum Mean Cycle in a Digraph. Discrete Mathematics 23 (1978), 309–311.

Kelley Jr., J. E. [1961]: Critical-Path Planning and Scheduling: Mathematical Basis. Operations Research 9 (1961), 296–320.

Khandelwal, V., and Srivastava, A. [2008]: Variability-Driven Formulation for Simultaneous Gate Sizing and Postsilicon Tunability Allocation. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 27 (2008), 610–620.

King, V., Rao, S., and Tarjan, R. [1994]: A faster deterministic maximum flow algorithm. J. Algorithms 17 (1994), 447–474.

Kleff, A. [2008]: Iterative Balancierungsverfahren und ihre Anwendung für das Clock-Skew-Scheduling. Diplomarbeit, University of Bonn, 2008.

Kong, T. T. [2002]: A novel net weighting algorithm for timing-driven placement. Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, 172–176, 2002.

Korte, B., and Vygen, J. [2008]: Combinatorial Optimization: Theory and Algorithms. Fourth Edition. Algorithms and Combinatorics, Springer, Berlin, Heidelberg, New York, 2008.

Kourtev, I. S., and Friedman, E. G. [1999]: A Quadratic Programming Approach to Clock Skew Scheduling for Reduced Sensitivity to Process Parameter Variations. Proceedings of the IEEE International ASIC/SOC Conference, 210–215, 1999.

Kraft, L. G. [1949]: A device for quantizing grouping and coding amplitude modulated pulses. Master thesis, EE Dept., MIT, Cambridge, 1949.

Kursun, E., Ghiasi, S., and Sarrafzadeh, M. [2004]: Transistor Level Budgeting for Power Optimization. Proceedings of the 5th International Symposium on Quality Electronic Design (2004), 116–121.

Langkau, K. [2000]: Gate-Sizing im VLSI-Design. Diplomarbeit, University of Bonn, 2000.

Lawler, E. L. [1976]: Combinatorial Optimization: Networks And Matroids. Holt, Rinehart And Winston, 1976.

Levner, E. V., and Nemirovsky, A. S. [1994]: A network flow algorithm for just-in-time project scheduling. European Journal of Operational Research 79 (1994), 167–175.

Li, Z., and Shi, W. [2006a]: An $O(bn^2)$ time algorithm for optimal buffer insertion with b buffer types. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 25 (2006a).

Li, Z., and Shi, W. [2006b]: An O(mn) time algorithm for optimal buffer insertion of nets with m sinks. Proceedings of the 2006 Conference on Asia South Pacific Design Automation, 320–325, 2006.

Lillis, J., Cheng, C.-K., and Lin, T.-T. Y. [1995]: Optimal wire sizing and buffer insertion for low power and a generalized delay model. Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, 138–143, 1995.

Lillis, J., Cheng, C.-K., Lin, T.-T. Y., and Ho, C.-Y. [1996]: New performance driven routing techniques with explicit area/delay tradeoff and simultaneous wire sizing. Proceedings ACM/IEEE Design Automation Conference, 395–400, 1996.

Mani, M., Devgan, A., Orshansky, M., and Zhan, Y. [2007]: A Statistical Algorithm for Power- and Timing-Limited Parametric Yield Optimization of Large Integrated Circuits. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 26 (2007), 1790–1802.

Marple, D. P. [1986]: Performance optimization of digital VLSI circuits. Technical Report CSL-TR-86-308, Stanford University, 1986.

Maßberg, J., and Rautenbach, D. [2007]: Binary trees with choosable edge lengths. Technical Report 07973, Research Institute for Discrete Mathematics, University of Bonn, 2007.

Maßberg, J., and Vygen, J. [2005]: Approximation Algorithms for Network Design and Facility Location with Service Capacities. APPROX 2005, 158–169, 2005.

Maßberg, J., and Vygen, J. [2008]: Approximation algorithms for a facility location problem with service capacities. ACM Transactions on Algorithms, to appear. (Preliminary version in APPROX 2005) (2008).

