Computing Lower Bounds on Functional Units Before Scheduling[1]

Samit Chaudhuri[2], Robert A. Walker[3]

Abstract— **This paper presents a new polynomial-time algorithm for computing lower bounds on the number of functional units (FUs) of each type required to schedule a data flow graph in a specified number of control steps. A formal approach is presented that is guaranteed to find the tightest possible bounds that can be found by relaxing either the precedence constraints or integrality constraints on the scheduling problem. This tight, yet fairly efficient, bounding method can be used to estimate FU area, to generate resource constraints for reducing the search space, or in conjunction with exact techniques for efficient optimal design space exploration.**

I. Introduction

One of the central problems in high-level synthesis is the *scheduling* problem – the problem of mapping operations onto control steps in the proper order. The process of solving the scheduling problem can be viewed as the process of exploring a 2-dimensional (2D) design space, with axes representing time (schedule length) and area (ideally total area, but often simplified to functional unit area). This 2D design space is shown in Figure 1, where feasible designs lie in the shaded region, infeasible designs lie in the white region, and optimal designs lie on the curve between the two regions.

A variety of methodologies can be used to explore this design space. If optimal solutions are required, then *exact techniques*, such as ILP formulations [1], [2], must be used. However, exploring the entire design space with exact techniques can be extremely time consuming, so in practice a more efficient methodology may be required. One such methodology uses heuristic algorithms to quickly derive two *bounds on the optimal solutions*: a lower bound (which may be infeasible) and an upper bound (which must be feasible).

Although these lower bounds do not necessarily correspond to feasible designs, they can still provide much valuable information during the design process. For example, the lower bounds provide an estimate of the FU area required for a design. They can also be used to add resource constraints to a Time-Constrained Scheduling (TCS) problem, reducing the size of the feasible region of that problem so that it can be solved more quickly. Finally, they can be used together with heuristic scheduling algorithms (which produce upper bounds) for efficient optimal design space exploration: if those two simple approaches produce the same result, then that result is optimal, and the search is complete; only in those cases when the two differ must a more time-consuming exact technique be used instead.

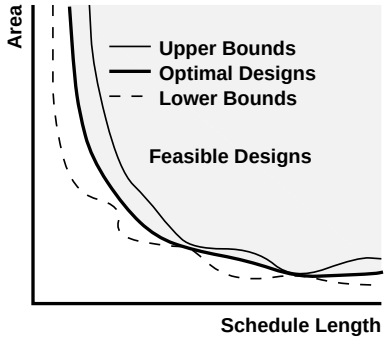

Fig. 1. The Design Space for the Scheduling Problem

A. Formulating the FU Lower-Bounding Problem

This Functional Unit Lower-Bounding (FU-LB) problem derives from the Functional Unit Minimization (FU-Min) problem: given a Data Flow Graph (DFG) and a time constraint c on the schedule length, compute the minimum number m_k^* of FUs needed for any feasible schedule, for each FU-type $k \in K$ (the set of functional unit types). Unfortunately, the FU-Min problem is NP-hard, so there is little hope of finding a polynomial-time solution.

However, an easier problem, called the Functional Unit Lower-Bounding (FU-LB) problem, can be formulated to compute a *lower bound* $\underline{m_k^*}$ on m_k^*. The goal is then to solve this easier problem, preferably in polynomial time, and to use the results for more efficient design space exploration.

Unfortunately, there is no single, unique FU-LB problem. A FU-LB problem is formed by relaxing some constraints of the original FU-Min problem, and different relaxations produce different FU-LB problems: some easier to solve, some harder; some producing tighter bounds, some looser. A comprehensive design space exploration system should therefore provide a "suite" of efficient solutions to a variety of FU-LB problems, and allow the designer to choose the appropriate algorithm for the task at hand (see Table I).

[1]This material is based upon work supported by the National Science Foundation under Grant No. MIP-9211323.

[2]This work was performed when S. Chaudhuri was with the Department of Electrical, Computer, and Systems Engineering, Rensselaer Polytechnic Institute, Troy, NY 12180. S. Chaudhuri is now with Cadence Design Systems, Inc., Chelmsford, MA 01824. E-mail: samit@cadence.com.

[3]R. A. Walker is with the Department of Computer Science and the Department of Electrical, Computer, and Systems Engineering, Rensselaer Polytechnic Institute, Troy, NY 12180. E-mail: walkerb@cs.rpi.edu.

