Cycle Time and Slack Optimization for VLSI–Chips

C. Albrecht B. Korte J. Schietke J. Vygen

Research Institute for Discrete Mathematics, University of Bonn

Lennéstr. 2, 53113 Bonn, Germany

Abstract

We consider the problem of finding an optimal clock schedule, i.e. optimal arrival times for clock signals at latches of a VLSI chip. We describe a general model which includes all previously considered models. Then we show how to optimize the cycle time and optimally balance slacks on data paths and on clocktree paths.

The problem of finding a clock schedule with the optimum cycle time was solved before, either by linear programming or by binary search, using a test for negative circuits in a digraph as a subroutine. We show for the first time that a direct combinatorial algorithm solves this problem optimally. Incidentally, this yields a new efficient method for timing analysis with transparent latches.

Moreover, we extend this algorithm to the slack balancing problem: To make the chip less sensitive to routing detours, process variations and manufacturing skew it is desirable to have as few critical paths as possible. We show how to find the clock schedule with minimum number of critical paths (optimum slack distribution) in a well–defined sense.

Rather than fixed clock arrival times we show how to obtain as large as possible intervals for the clock arrival times. This can be considered as slack on clocktree paths. Indeed, we can find the global optimum of simultaneous optimization of slacks on all data paths and clocktree paths.

All the above is done by very efficient network optimization algorithms, based on parametric shortest paths. Our computational results with recent IBM processor chips show that the number of critical paths decreases dramatically, in addition to a considerable improvement of the cycle time. The running times are reasonable even for the largest designs.

1 Introduction

Optimization of the cycle time is a main goal in VLSI chip design. All computations on a chip are synchronized by storage elements which receive a clock signal periodically. The period is called the cycle time and is denoted by T in this paper. We solve the problem of finding an optimal clocking schedule to minimize T.

In Section 2 we develop a very general model and show how to model various constraints. In Section 3 it is shown that the problem of computing the optimum cycle time reduces to a parametric shortest path problem. This can be solved by the algorithm of Young, Tarjan and Orlin [14], of which an outline is given. Previously the problem has been solved either by linear programming [4, 8] or by a binary search approach [11, 10, 12, 2, 7]; both methods are less efficient.

In Section 4 and 5 we develop an algorithm which takes additional objectives into account. Among all solutions with the optimum cycle time we find the one with optimally balanced slacks, for data paths as well as for clocktree paths. This is the first algorithm which computes the optimum cycle time and slacks and runs fast enough even for the largest designs.

In Section 4 we first consider slacks on data paths. We obtain a solution where as few as necessary data paths are critical. The chip becomes less sensitive to routing detours, process variations and manufacturing skew. Moreover, if one tries to optimize the cycle time further only few data paths have to be considered.

In Section 5 it is shown how this algorithm can be modified to increase the slacks on the clocktree paths. These slacks can be used in the design of the clocktree: For most latches it is not necessary to fix the arrival time of the clock signal, instead a maximal interval of feasible arrival times is computed.

Computational results with recent IBM processor chips in Section 6 demonstrate the power of our method. The cycle time of the chips is improved by 2.5 up to 5.5 percent. Moreover, the number of critical data and clocktree paths (with zero or small positive slack) decreases substantially. The running times are acceptable even for the largest designs.

2 Problem formulation

Let S be the set of simple storage elements, each storing one bit. The clock input of $s \in S$ has value 1 from time a_s to b_s, then again from $a_s + T$ to $b_s + T$, from $a_s + 2T$ to $b_s + 2T$ and so on. In the remaining time it has the value 0. When the clock input has the value 1, the storage element is open, i.e. it stores the value currently seen at the data input. When the clock input has value 0, the stored bit remains unchanged. (Sometimes the roles of 0 and 1 are interchanged, but this does not matter.) At any time, the stored bit is available at the data output for subsequent computations.

0-7803-5832-X /99/$10.00

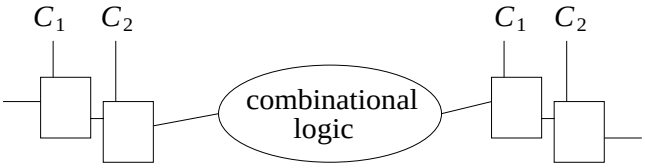

Figure 1: Two master–slave latches with clock signals C_1 and C_2 and a data path in between

All types of storage elements can be handled with this model. For edge–triggered flip–flops we simply set $a_s = b_s$. Master–slave latches can be represented by two simple storage elements s and s' (cf. Figure 1).