Megiddo, N. [1983]: Applying parallel computation algorithms in the design of serial algorithms. Journal of the ACM 30 (1983), 852–865.

Mo, Y.-Y., and Chu, C. C. N. [2000]: A hybrid dynamic / quadratic programming algorithm for interconnect tree optimization. Proceedings of the ACM International Symposium on Physical Design, 134–139, 2000.

Müller, D. [2006]: Optimizing Yield in Global Routing. Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, 480–486, 2006.

Müller-Hannemann, M., and Zimmermann, U. [2003]: Slack optimization of timing-critical nets. Algorithms – Proceedings of the 11th Annual European Symposium on Algorithms (ESA 2003), Springer, Berlin, 727–739, 2003.

Nagel, L. W., and Pederson, D. O. [1973]: SPICE (Simulation Program with Integrated Circuit Emphasis). Technical Report Memorandum No. ERL-M382, University of California, Berkeley, 1973.

Okamoto, T., and Cong, J. [1996]: Buffered Steiner tree construction with wire sizing for interconnect layout optimization. Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, 44–49, 1996.

Orlin, J. B. [1993]: A faster strongly polynomial minimum cost flow algorithm. Operations Research 41 (1993), 338–350.

Osborne, E. E. [1960]: On Pre-Conditioning of Matrices. Journal of the ACM 7 (1960), 338–345.

Pan, M., Chu, C., and Patra, P. [2007]: A Novel Performance-Driven Topology Design Algorithm. Proceedings of the 2007 Conference on Asia South Pacific Design Automation, 145–150, 2007.

Phillips, J. S., and Dessouky, M. I. [1977]: Solving the project time/cost tradeoff problem using the minimal cut concept. Management Science 24 (1977), 393–400.

Radunović, B., and Le Boudec, J.-Y. [2007]: A unified framework for max-min and min-max fairness with applications. IEEE/ACM Transactions on Networking 15 (2007), 1073–1083.

Rao, R. R., Devgan, A., Blaauw, D., and Sylvester, D. [2004]: Parametric yield estimation considering leakage variability. Proceedings of the 41st Design Automation Conference, 442–447, 2004.

Ratzlaff, C. L., Gopal, N., and Pillage, L. T. [1991]: RICE: Rapid interconnect circuit evaluator. Proceedings of the 28th Annual Conference on Design Automation, 555–560, 1991.

Rautenbach, D., and Szegedy, C. [2007]: A Class of Problems for which Cyclic Relaxation converges linearly. Computational Optimization and Applications (accepted for appearance) (2007).

Ravindran, K., Kuehlmann, A., and Sentovich, E. [2003]: Multi-Domain Clock Skew Scheduling. Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, 801–808, 2003.

Ren, H., Pan, D. Z., and Kung, D. S. [2005]: Sensitivity guided net weighting for placement-driven synthesis. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 24 (2005), 711–721.

Rohe, A. [2002]: Sequential and Parallel Algorithms for Local Routing. Dissertation, University of Bonn, 2002.

Rosdi, B. A., and Takahashi, A. [2004]: Reduction on the usage of intermediate registers for pipelined circuits. Proceedings Workshop on Synthesis and System Integration of Mixed Systems, 333–338, 2004.

Sakallah, K. A., Mudge, T. N., and Olukotun, O. A. [1990]: checkTc and minTc: Timing verification and optimal clocking of digital circuits. Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, 552–555, 1990.

Sapatnekar, S. S. [2004]: Timing. Kluwer Academic Publishers, Boston, Dordrecht, New York, London, 2004.

Sapatnekar, S. S., Rao, V. B., Vaidya, P. M., and Kang, S.-M. [1993]: An Exact Solution to the Transistor Sizing Problem for CMOS Circuits Using Convex Optimization. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 12 (1993), 1621–1634.