	Precedence Relaxation	LP-Relaxation	Tightest Prec. Relaxation
Quality of Bounds	Fairly Tight	Usually Tightest	Guaranteed Tightest
Complexity of Solution	$O(c^2n)$	Solve 1 LP (polynomial)	Solve 2 LPs (polynomial)

TABLE I
DIFFERENT APPROACHES TO FU LOWER-BOUNDING

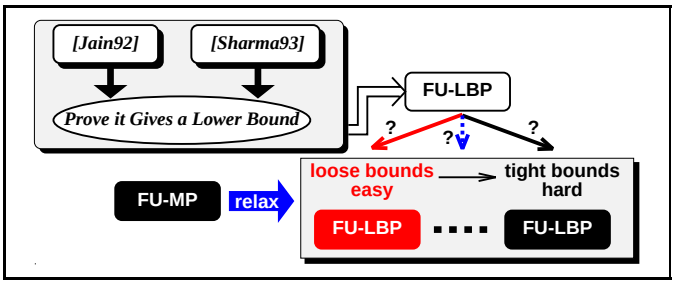

Fig. 2. Previous Approaches to Solving the FU-LB Problem by Relaxing the Precedence Constraints

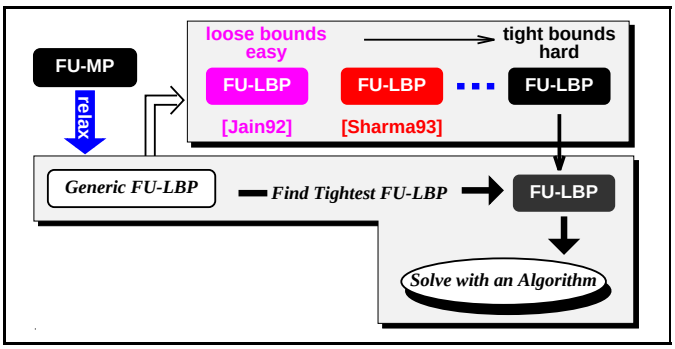

Fig. 3. Voyager's Approach to Solving the FU Lower-Bounding Problem

A.1 Forming FU-LB Problems by Precedence Relaxation

One approach to forming a FU-LB problem from the FU-Min problem is to relax the precedence constraints between operations. The simplest method for relaxing the precedence constraints is to eliminate them completely, as was done by Jain *et al.* in [3]. Although the resulting FU-LB problem can be solved quickly, it can produce very loose bounds, as was later demonstrated by Sharma and Jain in [4]. A similar relaxation was used by Küçükçakar for estimating FU area in [5].

A tighter scheme for relaxing the precedence constraints and solving the resulting FU-LB problem is based on a theorem originally given by Fernández and Bussell in [6, Theorem 1]. Briefly, that technique considers each cstep interval $[s,t]$ between 1 and c, and calculates the minimum load $l^k_{s,t}$ in each interval; this minimum load $l^k_{s,t}$ is computed by moving the operations within their schedule intervals so that they have a minimum possible overlap with $[s,t]$, and summing all the overlaps. A lower bound $\underline{m}'_k$ on the number of functional units for each FU-type $k \in K$ can then be computed as

$$\underline{m}'_k = \max_{[s,t]\subseteq[1,c]} \{\lceil l^k_{s,t}/(t-s+1)\rceil\}. \quad (1)$$

In high-level synthesis, Fernández & Bussell's theorem has been used by Sharma and Jain [4], by Ohm *et al.* [7], and by Hu *et al.* [8], [9] to compute lower bounds on the number of FUs [1]. Rabaey and Potkonjak [11] also produce the same bounds, although using a different method.

As illustrated in Figure 2, both of the approaches described in [3] and [4] first give an algorithm, and then prove that the algorithm indeed produces a lower bound. Therefore, each of these algorithms must be solving a particular FU-LB problem, but we do not know the position of each problem in the range of possible problems, and thus we do not know the quality of the bounds that might be expected.

A.2 Forming FU-LB Problems by LP-Relaxation

Another approach, resulting in a different type of FU-LB problem, is based on the fact that the FU-Min problem can be formulated as an Integer Linear Program (ILP), as shown in the next section. In the worst case, solving an ILP formulation takes exponential running time.

However, an ILP can be relaxed by dropping the integrality restrictions on the variables. This relaxed problem, called the LP-relaxation of the original ILP, can be solved in polynomial time, and provides a lower bound on the solution to the original problem.