Now we may shift the clock input for each $s \in S$ by some value y_s, meaning that the clock input has the value 1 in the time intervals $[a_s + y_s, b_s + y_s]$, $[a_s + y_s + T, b_s + y_s + T]$, $[a_s + y_s + 2T, b_s + y_s + 2T]$ and so on. We assume that the length of the interval $b_s - a_s$ cannot be changed.

Since the shifting times y_s have to be realized by a clocktree it is reasonable to impose a lower bound l_s and an upper bound u_s on y_s for each storage element s:

$$l_s \le y_s \le u_s \qquad (s \in S). \tag{1}$$

We introduce a second variable x_s for the time when the data signal is valid at the data input of s. We must have

$$a_s + y_s \le x_s \le b_s + y_s \qquad (s \in S), \tag{2}$$

and the data signal should remain valid within the whole interval $[x_s, b_s + y_s]$. For primary inputs and outputs v we set $y_v = 0$ and $a_v = b_v = x_v$ (this arrival time resp. required arrival time is part of the specification of the chip).

A data signal might encounter more than one storage element per cycle (for example in designs using transparent latches). So for each data path from v to w it has to be specified whether a signal starting at s within the time interval $[a_v, b_v]$ must arrive before b_w (i.e. in the same cycle) or before $b_w + T$ (i.e. in the next cycle). In the first case we set $\zeta_{vw} := 0$, in the second case we set $\zeta_{vw} := 1$. Then the late mode constraints read as follows:

$$x_v + t_{vw}^{max} \le x_w + \zeta_{vw} T \qquad ((v,w) \in E(G)), \tag{3}$$

where G is the digraph containing an edge for each pair of latches connected by a data path. For the early mode constraints we have to take the whole intervals into account where the storage elements are open:

$$a_v + y_v + t_{vw}^{min} \ge b_w + y_w + (\zeta_{vw} - 1)T \quad ((v,w) \in E(G)). \tag{4}$$

This describes a quite general model. In the next sections we show how to solve the above–defined linear program efficiently. We show how to minimize the cycle time T and how to distribute slacks on data and clock paths optimally.

Our model includes the previous models for cycle time optimization [4, 8, 11, 10, 2, 7, 13]. It is quite flexible: additional technical constraints can easily be incorporated. For example one can model dynamic circuits by constraints of the same type as above in a straightforward way. Some other constraints (end–of–cycle test, simultaneous clock signals for registers) will be mentioned in Section 6. Another advantage of this model is that there is no need to distinguish between master–slave latches, simple level–sensitive latches and edge–triggering flip–flops.

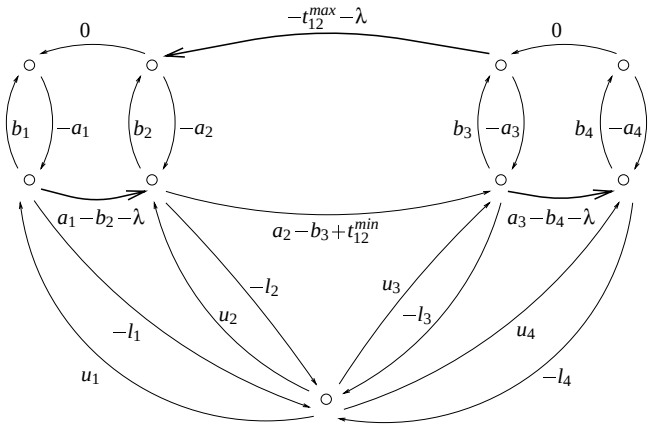

Figure 2: The vertices and edges in G' for two master–slave latches and a data path in between. The bold edges are parameterized.

