Efficient Timing Constraint Derivation for Optimally Retiming High Speed Processing Units

A. van der Werf\textsuperscript{1}  J.L. Van Meerbergen\textsuperscript{1}  E.H.L. Aarts\textsuperscript{1,2}  W.F.J. Verhaegh\textsuperscript{1}  P.E.R. Lippens\textsuperscript{1}

\textsuperscript{1} Philips Research Laboratories, Prof. Holstlaan 4, 5656 AA Eindhoven, The Netherlands
\textsuperscript{2} Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands

Abstract

We apply retiming including pipelining to make Processing Units (PUs) run at a required throughput rate containing a minimum number of registers. In a first step, retiming performs a timing analysis of a PU which results in inequality constraints on operations' retimings. The constraints form together with a cost function expressing the number of registers in a retimed PU an instance of an ILP problem which is solved to optimality in a second step. In this paper we concentrate on the constraint derivation task. We present two new constraint derivation algorithms, one of which is more memory efficient and the other more run time efficient. We show that the run time efficient algorithm makes it possible to minimize the area of a huge standard cell network, possibly representing a complete IC, within acceptable run time limits.

1 Introduction

Manually designing application specific video signal processors is a cumbersome and error prone task. The CAD methodology PHIDEO\textsuperscript{1} is developed to automate part of this task and to assist the designer in exploring the design space [1]. Part of PHIDEO is the design of high speed Processing Units (PUs) which implement the computationally demanding parts of the system specification [2]. A PU must run at a required throughput rate, i.e., each path in the combinational logic between registers must have a delay smaller than a specified clock period, such that during one clock period all the functional elements are given sufficient time to settle their outputs. To achieve this, we apply retiming which is a technique to position registers in a PU such that the delay of each path is smaller than the clock period [3]. The name of the technique stems from the way in which the positions of the registers in a retimed PU are determined, namely by retiming operations a number of clock cycles earlier or later with respect to their original execution times. Except for the reason to obtain a PU with a required throughput rate, retiming may be applied to give a causal behaviour to a PU which initially contains negative signal delays. Pipelining is considered part of retiming by allowing it to change consumption and production times of I/O signals. A designer may constraint retiming by restricting the freedom to pipeline a PU such that at its boundary it matches the timing behaviour of its environment. Retiming can be applied successfully in other application areas than video signal processing. Therefore, we developed a stand alone retiming tool called OPTIMA [6] based on the techniques that are presented in this paper.

One may distinguish between two types of retiming problems. In the optimization variant the problem is to find a retiming that meets the timing constraints using a minimum number of registers. In the decision variant one is asked to find a feasible retiming. Often the PUs of a video signal processor contain many pipeline stages and consequently the registers contribute to a great extend to the area and power consumption of the processor. Therefore, we need optimally retiming which results in PUs with minimal numbers of registers instead of feasibly retiming which leads to very expensive PU implementations. In this paper we focus on efficient algorithms for optimally retiming PUs.

Optimally retiming a PU can be done in polynomial time by the classical method introduced by Leiserson and Saxe [3]. This method consists of two major steps. In the first step, an instance of a retiming problem is derived and cast into an instance of a special case of an Integer Linear Programming (ILP) problem, namely the dual of a minimum-cost flow problem. The major task of the first step is a timing analysis resulting into inequality constraints on operations' retimings. According to Leiserson and Saxe, an all pairs shortest path algorithm is required to derive the inequalities constraints. In the second step, the ILP problem instance is solved to optimality by a minimum-cost flow algorithm. In this paper we concentrate on the first step, and more specifically on the

\footnotesize{\textsuperscript{1}This project is sponsored by the EU as ESPRIT 2260}
constraint derivation task.

We use retiming to optimize PUs which are given in a net list format at the level of standard cells. Furthermore, the PUs are characterized by a small ratio between the clock period and the delay of the longest path which typically results in pipelined PUs. Given these characteristics, a drawback of Leiserson and Saxe's constraint derivation algorithm is that the first step may be very time consuming and that it may generate an abundance of constraints. This was also concluded by Munzner and Hemme [4] which was for them a reason to develop an approximation algorithm for pipelining. Although we worked initially on approximation methods [2] we went through pursuing efficient optimization methods and succeeded. We found that, in this context, an important property of the timing constraints is redundancy. Redundancy may be present among the constraints which means that some constraints can be left out or added without any consequences for the obtained result. In this paper we show that when the clock period is small compared to the delay of the longest path in a PU, redundancy in constraints can be used to make a significant trade off between storage of the constraints and the speed of an optimal retiming algorithm. We present two new constraint derivation algorithms, one of which is more memory efficient and the other more run time efficient. We show that the run time efficient algorithm makes it possible to minimize the area of a huge standard cell network, possibly representing a complete IC, within acceptable run time limits.