This method was used by Gebotys in [1], where the LP-relaxation of the FU-Min problem was used to compute lower bounds on the number of functional units (thus solving a FU-LB problem). In all of her benchmark examples, this method found the exact bounds (i.e., bounds that exactly match the the optimal solution) by solving only a single LP-relaxation, as shown in the column labeled **LP-Relaxation** in Table I. However, there is no guarantee that this method will always produce exact bounds.

A.3 Our Approach – Forming an FU-LB Problem as the Tightest Possible Precedence Relaxation

In Rensselaer's Voyager design space exploration system [12], we take yet another approach, although one also based on precedence relaxations (see the column labeled **Tightest Prec. Relaxation** in Table I). Voyager's solution approach can be summarized as follows (see Figure 3). We begin with a formal description of the FU minimization problem (FU-Min), and form a generic problem by relaxing the precedence constraints. This generic problem concisely describes all possible FU-LB problems based on precedence relaxations, including those solved by the previous methods. We then we select the one FU-LB problem that produces the tightest bound, and solve that FU-LB problem in polynomial time (by solving at most two LPs). Thus Voyager's approach formalizes an entire class of FU lower-bounding problems (those based on precedence relaxations), and is guaranteed to produce the tightest possible

[1] Although their problems were not presented as relaxations, we proved in [10] that the same bounds can be produced by solving the precedence relaxation.

bounds of any precedence relaxation.

It is more difficult to compare Voyager's approach to the LP-relaxation approach. In our experiments (see Section VI), both approaches always found exact bounds. However, in Section V, we show that the bound obtained by rounding up the solution of the LP relaxation of the FU-Min problem may be as tight as the bound produced by Voyager's approach, but is not guaranteed to always be that tight. Thus Voyager's approach provides a guaranteed solution quality, at the cost of solving only one additional LP.

The remainder of this paper is organized as follows. In Section II, a formal description of the FU lower-bounding problem is presented in equation (LBP_k). Although our method for solving (LBP_k) is straightforward, the proof of its correctness requires an elaborate theoretical development which is presented in the next three sections: Section III introduces an extended problem (E_k), Section IV discusses the cost function of (E_k), and its implications that lead to the lower-bounding method described in Section V. Experimental results are presented in Section VI.

II. Formal Description of the FU Lower-Bounding (FU-LB) Problem

This section presents an integer linear programming (ILP) formulation of the FU lower-bounding problem. The constraints of the ILP are similar to those of the scheduling problem, and have been used by several ILP-based schedulers [13], [1], [2].

Given a Data Flow Graph (DFG), let the set of all operations be denoted as $\{o_i \mid i \in I\}$, where $I = \{1, \ldots, n\}$ is the index set of all operations. Let $o_i \rightarrow o_j$ denote a precedence relation, meaning operation i must finish execution before operation j can start.

Suppose the DFG is to be scheduled onto a set $S = \{1, \ldots, c\}$ of control steps. Let $asap_i$ (resp. $alap_i$) denote the as-soon-as-possible (resp. as-late-as-possible) control step (cstep) into which operation o_i can start execution. The control step interval $S_i = [asap_i, alap_i]$ is then referred to as the *schedule interval* of operation o_i.

Let the *type* of a functional unit (FU) indicate its functionality (e.g., multiplication, or addition). Let K be the set of types that are available and let m_k be the number of functional units of type $k \in K$. The types of the operations are determined by the type function $\tau : I \rightarrow K$, where $\tau(i) = k$ means operation o_i is executed on a functional unit of type k. By using the function τ, we have implicitly assumed that each operation can be scheduled on only one type of FU. Thus each FU-type k must execute all operations with index set $I_k = \{\, i \mid i \in I;\ \tau(i) = k \,\}$, and $\{I_k\}$ for $k \in K$ is a partition of I.

The FU Minimization (FU-Min) problem then finds the minimum value of m_k for a particular $k \in K$, and can be formulated as:

$$m_k^* = \min m_k,$$

subject to:

$$\sum_{s \in S_i} x_{i,s} = 1 \qquad \forall\, i \in I, \tag{A}$$

$$\sum_{i \in I_k} x_{i,s} \le m_k \qquad \forall\, s \in S\,,\ \ k \in K, \tag{R}$$

$$\sum_{\substack{s_i \ge s,\ s_i \in S_i \\ s_j \le s,\ s_j \in S_j}} (x_{i,s_i} + x_{j,s_j}) \le 1 \qquad \forall\, s \in S_i \cap S_j,\ \ \forall\, o_i \rightarrow o_j, \tag{P}$$