3 Computing the optimal cycle time

Observe that each of the inequality constraints (1), (2), (3) and (4) has one or two x– or y–variables and in addition possibly the special variable T. If a constraint has two x– or y–variables, they have opposite sign. To have exactly two variables per inequality we introduce an artificial variable z_0 which we assume to have value zero. For technical reasons we substitute $\lambda := -T$ and obtain a linear program of the following very special type:

$$\max\ \lambda \qquad \text{subject to}$$

$$z_i + c_{ij} \ge z_j \qquad \text{for } (i,j) \in E_1, \tag{5}$$

$$z_i + c_{ij} - \lambda \ge z_j \qquad \text{for } (i,j) \in E_2, \tag{6}$$

where λ and $z_0, z_1, z_2, \ldots, z_n$ are variables, the c_{ij} are constants and $E_1, E_2 \subseteq \{1,\ldots,n\} \times \{1,\ldots,n\}$. Each constraint (1) corresponds to two elements $(i,0), (0,i)$ of E_1, each constraint (2) corresponds to one element of E_1, and each of the constraints (3) and (4) corresponds to an element of E_1 or E_2, depending on the ζ–constant. Note that the assumption that $z_0 = 0$ causes no loss of generality since adding a constant to all variables z_i does not affect feasibility.

Now we translate our linear optimization problem to a network problem. Given the above linear program we construct a directed graph $G' = (V, E)$ as follows: For each variable z_i there is a vertex v_i. For a constraint of type (5) we have a directed edge from vertex v_i to vertex v_j of cost c_{ij}. For a constraint of type (6) we also have an edge from v_i to v_j, but the cost is $c_{ij} - \lambda$. Such an edge is called parameterized: the cost of the edge depends on the parameter λ.

As an example, Figure 2 shows the vertices and edges of G' for two master–slave latches with a data path in between (from left to right; cf. Figure 1). Each master–slave latch consists of two simple latches, and for each simple latch we have an x–variable (on top) and a y–variable. So there are four variables (vertices) for each

master–slave latch; the artificial variable z_0 is represented by the vertex at the bottom. There are eight edges for constraints of type (1), eight for type (2), three edges for type (3) and three for type (4). Three of the edges are parameterized.

We are looking for the maximum value for λ such that values for the variables z_i exist fulfilling all constraints. It is easy to see [11, 10, 12, 2] that such values exist if and only if the digraph G' contains no directed circuit of negative cost (negative circuit, for short). The problem might be infeasible in some cases. Our algorithm detects infeasibility and returns the negative circuit(s) causing the problem. Usually one can cope with this by omitting some early mode constraints: these can be met by inserting buffers (increasing the delay of the path). In fact it is usually not good to take all early mode constraints into account because this might increase the optimum value of the LP (hence the cycle time); one usually prefers inserting a buffer. As we describe in Section 6 we incorporate early mode constraints only after having determined the best possible cycle time. In the following we assume that the LP is feasible.

Previous authors solved this LP either by linear programming [4, 8] or by binary search with a subroutine testing for a negative circuit [11, 10, 12, 2, 7]. However, the problem can be solved more efficiently by a direct combinatorial algorithm of Young, Tarjan and Orlin [14]. We briefly describe their algorithm since we shall extend it in Section 4 and 5. For a detailed description and an efficient implementation see also [1].

The algorithm computes a sequence $-\infty = \lambda_0 \leq \lambda_1 \leq \ldots \leq \lambda_k$ of values for the parameter λ and a sequence $T_1, \ldots, T_k$ of shortest paths trees in G' from a specified vertex r, the root (in our case we can take the vertex corresponding to the artificial variable z_0: all vertices are reachable from this vertex), such that T_i is a shortest paths tree for all parameters λ with $\lambda_{i-1} \leq \lambda \leq \lambda_i$ $(i = 1, \ldots, k)$. The last value λ_k will be the solution of the linear program.

We start by computing a shortest paths tree for $\lambda = -\sum_{e \in E} |c(e)|$. This value of λ is small enough such that no negative circuit exists. The resulting tree is T_1.

Assuming that T_i is already computed we show how to compute λ_i and T_{i+1}. Let P_{rv} be the path from r to v in T_i. We check for each edge $e = (u, v)$ whether the path $P_{ru} + e$ contains more parameterized edges than P_{rv}. If so, $P_{ru} + e$ is a potential pivot path and e is a potential pivot edge. For some value λ_e the path $P_{ru} + e$ will be shorter than P_{rv}.