In Section 2 we present a formal definition of PUs and the timing model of operations which is used to determine delays of paths in PUs. In Section 3 we present the two better constraint derivation algorithms. Results are presented in Section 4 and conclusions in Section 5.

2 Formal Model

2.1 Processing Units and Retiming

Definition 2.1 (processing unit)
A processing unit $G$ is given by a four tuple $(V,F,E)$ in which

$V$ : a set of operations,
$F \subseteq V \times V$ : a multi set of constraint edges, representing timing constraints,
$T = O \cup I \subseteq V \times \mathbb{N}$ : a multi set of operation terminals
where $O$ is a set of output terminals, and $I$ is a set of input terminals.

$E \subseteq O \times I$ : a set of signal edges, representing the signal flow.

A signal in a PU may be delayed a number of clock cycles, which can be regarded as a relation between the execution time of the operation producing the signal and the one consuming it. Also the timing constraint edges impose a relation between the execution times of operations. To quantify these relations a weight function on both types of edges is defined as follows.

Definition 2.2 (weight function)
Given a PU $G$, then the weight function $w : F \cup E \rightarrow \mathbb{Z}$ is defined as follows. (i) For each $f = (u, v) \in F$, $w(f)$ is the maximum number of clock cycles $u$ can be delayed more than $v$. (ii) For each $e \in E$, $w(e)$ is the number of clock cycles a signal on a signal edge $e$ is delayed.

The signal edges between operation terminals in a PU also imply a time order of the operations, i.e., each signal must be delayed a positive number of clock cycles. Therefore, for each $e = ((v, a), (u, b)) \in E$ there exists an $f = (v, u) \in F$ with $w(f) \leq w(e)$.

Note that a given PU which must be retimed may have a non-causal behaviour if there exists an $e \in E$ with $w(e) < 0$.

Definition 2.3 (retiming)
Given a PU $G$ and a weight function $w$, then a retiming is a function $r : V \rightarrow \mathbb{Z}$ that returns for each operation in $V$ the number of clock cycles the operation is delayed with respect to its original execution time. Furthermore, retiming defines a retimed weight function $w_r : F \cup E \rightarrow \mathbb{Z}$ as follows. (i) For each $f = (u, v) \in F$, $w_r(f) = w(f) + r(v) - r(u)$. (ii) For each $e = ((u, a), (v, b)) \in E$, $w_r(e) = w(e) + r(v) - r(u)$. A retiming of a PU $G$ is feasible iff for each $f = (u, v) \in F$, $w_r(f) \geq 0$ which is equivalent to $r(u) - r(v) \leq w(f)$.

A retimed PU is a combination of a PU and a retiming function defined on its operations.

Retiming a PU in order to minimize the number of registers and subject to timing constraints can be done in polynomial time as is shown by Leiserson and Saxe [3]. They showed that the retiming problem can be mapped on the dual of the minimum-cost flow problem. Therefor a graph is constructed in which operations are represented by nodes and constraints by edges. We do not further explain this but we concentrate on the efficient derivation of sufficient timing constraint edges in the rest of this paper. Minimum-cost flow algorithms belong to the class of network optimization methods for which we refer to an excellent overview given by Ahuja et al. [5].

The maximum achievable processing rate at which a PU can operate depends on the delay of operations. The
Definition 2.4 (delay)
Given a PU \( G \), then the delay is a function \( \text{del} : I \times O \rightarrow \mathbb{R}^+ \cup \{\text{undef}\} \) that returns the delay between an input and an output. If there is no path between the input and output inside one operation, then the delay is undefined, which is indicated by \( \text{undef} \). \( \square \)

To allow the timing model to be used for sequential operations, e.g., pipelined multiplications, it can be extended with a number of functions as described in [7].

Below we assume that for a PU \( G \), a given clock period \( C \), and delay function \( \text{del} \), every operation in the PU can be executed within one clock period, i.e., for each \( (i, o) \in I \times O \), if \( \text{del}(i, o) \neq \text{undef} \) it follows that \( \text{del}(i, o) < C \).