$$x_{i,s} \in \mathbb{Z}_+ \qquad \forall\, i \in I,\ s \in S_i.$$

Let $\mathbf{M}_a$ be the coefficient matrix due to the assignment constraints (A), $\mathbf{M}_r$ be the coefficient matrix due to the resource constraints (R), and $\mathbf{M}_t$ be the coefficient matrix due to the precedence constraints (P). Then the FU-Min problem can be represented more concisely in the following form:

$$m_k^* = \min \{ m_k \mid \mathbf{M}_t \mathbf{x} \le \mathbf{1}\,;\ \mathbf{x} \in \mathcal{R}_k \}, \tag{MP_k}$$

where

$$\mathcal{R}_k = \{\, \mathbf{x} \in P_F(\mathcal{R}_k) \mid \mathbf{x} \text{ integer} \,\}, \text{ and} \tag{2}$$

$$P_F(\mathcal{R}_k) = \{\, [\mathbf{x}, m_k] \in \mathbb{R}_+^{n+1} \mid \mathbf{M}_a \mathbf{x} = \mathbf{1};\ \mathbf{M}_r \mathbf{x} \le \mathbf{m} \,\}. \tag{3}$$

As mentioned earlier, this problem is similar to the TCS problem, so it is NP-hard [14]. Therefore in order to find the lower bound efficiently, we have to consider a relaxation [15] of (MP_k). A number of different relaxations of (MP_k) are possible, each of which produces a lower bound on m_k^* and represents a valid FU-LB problem. Our goal was to find the *tightest* possible lower bound that could be found by relaxing the precedence constraints of (MP_k).

The following equation presents a generic problem $\mathrm{GLBP}_k(\boldsymbol{\lambda})$ which produces a lower bound on m_k^* for each nonnegative value of $\boldsymbol{\lambda}$:

$$\underline{m}_k(\boldsymbol{\lambda}) = \min \{ m_k + \boldsymbol{\lambda}(\mathbf{M}_t \mathbf{x} - \mathbf{1}) \mid \mathbf{x} \in \mathcal{R}_k \}. \quad \mathrm{GLBP}_k(\boldsymbol{\lambda})$$

where $\boldsymbol{\lambda}$ is a vector of positive real numbers. Note that problem $\mathrm{GLBP}_k(\boldsymbol{\lambda})$ does not explicitly contain the precedence constraints. Instead, they have been included in the objective function with the penalty term $\boldsymbol{\lambda}(\mathbf{M}_t\mathbf{x} - \mathbf{1})$. Since $\boldsymbol{\lambda} \ge \mathbf{0}$, violations of the precedence constraints will make the penalty term positive, and thus intuitively $\mathbf{M}_t\mathbf{x} \le \mathbf{1}$ will be satisfied if $\boldsymbol{\lambda}$ is suitably large.

It can be easily seen that $\underline{m}_k(\boldsymbol{\lambda}) \le m_k^*$ for all $\boldsymbol{\lambda} \ge \mathbf{0}$; in other words, $\underline{m}_k(\boldsymbol{\lambda})$ provides a lower bound on m_k^*. For this reason, we refer to $\mathrm{GLBP}_k(\boldsymbol{\lambda})$ as a generic FU-LB problem. Solving the problem

$$\underline{m}_k^* = \max_{\boldsymbol{\lambda} \ge \mathbf{0}} \underline{m}_k(\boldsymbol{\lambda}) \tag{LBP_k}$$

then gives the largest lower bound $\underline{m}_k^*$ possible of the infinite number of lower bounds $\{\underline{m}_k(\boldsymbol{\lambda}) \mid \boldsymbol{\lambda} \ge \mathbf{0}\}$. Therefore $\underline{m}_k^*$ is the tightest lower bound that can be found by relaxing the precedence constraints.

In the solution of $\mathrm{GLBP}_k(\boldsymbol{\lambda})$, although the elements of $\mathbf{x}$ have to be integral, the elements of $\boldsymbol{\lambda}$ can be fractional.

Therefore it is possible that the value of $\underline{m}_k^*$ will be fractional, in which case $\lceil \underline{m}_k^* \rceil$ is used as the lower bound on the number of resources in any feasible schedule.