Saxena, P., Menezes, N., Cocchini, P., and Kirkpatrick, D. [2003]: The scaling challenge: can correct-by-construction design help? Proceedings of the ACM International Symposium on Physical Design, 51–58, 2003.

Schietke, J. [1999]: Timing-Optimierung beim physikalischen Layout von nicht-hierarchischen Designs hochintegrierter Logikchips. Dissertation, University of Bonn, 1999.

Schmedding, R. [2007]: Time-Cost-Tradeoff Probleme und eine Anwendung in der Timing-Optimierung im VLSI-Design. Diplomarbeit, University of Bonn, 2007.

Schneider, H., and Schneider, M. [1991]: Max-balancing weighted directed graphs and matrix scaling. Mathematics of Operations Research 16 (1991), 208–220.

Shah, S., Srivastava, A., Sharma, D., Sylvester, D., Blaauw, D., and Zolotov, V. [2005]: Discrete Vt assignment and gate sizing using a self-snapping continuous formulation. Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, 705–712, 2005.

Shenoy, N., Brayton, R. K., and Sangiovanni-Vincentelli, A. L. [1992]: Graph algorithms for clock schedule optimization. Proceedings of the IEEE/ACM international conference on Computer-Aided design, 132–136, 1992.

Shi, W., and Li, Z. [2005]: A fast algorithm for optimal buffer insertion. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 26 (2005), 879–891.

Singh, J., Luo, Z.-Q., and Sapatnekar, S. S. [2008]: A Geometric Programming-Based Worst Case Gate Sizing Method Incorporating Spatial Correlation. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 27 (2008), 295–308.

Sinha, D., Shenoy, N. V., and Zhou, H. [2006]: Statistical Timing Yield Optimization by Gate Sizing. IEEE Transactions on VLSI Systems 14 (2006), 1140–1146.

Skutella, M. [1998]: Approximation algorithms for the discrete time-cost tradeoff problem. Mathematics of Operations Research 23 (1998), 909–929.

Sundararajan, V., Sapatnekar, S. S., and Parhi, K. K. [2002]: Fast and exact transistor sizing based on iterative relaxation. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 21 (2002), 568 – 581.

Szegedy, C. [2005a]: personal communication, 2005.

Szegedy, C. [2005b]: Some Applications of the weighted combinatorial Laplacian. Dissertation, University of Bonn, 2005.

Szymanski, T. [1992]: Computing Optimal Clock Schedules. Proceedings ACM/IEEE Design Automation Conference, 399–404, 1992.

Takahashi, A. [2006]: Practical Fast Clock-Schedule Design Algorithms. IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences E89-A (2006), 1005–1011.

Takahashi, H., and Matsuyama, A. [1980]: An approximate solution for the Steiner problem in graphs. Math. Japonica 24 (1980), 573–577.

Tennakoon, H., and Sechen, C. [2002]: Gate sizing using Lagrangian relaxation combined with a fast gradient-based pre-processing step. Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, 395–402, 2002.

Trevillyan, L., Kung, D., Puri, R., Reddy, L. N., and Kazda, M. A. [2004]: An integrated environment for technology closure of deep-submicron IC designs. IEEE Design & Test of Computers, 21 (2004), 14–22.

Vujkovic, M., Wadkins, D., Swartz, B., and Sechen, C. [2004]: Efficient timing closure without timing driven placement and routing. Proceedings ACM/IEEE Design Automation Conference, 268–273, 2004.

Vygen, J. [2001]: Theory of VLSI Layout. Habilitation thesis, University of Bonn, 2001.

Vygen, J. [2004]: Near Optimum Global Routing with Coupling, Delay Bounds, and Power Consumption. Proceedings of the 10th International IPCO Conference, 308–324, 2004.

Vygen, J. [2006]: Slack in static timing analysis. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 25 (2006), 1876–1885.

Vygen, J. [2007]: New theoretical results on quadratic placement. Integration, the VLSI Journal 40 (2007), 305–314.