We discuss some basic theory on paths in PUs that we need for the discussion on the derivation of speed constraints.

Definition 2.5 (path)
Given a PU \( G \), then a path \( p \) is an alternating sequence of output and input terminals starting with an output and ending with an input of the following form: \( p = (o_1, i_2, o_2, \ldots, o_{n-1}, i_n) \), where for each \( k \in \mathbb{N} \), \( 1 \leq k \leq n - 1 : (o_k, i_{k+1}) \in E \) and \( \text{del}(i_k, o_k) \neq \text{undef} \). Furthermore, \( P_G \) denotes the set of all paths in \( G \). \( \square \)

The notion of delay functions in the timing model and weights on signal edges is extended to the domain of paths.

Definition 2.6 ((retimed) path delay)
Given a PU \( G \) and a functions \( \text{del} \), then the path delay is a function \( D : P_G \rightarrow \mathbb{R}^+ \) that returns the delay of a path which is defined as follows. For each \( p = (o_1, i_2, o_2, \ldots, o_{n-1}, i_n) \in P_G \), \( D(p) = \text{drt}(o_1) + \sum_{k=2}^{n-1} \text{del}(i_k, o_k) + \text{dat}(i_n) \), where \( \text{drt}(o_1) = \max\{\text{del}(i, o_1) | i \in I \text{ and } \text{del}(i, o_1) \neq \text{undef}\} \) and \( \text{dat}(i_n) = \max\{\text{del}(i_n, o) | o \in O \text{ and } \text{del}(i_n, o) \neq \text{undef}\} \). The retimed path delay function \( D_r \) is equivalently defined for paths in a retimed PU. \( \square \)

Note that the delay of a path does not depend on the weights of the signal edges in the path. Consequently, the delay of a path can be greater than the clock period at which a PU can be executed.

Definition 2.7 ((retimed) path weight)
Given a PU \( G \) and a function \( w \), then the path weight is a function \( W : P_G \rightarrow \mathbb{Z} \) that returns the weight of a path which is defined as follows. For each \( p = (o_1, i_2, o_2, \ldots, o_{n-1}, i_n) \in P_G \), \( W(p) = \sum_{k=1}^{n-1} w(o_k, i_{k+1}) \). The retimed path weight function \( W_r \) is likewise defined for paths in a retimed PU, given the retimed weight function of the signal edges. \( \square \)

3 Derivation of Speed Constraints

A path between two shift registers in a retimed PU must have a delay less than the clock period, i.e., for each \( p = ((u, a), \ldots, (v, b)) \in P_G \), \( W_r(p) = 0 \Rightarrow D(p) < C \). In other words, every path that is longer than the clock period must have placed at least one register on it, i.e., for each \( p \in P_G \), \( D(p) \geq C \Rightarrow W_r(p) \geq 1 \). Since \( D \) is invariant under retiming, we can detect in a given PU before retiming which paths are longer than the clock period after retiming. For each of these paths \( p \) we must have a constraint stating that \( W_r(p) \geq 1 \). Since \( W_r(p) - 1 \geq 0 \Leftrightarrow r(v) - r(u) \geq 1 - W(p) \) it follows that a constraint edge \( f = (u, v) \) must be in the set \( F \) of the PU \( G \) and that the weight function of the constraint edge is defined to be \( w(f) = W(p) - 1 \). A feasible retiming meets this constraint and consequently satisfies \( W_r(p) \geq 1 \).

For constraint derivation we make a trade-off between the number of derived constraints that must be stored and the run-time of the derivation algorithm. The number of constraints also affects the run time of the minimum-cost flow algorithm, but that topic is considered outside the scope of this paper. Derivation is a special form of path length analysis where the distance is defined in terms of weight and delay. Since PUs have a sparse structure, derivation is performed starting from every output of an operation in a PU. The derivation per starting output is partitioned into two steps. In a first step, inputs are labelled with delay labels (\( D \)) and weight labels (\( W \)) and in a second step, these labels are used to generate constraint edges to be included in the set \( F \) of a PU \( G \). Before discussing labelling methods we first discuss constraint generation because some of its characteristics are important for the discussion of the first step.