For the relaxation approach presented above to be useful, an efficient solution technique is needed to find the value of $\lceil \underline{m}_k^* \rceil$. However, for any $\boldsymbol{\lambda}$, the generic problem $\text{GLBP}_k(\boldsymbol{\lambda})$, in its present form, is not solvable in polynomial time. Furthermore since solving (LBP_k) requires solving $\text{GLBP}_k(\boldsymbol{\lambda})$ several times for different values of $\boldsymbol{\lambda}$, it is hard to compute $\underline{m}_k^*$ by solving (LBP_k) in this manner.

Fortunately, it is possible to compute $\lceil \underline{m}_k^* \rceil$ in polynomial time, without explicitly solving $\text{GLBP}_k(\boldsymbol{\lambda})$ or (LBP_k). The method is described in Section V, and it computes $\lceil \underline{m}_k^* \rceil$ indirectly by solving a different problem (E_k) which is presented in the next section.

III. Forming the Extended Problem (E_k)

A road-map showing the development of Voyager's solution approach is sketched in Figure 4. The previous section defined the problem (LBP_k), which is derived from the problem (MP_k), and showed how solving the problem problem (LBP_k) produces the tightest possible lower bound that can be found by relaxing the precedence constraints of (MP_k). Unfortunately, the problem (LBP_k) is not solvable in polynomial time.

However, instead of directly solving (LBP_k) to compute $\underline{m}_k^*$, it is possible to compute $\lceil \underline{m}_k^* \rceil$ by solving a different problem, which is formulated by transforming (LBP_k). This procedure is also sketched in Figure 4, and is explained in a step-by-step manner in the following sections.

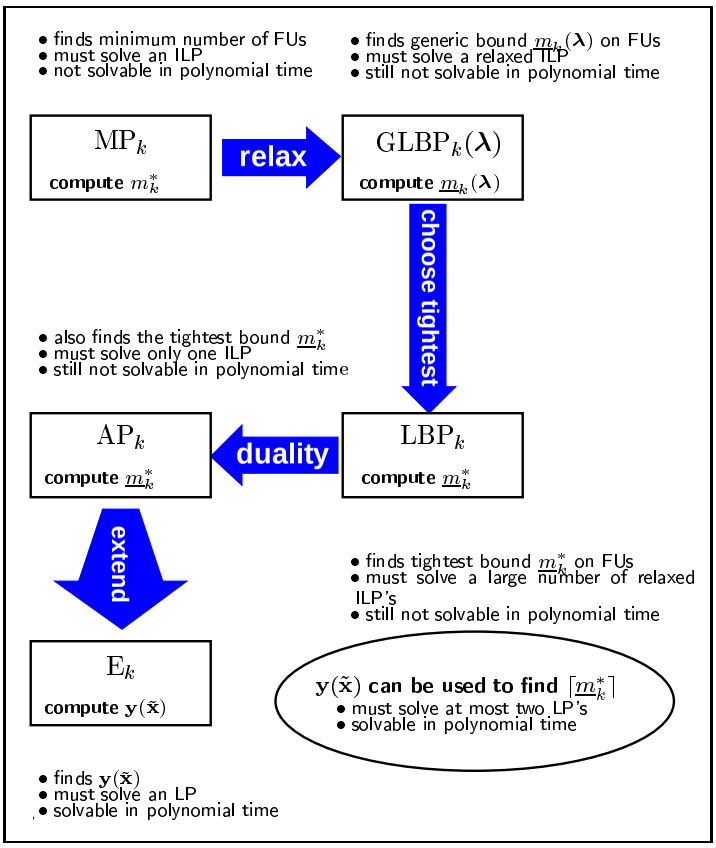

Fig. 4. Road-Map Showing the Development of Voyager's FU Lower-Bounding Solution Method

Step 1: Forming the Alternative Problem

From the description of (LBP_k), it can be seen that in order to compute $\underline{m}_k^*$, it is necessary to compute $\underline{m}_k(\boldsymbol{\lambda})$ for different values of $\boldsymbol{\lambda}$. Thus the problem $\text{GLBP}_k(\boldsymbol{\lambda})$ needs to be solved a number of times; however, for an arbitrary $\boldsymbol{\lambda}$, we do not know of any polynomial-time algorithm to solve $\text{GLBP}_k(\boldsymbol{\lambda})$. To work around this difficulty, we formulate an alternative problem (AP_k) that yields the value of $\underline{m}_k^*$ without solving $\text{GLBP}_k(\boldsymbol{\lambda})$ a number of times.