λ_i is the minimum value λ_e for all potential pivot edges e. One edge $e = (u, v)$ with the minimum value λ_e becomes the pivot edge. We perform a pivot step by deleting the edge with head v from T_i and inserting edge e. The resulting tree is T_{i+1}.

If adding the pivot edge e results in a directed circuit, the algorithm stops. The cost of this directed circuit is zero for λ_e, and $\lambda_e = \lambda_k$ is the maximum value of λ such that G' contains no negative circuit.

The last tree T_k also provides a solution for the variables z_i: one can set z_i to the cost of the path P_{rv_i} in T_k for $\lambda = \lambda_k$. This solution is also called a shortest paths potential.

The worst–case running time of this algorithm (with an efficient implementation) is $O(nm + n^2 \log n)$ where $n = |V(G')|$ and $m = |E(G')|$. However, it is much faster in practice as the experimental results will demonstrate.

We showed how to determine the clocking schedule with the optimum cycle time. However, the solution obtained so far has a serious drawback. Many inequalities of the linear program (in particular all whose corresponding edges belong to the tree T_k) are satisfied with equality. If such a tight inequality corresponds to a data path, then this data path will be critical, i.e. the slack is zero. In the next section we show how the slack can be increased for many critical data paths. In Section 5 we show how to increase slack on clocktree paths optimally.

We should note that the above method can also be used to do static timing analysis with transparent latches, without changing clock arrival times. In fact, even for this task (for which it was not built) the algorithm seems to be superior to traditional methods which use naive iterated propagation.

4 Balancing slacks on data paths

Having computed the optimal cycle time T subject to the constraints described in Section 2 we now increase the slacks on data paths. If ε_{vw}^{max} and ε_{vw}^{min} are the slacks on the data path from v to w for late mode and early mode respectively, the constraints (3) and (4) become

$$(3') \qquad x_v + t_{vw}^{max} + \varepsilon_{vw}^{max} \;\leq\; x_w + \zeta_{vw} T,$$

$$(4') \qquad a_v + y_v + t_{vw}^{min} - \varepsilon_{vw}^{min} \;\geq\; b_w + y_w + (\zeta_{vw} - 1) T.$$

The task is to maximize the slack variables ε_{vw}^{max}, ε_{vw}^{min} for as many data paths as possible, maintaining feasibility. More precisely:

Definition 4.1 *The* **slack balancing problem** *consists of finding a solution of* (1), (2), (3′) *and* (4′) *such that the vector of all slack variables in nondecreasing order is lexicographically minimal.*

It will be shown that there is a unique optimum solution for the slack balancing problem. We now describe an algorithm which finds this solution. It proceeds as follows: The slacks of all data paths are increased simultaneously until they cannot be increased anymore, i.e. some constraints which form a directed circuit, are already tight. Then we take the subset of all data paths on which the slack can still be increased and continue with this subset.

The same digraph $G' = (V, E)$ as in Section 3 is constructed, but the costs are different. The optimal cycle time T is already computed and should not change, it becomes part of the cost of the respective edges. The parameterized edges are now those which correspond to constraints with ε_{vw}^{max} or ε_{vw}^{min} (i.e. (3′) and (4′)), all other edges are not parameterized. The parameter λ now represents the slack of all data paths.

We first compute again the maximum value λ such that G' contains no negative circuit, using the parametric shortest path algorithm described in Section 3. This value is zero (if T was the optimal cycle time), and so is the slack of all data paths for which the corresponding edges belong to the zero cost directed circuit C found by the algorithm. Increasing the parameter λ of any of the parameterized edges on C is impossible, because it would result in a negative circuit. All edges on C lose their parameter, only the parameter of all other edges is increased.

C is contracted, and the costs of the edges leaving and entering C are adjusted: Let z be the vertex to which C is contracted, and let w be the vertex of C nearest to the root r in the last tree T_k computed. For a vertex v of C denote by $c(P_{wv})$ the cost of the path from w to v in T_k for parameter λ_k.

For an edge $e=(v,u)$ leaving C, i.e. v belongs to C but u does not, the corresponding new edge $e'=(z,u)$ after the contraction gets the cost $c(e')=c(e)+c(P_{wv})$. For an edge $e=(u,v)$ entering C, the corresponding new edge gets the cost $c(e')=c(e)-c(P_{wv})$.