3.1 Constraint generation

The result of the first step is that inputs are labelled with delays and weights which represent for each input the delay and weight of one of the reconvergent paths from the starting output to it. The constraint generation is implemented as a depth first search for inputs labelled with a delay greater than the clock period along paths defined by the labels. At every input the labels are used to select a path from the set of reconvergent paths from the starting output to it, for which a corresponding constraint edge may be generated. More specif-
ically, during the generation of constraint edges, paths are followed by only traversing from an input of an operation \(i_1 = (u, a) \in I\) via an output of the same operation \(o_1 = (u, b) \in O\), \(\text{del}(i_1, o_1) \neq \text{undef}\) to an input of another operation \(i_2 \in I, e = (o_1, i_2) \in E\), if \(W(i_2) = W(i_1) + w((o_1, i_2))\) and \(D(i_2) = D(i_1) + \text{del}(i_1, o_1)\). If an input is labelled with a delay greater than the clock period, then a path with that delay runs between the starting output and the input and consequently a constraint edge must be generated from the operation to which the output belongs to the operation to which the input belongs for which the weight label is used. Each path that has a subpath with a delay greater than the clock period has a redundant corresponding constraint edge such that the depth of the search can be limited by the clock period.

3.2 Delay and Weight Labelling

The task of the labelling step is to label an input with a delay and a weight which represent the delay and weight of one of the reconvergent paths from the starting output to the input. We discuss three different labelling methods which differ in selecting one of the reconvergent paths to which the labels belong. We first discuss the relevant path labelling method that derives as few constraint edges as possible. Second, the classical labelling method is presented based on the work by Leiserson and Saxe. The third method is the clock period limited labelling method, which takes the clock period into account during derivation and produces of all methods the most redundant constraint edges.

### Relevant path labelling

On all reconvergent paths the same number of registers is added by retiming. A relevant path between two operations is the path on which the largest number of registers must be added by retiming. So a relevant path \(p_r \in P_G\) is a path for which every reconvergent path \(p_s \in P_G\), \(\frac{\text{delay}(p_s) - W(p_r)}{\text{clock}} \geq \frac{\text{delay}(p_s) - W(p_r)}{\text{clock}}\). In order to generate as few as possible constraint edges, each input is labelled with the delay and weight of a relevant path to it. The delay and weight of relevant paths starting from one output to various inputs can be derived by a shortest path algorithm for which the distance can be determined as follows. Traversing a signal edge \(e\) increases the distance with \(w(e)\), while traversing through an operation, from an input \(i\) to an output \(o\), decreases the distance by \(\frac{\text{del}(i, o)}{\text{clock}}\).

To explain the labelling method, we use an example PU as shown in Figure 1. In the figure the labels belonging to a path length analysis starting from the output of operation A and the generated constraint edges for that starting output are shown. It should be noted that the delay label of an input does include the data available time of the input and the data ready time of the starting output. The labels define the paths which the constraint generation must follow. In the figure the signal edges and the operation delay edges of these paths are indicated by bold arrows.

![Figure 1: An example of a PU with labels corresponding to the output of operation A.](image)

**Classical labelling** In this section a method is presented that is primarily based on the work by Leiserson and Saxe [3]. Retiming removes from all reconvergent paths between two operations the same number of registers. Furthermore, in order to assure feasibility, retiming will not remove more registers than available on the path with the minimum number of registers on it. Consequently, a path that does not have the smallest number of registers on it has one or more registers trapped on it. Therefore, it is not necessary to generate a constraint to let retiming place at least one register on such a path.

A shortest path algorithm can be used to label inputs with the delay and weight of paths that do not have registers trapped on it. Of all reconvergent paths the path with the smallest weight, and if there is more than one, the one of these with the greatest delay is considered the shortest. Like the previously discussed method the labelling method must label each input for every output. In other words, an all-pairs shortest path algorithm is required. Compared to the previous method many more constraint edges may be generated since the path with the minimum number of registers on it may be different from the path on which the highest number of registers must be added.

Figure 2 shows the labels and generated constraint edges which correspond to the classical labelling method. Compared to the relevant path labelling method one additional constraint edge is generated; cf. Figure 1. This constraint edge is corresponding to the path from the out-
put of operation A via operation C to an input of operation D which is different from the relevant path from the output of operation A, via operation B and C to the same input of D.