Werber, J., Rautenbach, D., and Szegedy, C. [2007]: Timing optimization by restructuring long combinatorial paths. Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, 536–543, 2007.

Young, N. E., Tarjan, R. E., and Orlin, J. B. [1991]: Faster Parametric Shortest Path and Minimum-Balance Algorithms. Networks 21 (1991), 205–221.

Zejda, J., and Frain, P. [2002]: General framework for removal of clock network pessimism. Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, 632–639, 2002.

Summary

Timing closure is a major challenge to the physical design of a computer chip. In this thesis, we proposed new algorithms for the key tasks of performance optimization, namely repeater tree construction; circuit sizing; clock skew scheduling; threshold voltage optimization and plane assignment. Furthermore, a new program flow for timing closure was developed that integrates these algorithms with placement and clocktree construction. The algorithms and the program flow are designed to handle the largest real-world chip instances.

An algorithm for computing repeater tree topologies, which are later filled with repeaters, is presented at the beginning of this thesis. To this end, we propose a new delay model for topologies that not only accounts for the path lengths in the tree, as existing approaches do, but also for the number of bifurcations on a path, which introduce extra capacitance and thereby delay. At each bifurcation, a fixed total bifurcation delay has to be distributed to the two branching edges. The freedom to distribute the bifurcation cost among the two branches simulates the possibility of shielding the capacitance of the less critical branch by inserting a small repeater.

In the extreme cases of pure power optimization and pure delay optimization we have shown that the optimum topologies regarding our delay model are minimum Steiner trees and alphabetic code trees with the shortest possible path lengths. We presented a new, extremely fast algorithm that scales seamlessly between the two opposite objectives. It inserts the sinks in the order of decreasing timing criticality, such that the worst slack at the root is maximized with respect to the current insertion. Very large instances are preclustered by a fast facility location approach. Although the algorithm has a cubic worst case running time, it is indeed very fast. Several million topologies are computed within a few minutes. For special cases, we could prove the optimality of our algorithm. The quality of the generated trees in practice is demonstrated by comprehensive experimental results and comparison with lower bounds on the wire length, the worst path delays, and the number of repeaters that are inserted later. It turns out that constructing topologies for optimum performance not only improves timing, but also saves power, as the number of shielding repeaters can be kept low. This is an additional justification of our approach, which keeps the number of bifurcations on the critical path particularly small, where existing algorithms for repeater insertion tend to insert shielding repeaters.

The task of circuit sizing is to assign millions of small elementary logic circuits to elements from a discrete set of logically equivalent, predefined physical layouts such that power consumption is minimized and all signal paths are sufficiently fast. Existing approaches assume simplified delay models and consider the continuous

relaxation of the discrete problem. Though the resulting geometric programs can be solved optimally, they can either solve only relatively small instances or require very simple delay approximations. Afterwards, the results have to be mapped to discrete layouts entailing potential delay degradations and electrical violations.

In this thesis we developed a fast heuristic approach for global circuit sizing, followed by a local search into a local optimum. Our algorithms use the existing discrete layout choices and accurate delay models with full propagation of signal slews. The global approach iteratively assigns slew targets to all source pins of the chip and chooses a discrete layout of minimum size preserving the slew targets. In each iteration, the slew target of the source pin of each net is refined, based on global and local criticality. The source pin is globally critical if it has a signal with negative slack. In this case we consider the local criticality, which is defined as the difference between the worst slack of a signal at the source pin and the worst slack of a signal at its predecessor source pins. If the local criticality is zero, the current pin must be located on a most critical path through a predecessor and the slew target is tightened. Otherwise, there is a more critical path through an input driver, which can be sped up by downsizing the current circuit, indirectly by relaxing the slew target of the current source pin. If the current pin has a positive worst slack making it globally uncritical, its slew target is relaxed to reduce the power consumption. We proposed a method to compute a lower bound on the delay of the most critical paths. In our comprehensive experiments on real instances, we demonstrate that the worst path delay is within 7% of its lower bound on average after a few iterations. The subsequent local search reduces this gap to 2% on average. The local search iterates cyclically over the circuits on the critical path and assigns them to the local optimum layout. As the global algorithm works without extensive incremental timing updates, it is capable of sizing designs with more than 5.7 million circuits within 3 hours on modern personal computers.