The algorithm continues to increase the parameter λ and to change the tree such that it remains a shortest paths tree until the next directed circuit of zero cost is found. The value of the parameter λ at this state is again the slack of the data paths for which the corresponding edges belong to the directed circuit. It is easy to observe:

Theorem 4.2 *The slack balancing problem has a unique optimum solution and the algorithm described above finds this solution.*

The algorithm presented here is a modification of an algorithm for the minimum balance problem [14]. While for the minimum balance problem all edges are parameterized, in our case only those edges are parameterized which correspond to constraints of data paths with variables ε_{vw}^{max} or ε_{vw}^{min}. This is also the reason why we speak of balancing the slacks: the slacks are increased and distributed "equally" on the data paths.

With an efficient implementation the worst–case running time of the algorithm described above is $O(nm+n^2\log n)$. But note that it is not necessary to run the algorithm to the very end. The algorithm can be stopped at any time, e.g. when a certain value of the parameter λ is reached. Then the slack of the data paths are only increased up to this value. Since slacks exceeding a certain amount are usually not interesting this option is used in practice; see Section 6.

5 Balancing slacks on clocktree paths

Prescribed arrival times for clock signals at latches cannot be realized since process variations make it impossible to predict the exact arrival times. This is usually taken into account by adding a constant to the cycle time. If one has an interval $[y_s-\varepsilon_s, y_s+\varepsilon_s]$ of valid clock shifts for each latch s (instead of a fixed number y_s) this problem diminishes (see also [7]).

Clocktrees with prescribed skews can be designed by basically the same algorithm as zero–skew clocktrees. Although this can be done quite efficiently, prescribed skews (zero or not) make detours in the clocktree wiring necessary. If one has intervals for the arrival times of the clock signals one can design clocktrees with significantly smaller wirelength; see e.g. [5].

So for each latch s we introduce an additional variable ε_s: the clock signal is 1 in the interval $[a_s+y'_s, b_s+y'_s]$ for some $y'_s\in[y_s-\varepsilon_s, y_s+\varepsilon_s]$. All constraints must be met for all possible values of y'_s within this interval. ε_s can be considered as the slack on the clocktree path ending at latch s. With these additional variables ε_s for all latches we can reformulate the linear inequalities:

$$(2'') \qquad a_s+y_s+\varepsilon_s \;\le\; x_s \;\le\; b_s+y_s-\varepsilon_s$$

$$(4'') \qquad a_v+y_v-\varepsilon_v+t_{vw}^{min} \;\ge\; b_w+y_w+\varepsilon_w+(\zeta_{vw}-1)T.$$

The problem is solved similarly to the slack balancing problem for data paths (Section 4). However, some modifications are needed since inequality $(4'')$ contains the slack variables ε_v and ε_w of two different latches v and w which are connected by a data path. This can be modelled by introducing an additional variable and splitting the inequality into two. A straightforward modification of the algorithm takes care of the problem that several inequalities contain the same slack variables.

Moreover, it is also possible to balance the slacks on data paths and clocktree paths simultaneously. The constraint (4) is substituted by

$$(4''') \quad a_v+y_v-\varepsilon_v+t_{vw}^{min}-\varepsilon_{vw}^{min} \;\ge\; b_w+y_w+\varepsilon_w+(\zeta_{vw}-1)T.$$

We look for a solution of constraints (1), $(2'')$, $(3')$ and $(4''')$. For each constraint $(4''')$ we introduce two additional variables and split the constraint into three. Then we apply basically the same algorithm as above. It can be derived in the same way as above that the optimum solution is unique and that the algorithm finds it. We obtain:

Theorem 5.1 *The slack balancing problem for data and clocktree paths (constraints (1), $(2'')$, $(3')$ and $(4''')$) has a unique optimum solution which can be computed in time $O(nm+n^2\log n)$ where n is the number of primary inputs, primary outputs and latches and m is the number of data paths.*

The running time follows from bounding the number of pivot steps by $O(n^2)$. The details will be given in the full version of this paper (http://www.or.uni-bonn.de/~vygen/cycleopt.ps).