Clock period limited labelling  Disadvantage of both methods presented above is that an instance of an all-pairs shortest path problem must be solved. When the delay of the longest path in a PU is (considerably) greater than the clock period, it is very likely that not all input labels are used because during constraint generation the search depth is limited by the clock period. We present a new labelling algorithm that avoids unnecessary labelling by taking the clock period into account such that the number of labelled inputs is reduced. The algorithm labels only inputs that have a delay smaller or just greater than the clock period. Although this labelling method is more run time efficient, more redundant constraint edges may be generated compared to the previous presented methods which is explained as follows. Not all paths arriving at an input are known, only those that have a delay smaller or just longer than the clock period. If for one of the known paths a corresponding constraint edge is generated, it is possible that this constraint edge is redundant with respect to one of the corresponding constraint edges of the unknown paths to the input.

Figure 3 shows the labels and generated constraint edges which correspond to the clock period limited labelling method. Compared to the previous two labelling methods additional constraint edges are generated; cf. Figures 1 and 2.

4 Results

We implemented efficient variants of the classical and clock period limited labelling methods and used them for generating some results. Although we do not have an implementation available of the relevant path labelling method, we can estimate the number of constraint edges it would generate in the following way. First, we use the efficient clock period limited algorithm to determine a set of constraint edges. Second, we remove all constraint edges from the set that do not lie on a shortest path between two operations, where a path consists of constraint edges and its length is the total weight of the edges. This is allowed since in a solution of an instance of the uncapacitated minimum-cost flow problem there only exists flow between two operations over the shortest paths between them. The resulting number of constraint edges is an upper bound on the number of constraint edges that a relevant path labelling method would have derived.

The three methods are applied to two PUS: P1 and P2. In Table 1 characteristics of the PUs and the results are shown. It should be noted that P2 represents a complete IC including memories! From the ratios between the clock period and the delay of a longest path in a PU it can be concluded that both retimed PUS must contain a number of pipeline stages. The CPU times are obtained on a 124 MIPS workstation. In the table also the size of a matrix representation of the shortest paths between all pairs of operations as originally proposed by Leiserson and Saxe is given.

From the results we conclude that for the two PUs, the clock period limited labelling algorithm is about 20 times faster than the classical labelling algorithm at the cost of a marginal number of additional redundant constraints. Furthermore, the relevant path labelling algorithm will derive
more than 4 times less constraints than the clock period limited or classical labelling algorithms and representing constraints between all pairs of operations requires more than 100 times more memory space.

5 Conclusions

We revisited the problem of deriving sufficient timing constraints for creating an instance of the retiming problem. We showed that in addition to a classical constraint derivation algorithm according to the theory of Leiserson and Saxe [3], two better alternatives exist. Both algorithms are based on redundancy in constraints. One algorithm derives constraints containing less redundancy and consequently is more memory efficient while the other allows much redundancy but is more run time efficient. Although the two new algorithms have theoretically the same worst-case run time complexity expressed in the number of operations and the number of signals as the classical one, the practical difference is important. The advantages of the new algorithms become larger when the clock period is small compared to the largest path delay in a PU. This situation is quite common in video signal processors, where retimed processing units often contain many pipeline stages. Experimental results show that the run time efficient algorithm makes it possible to minimize the area of a huge standard cell network which represents a complete IC within acceptable run time limits. Furthermore, the presented results show that a speed improvement of more than 20 times or a memory occupation improvement of more than 4 times can be achieved. Depending on the implementation constraints (on memory or speed) of a CAD method including retiming, one of the two algorithms can be selected.

Table 1: Characteristics of two PUs and the results of three constraint derivation methods applied to them.

<table>
<thead>
<tr>
<th></th>
<th>P1</th>
<th>P2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>V</td>
<td>$</td>
</tr>
<tr>
<td>gate count</td>
<td>20K</td>
<td>40K</td>
</tr>
<tr>
<td>clock period (ns.)</td>
<td>21</td>
<td>24</td>
</tr>
<tr>
<td>longest path delay (ns.)</td>
<td>159</td>
<td>150</td>
</tr>
<tr>
<td>$</td>
<td>F</td>
<td>$</td>
</tr>
<tr>
<td>relevant path</td>
<td>73260</td>
<td>186303</td>
</tr>
<tr>
<td>classical</td>
<td>302818</td>
<td>800784</td>
</tr>
<tr>
<td>clock period limited</td>
<td>302994</td>
<td>803188</td>
</tr>
<tr>
<td>$</td>
<td>V</td>
<td>\times</td>
</tr>
</tbody>
</table>

References