Clock skew scheduling for optimizing the switching times of the registers can significantly improve the performance. We developed the first algorithm with a strongly polynomial running time for the cycle time minimization in the presence of different cycle times and multi-cycle paths. It turns out that this problem can be solved by a minimum ratio cycle computation in a balance graph with edge costs and weights. Such graph-based models cause a lot of overhead, especially in terms of memory consumption and preprocessing time. In fact, they are unsuitable for large chip instances. In practice, iterative local scheduling methods are much more efficient. These methods iterate cyclically over all registers and schedule them to their local optimum based in the worst slacks of incoming and outgoing signals. However, there has not been much mathematical insight into these methods. We have proven that the iterative method maximizes the worst slack, even when restricting the feasible schedule of the registers to certain time intervals, which is an indirect mean to control the power consumption of the clocktrees. Furthermore, we enhanced the iterative local approach to become a hybrid model that constructs the balance graph incrementally on the most critical paths. The additional global constraint edges allow us to hide the most critical paths that cannot be further improved.

Thereby, the hybrid model yields a lexicographically optimum slack distribution. However, the experimental results show that, in practice, the improvements of the optimum method over the purely iterative method are hardly traceable. The results therefore justify using the faster and very memory efficient iterative local method in practice.

The clock skew scheduling problem is then generalized to allow for simultaneous data path optimization. In fact, this is a time-cost tradeoff problem. On acyclic graphs, which would correspond to a fixed clock schedule, the time-cost tradeoff problem is well understood. We developed the first combinatorial algorithm for computing time-cost tradeoff curves in graphs that may contain cycles. Starting from the lowest-cost solution, the algorithm iteratively computes a descent direction for edge delays and node potentials by a minimum cost flow computation on the subgraph of most critical edges and vertices. The maximum feasible step length for the current descent direction is then determined by a minimum ratio cycle computation. Although the time-cost tradeoff curve, and thereby the algorithm, may have an exponential size and running time, it is still efficient if the critical subgraph, where the minimum cost flow is computed, is small in comparison to the overall graph. This approach can be used in chip design for optimization tasks, where a layout change has only a local impact on the delay of a single net or circuit, for example plane assignment and repeater tree construction. We apply this algorithm as a heuristic for threshold voltage optimization, with non-linear delays and discrete threshold voltage levels. For each circuit on a most critical path we compute the delay and power changes when switching to the neighboring threshold voltage levels. The difference quotients are used as the slopes of the piecewise linear functions in the linear relaxation. And the minimum cost flow result gives a hint for speeding up and delaying circuits.

Finally, the optimization routines were combined into a timing closure flow. Here, the placement of the circuits is alternated with global performance optimization by repeater tree construction and circuit sizing. Netweights are used to keep critical nets short during placement. When the global placement is determined, the performance is improved further by focusing on the most critical paths. Repeater trees are rebuilt targeting best performance and circuit sizing is restricted to local search. Furthermore, threshold voltages and assignment of nets to higher, and therefore faster, wiring planes are optimized. In the end, the clock schedule is optimized and clocktrees are inserted. Computational results of the design flow are obtained on real-world computer chips. The largest chip instances with more that 5 million circuits can be designed in about 30 hours. We have been able to reduce the running time of global performance optimization, which is performed between two global placement runs, from 3 days by a widely used industrial tool to less than 6 hours, allowing for more placement iterations and a higher quality of the placement. Today, the program flow is used by IBM to design many of the most challenging computer chips.