6 Computational Results

We have implemented the algorithm in C, all runs are on an IBM RISC System/6000 Model 595. Our algorithm has been applied, among others, to the G3 series of IBM S/390 processor chips (L2 and PU) and the latest follow ups (MBA). For details of the design system see [6]. Table 1 shows the different chips with target cycle time, number of circuits, nets, pins and primary inputs and outputs (some are bidirectional) and the number of data paths.

In addition to the constraints described so far further technical restrictions had to be taken into account. For example, for some master–slave latches it is required that the data signal arrives at the latch before the rising time of the clock signal of the slave latch (end–of–cycle test). In this case one has a constraint of the form

$$x_v+t_{vw}^{max} \le a_w+y_w+\zeta_{vw}T.$$

Moreover, for certain registers (which can be regarded as sets of latches) simultaneous clock signals are required. In this case only one y–variable for these latches is needed. For implementation reasons we still have separate y–variables and constraints of type $y_v+0\le y_w$ and $y_w+0\le y_v$. This explains the differences in the number of vertices and edges of the graph G' in Table 2 to the numbers which one would expect by Table 1.

Our program consists of two main parts: first the constraints are generated by simple forward propagation for late mode and early

Chip	cycle time (ns)	circuits	nets	pins	primary IOs	latches	data paths
L2	6.5	87177	103590	339351	928	17032	1173132
PU	6.5	164056	171666	591410	744	17265	2670459
MBA	4.46	394257	402373	1441312	586	40639	1475535

Table 1: Characteristics of chips.

	size of G'		running time (s)	
Chip	nodes	edges	timing	create G'
L2	52999	2103937	104.11	419.71
PU	68932	5433150	178.67	1377.62
MBA	268153	3637831	859.25	1006.31

Table 2: Size of the digraph G' constructed for balancing slacks on data paths. Total running time in seconds for evalutaing the timing of the chip by simple propagation (timing) and creating all constraints (create G').

mode constraints. During the propagation we store at each circuit the set of primary inputs and latches from which this circuit can be reached, along with the maximum propagation delay for late mode, resp. the minimum propagation delay for early mode. Table 2 (right–hand side) shows the running times for the generation of all constraints for the three different chips. A detailed description of the timing analysis program can be found in [9]. The running times for simple propagation (computation of arrival times and slacks only) are shown for comparison.

The second step consists of the main optimization algorithm. Rather than increasing the slacks on late mode and early mode constraints simultaneously, we first increase the slacks on late mode constraints up to a certain value λ^{late} while ignoring all early mode constraints. Then we add all early mode constraints and increase the slack on these constraints up to λ^{early}. For this the late mode constraints are added with unparameterized edges, such that the slack of late mode constraints with slack smaller than λ^{late} does not decrease and the slack of all other late mode constraints remains at least λ^{late}. The reason for treating late and early mode constraints differently is that early mode problems can usually be fixed quite efficiently by inserting a buffer. Finally, we balance the slacks on clock tree paths up to a value λ^{clock} as described in Section 5. Again it is assured that the slacks of late and early mode constraints do not decrease below the value to which they were optimized.

Obviously, the result of the optimization depends on the length of the interval given by l_s and u_s specifying by how much the clock signal arrival time can be shifted from its nominal value. For the L2 and PU the designers called for $l_s = -0.4$ns and $u_s = 0.4$ns for most latches. However, there was a substantial number of latches whose clock signal had to arrive at a prescribed time, i.e. $l_s = u_s = 0.0$ns. On the MBA we had $l_s = -0.4$ns and $u_s = 0.5$ns for most latches.

Table 3 shows the worst slack of all late mode constraints before and after optimization with respect to the target cycle time mentioned in Table 1. This means that the L2 could run with a cycle time of $6.5\text{ns} + 0.048\text{ns} = 6.548\text{ns}$ before the optimization and with a cycle time of $6.5\text{ns} - 0.313\text{ns} = 6.187\text{ns}$ after optimization: the cycle time was improved by 5.5%. Similarly, the cycle time of

Chip		worst slack	number data paths with late mode slack < -0.2	-0.1	0.0	0.1	0.2	0.3
L2	(a)	-0.048	0	0	594	731	740	5781
	(b)	0.313	0	0	0	0	0	0
PU	(a)	-0.103	0	1	143	1384	11349	51578
	(b)	0.060	0	0	0	44	1617	44285
MBA	(a)	-0.224	5	44	400	2633	9901	21780
	(b)	-0.051	0	0	28	89	2283	18768

Table 3: Improvement of the worst slack and the number of critical data paths for the three different chips: (a) before optimization, (b) after optimization.

Chip	cycle time	λ^{late} = 0.1	λ^{early} = 0.1	λ^{clock} = 0.05	λ^{late} = 0.2	λ^{early} = 0.2	λ^{clock} = 0.1
L2	153.21	4.91	6.93	11.56	4.94	31.87	300.54
PU	19.45	22.35	123.33	569.68	113.68	326.17	13004.25
MBA	20.36	37.63	135.48	671.48	81.34	259.43	92490.36

Chip	λ^{late} = 0.3	λ^{early} = 0.3	λ^{late} = 0.4	λ^{early} = 0.4	λ^{late} = 0.5	λ^{early} = 0.5
L2	5.45	97.48	104.80	634.16	282.21	1608.97
PU	1140.04	1768.60				
MBA	212.60	11011.82				

Table 4: Total running time in seconds for the main optimization routine: for computing the optimal cycle time and for balancing slacks on late and early mode constraints and also clock tree paths up to different values for λ^{late}, λ^{early} and λ^{clock}.

number of values: 1475535 minimum value: -0.223992

0.300 - 0.350	8635	**********
0.250 - 0.300	6754	**********
0.200 - 0.250	5125	**********
0.150 - 0.200	5239	**********
0.100 - 0.150	2029	******
0.050 - 0.100	1535	*****
-0.000 - 0.050	698	**
-0.050 - -0.000	255	
-0.100 - -0.050	101	
-0.150 - -0.100	37	
-0.200 - -0.150	2	
-0.250 - -0.200	5	

Figure 3: Slacks of late mode constraints (ε_{vw}^{max}) without optimization for MBA.

the PU was improved by 2.5% and the cycle time of the MBA by 3.7%.

Table 3 also shows the number of data paths with late mode slack smaller than -0.2ns, -0.1ns, ... This demonstrates how dramatically the number of critical paths decreases.

In Table 4 the running times for the algorithm for the three chips for computing the optimal cycle time and for balancing slacks for different scenarios with λ^{late}, λ^{early} and λ^{clock} are shown. Again, all late mode slacks are taken with respect to the target cycle times.

Figure 3 shows a frequency distribution of the slacks of all late mode constraints for the MBA for the case that no optimization is possible (i.e. $l_s = u_s = 0.0$ns for all latches s). The first column shows the different intervals, the second column gives the number of data paths whose slack is within the interval and the third column gives a graphical representation of this number by a proportional number of stars. For example Figure 3 says that the MBA without optimization has 101 datapaths with $-0.100\text{ns} \le \varepsilon_{vw}^{max} < -0.050\text{ns}$. Figure 4 shows the corresponding frequency distributions of late mode constraints after optimization for $\lambda = 0.2$ns.

Figures 5 and 6 show the same for the early mode constraints.

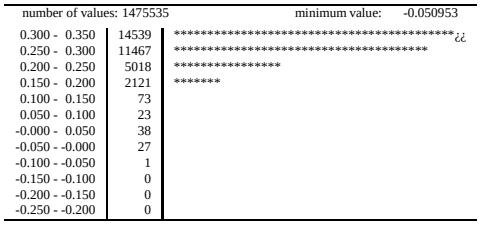

Figure 4: Optimized late mode slacks (ε_{vw}^{max}) for the MBA.

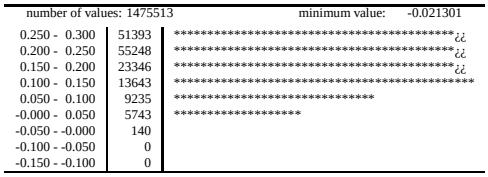

Figure 5: Slacks of early mode constraints (ε_{vw}^{min}) without optimization for MBA.

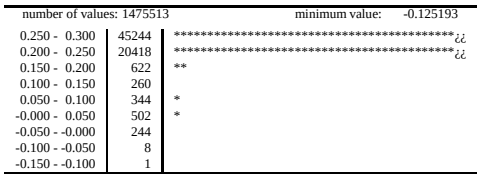

Figure 6: Optimized early mode slacks (ε_{vw}^{min}) for the MBA.

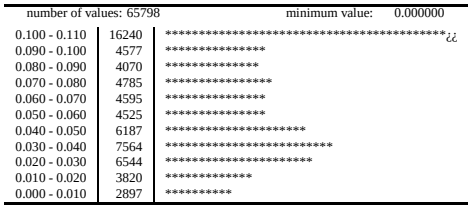

Figure 7: Slack on clock tree (ε_s) paths for the MBA.

Increasing the slacks on late mode constraints makes the worst slack of early mode constraints worse, but nevertheless the total number of early mode constraints with negative slack decreases considerably.

Finally, in Figure 7 the result of balancing the slacks on clock tree paths is shown. Even though the slacks on late mode and early mode constraints have already been increased up to 0.2ns, for most of the latches the clock signal still does not have to arrive at exactly the prescribed time. Due to a design contraint all interval centers y_s are between -0.4ns and 0.5ns; in fact more than 75 percent are within ± 0.2ns.

7 Conclusion

We showed how to find the clock schedule with minimum cycle time and optimum slack distribution by very efficient network algorithms. Experimental results show that this not only reduces the cycle time but also decreases the number of critical data and clocktree paths considerably.

References

[1] Bünnagel, U.: Effiziente Implementierung von Netzwerkalgorithmen. Diploma thesis, University of Bonn, 1998

[2] Deokar, R.B., Sapatnekar, S.: A Graph–theoretic Approach to Clock Skew Optimization. Proceedings of the IEEE International Symposium on Circuits and Systems (1994), 407-410

[3] Doettling, G., Getzlaff, K.J., Leppla, B., Lipponer, W., Pflueger, T., Schlipf, T., Schmunkamp, D., Wille, U.: S/390 Parallel Enterprise Server Generation3: A balanced system and cache structure. IBM Journal of Research and Development 41 (1997), 405-428

[4] Fishburn, J.P.: Clock Skew Optimization. Transactions on Computers C-39 (1990), 945-951

[5] Huang, J.H., Kahng, A.B., Tsao, C.-W.A.: On Bounded–Skew Routing Tree Problem. Proceedings of the 32nd Design Automation Conference (1995), 508-513

[6] Koehl, J., Baur, U., Ludwig, T., Kick, B., Pflueger, T.: A Flat, Timing–Driven Design System for a High-Performance CMOS Processor Chipset. Proceedings of the Conference Design, Automation and Test Conference in Europe (1998), 312-320

[7] Neves, J.L., Friedman, E.G.: Optimal Clock Skew Scheduling Tolerant to Process Variations. Proceedings of the 33rd Design Automation Conference (1996), 623-628

[8] Sakallah, K.A., Mudge, T.N., Olukotun, O.A.: *check*T_c and *min*T_c: Timing Verification and Optimal Clocking of Synchronous Digital Circuits. Proceedings of the IEEE International Conference on Computer–Aided Design (1990), 552-555

[9] Schietke, J.: Timing–Optimierung beim physikalischen Layout von nicht–hierarchischen Designs hochintegrierter Logikchips. Ph.D. thesis, University of Bonn, 1999

[10] Shenoy, N., Brayton, R.K.: Graph Algorithms for Clock Schedule Optimization. Proceedings of the IEEE International Conference on Computer–Aided Design (1992), 132-136

[11] Szymanski, T.G.: Computing Optimal Clock Schedules. Proceedings of the 29th Design Automation Conference (1992), 399-404

[12] Szymanski, T.G., Shenoy, N.: Verifying Clock Schedules. Proceedings of the IEEE International Conference on Computer–Aided Design (1992), 124-131

[13] Vygen, J.: Plazierung im VLSI-Design und ein zweidimensionales Zerlegungsproblem. Ph.D. thesis, University of Bonn, 1996

[14] Young, N.E., Tarjan, R.E., Orlin, J.B.: Faster Parametric Shortest Path and Minimum Balance Algorithms. Networks 21 (1991), 205-221