The description of (LBP_k), as presented in Section II, can be viewed as the *Lagrangian Dual* [15] of (MP_k). Therefore, we can use Proposition 6.2, Chapter II.3 in [15] to conclude that the value of $\underline{m}_k(\boldsymbol{\lambda}^*)$ can be computed by solving an alternative problem as shown below:

$$\underline{m}_k^* = \min \{ \underline{m}_k \mid \mathbf{M}_t \mathbf{x} \leq \mathbf{1} \,;\; \mathbf{x} \in \mathit{conv}(\mathcal{R}_k) \}, \qquad (\text{AP}_k)$$

where *conv* indicates convex hull. Thus we now have only a single problem to solve.

However, even the problem (AP_k), in its present form, is not solvable in polynomial time. Fortunately, as will be shown in the next step, we can transform this problem to a slightly different form so that the solution can be found in polynomial time.

Step 2: Extending the Alternative Problem

Since problem (AP_k), as indicated in the previous section, is not directly solvable in polynomial time, we extend the formulation to a higher dimension so that a polynomial-time solution can be found.

In the extended problem, we introduce additional variables, modify the resource constraints, and use a different objective function. The additional binary variables $y_{s,j}$, $s \in S$, $j = 1, \ldots, \overline{m}_k$ are used to denote that FU j is occupied at control step s. For correctness, the quantity $\overline{m}_k$ must be an upper bound on $\underline{m}_k$; such a bound can be trivially computed as $\overline{m}_k = max_{s \in S} |\{ i \mid s \in S_i \,;\; \tau(i) = k \}|$. However, such a trivial bound leads to a large number of variables, so for practical implementation, we use a simple heuristic to compute a tighter value for $\overline{m}_k$.

The modified resource constraints are given as

$$\sum_{i \in I_k} x_{i,s} - \sum_{j=1}^{\overline{m}_k} y_{s,j} = 0 \,, \quad s \in S, \; j = 1, \ldots, \overline{m}_k. \qquad (\text{R}')$$

The above constraints can be described in terms of their coefficient matrices as $\mathbf{M}_r \mathbf{x} - \mathbf{M}_y \mathbf{y} = \mathbf{0}$. The extended problem is now defined as

$$\min \{ f(\mathbf{y}) \mid \mathbf{M}_t \mathbf{x} \leq \mathbf{1} \,;\; \mathbf{x} \in P_F(\mathcal{R}'_k) \}, \qquad (\text{E}_k)$$

where

$$P_F(\mathcal{R}'_k) = \{ [\mathbf{x}, \mathbf{y}] \in \mathbb{R}_+^{n+p} \mid \mathbf{M}_a \mathbf{x} = \mathbf{1}; \, \mathbf{y} \leq \mathbf{1}; \, \mathbf{M}_r \mathbf{x} = \mathbf{M}_y \mathbf{y} \},$$
$$\mathcal{R}'_k = \{ [\mathbf{x}, \mathbf{y}] \in P_F(\mathcal{R}'_k) \mid [\mathbf{x}, \mathbf{y}] \text{ integer.} \}.$$

The cost function f is linear, and will be discussed in detail in Section IV.

It is easy to see that the extended problem (E_k) is a linear program (LP), and therefore can be solved in polynomial time. Thus although the alternative problem (AP_k) is not polynomially solvable, we are able to extend it to problem (E_k), which can be solved in polynomial time. However, it remains to be shown how the optimal solution of (E_k) can be used to compute the desired bound $\lceil \underline{m}_k^* \rceil$.

Step 3: Using the Solution of (E_k) *to Compute* $\lceil \underline{m}_k^* \rceil$

Unlike the previous problems in the road-map, the extended problem (E_k) does not contain the variable $\underline{m}_k$, and also minimizes a new cost function $f(\mathbf{y})$. Nevertheless, the optimal solution of (E_k) can be used to indirectly compute the value of $\lceil \underline{m}_k^* \rceil$, by solving at most two linear programs. This efficient solution is due to a careful choice of the cost function $f(\mathbf{y})$ for (E_k), which is discussed in the next section. The method for computing $\lceil \underline{m}_k^* \rceil$ using the optimal solution of (E_k) is then presented in Section V.

IV. Cost Function of the Extended Problem (E_k)

As mentioned earlier, our objective is to find a polynomial-time algorithm for the original FU lower-bounding problem (LBP_k). Unfortunately, neither (LBP_k) nor its equivalent, the alternative problem (AP_k), are directly solvable in polynomial time. Therefore, we have taken an indirect approach and have extended (AP_k) to formulate an extended problem (E_k), which is solvable in polynomial time and can be used to compute the desired bound $\lceil \underline{m}_k^* \rceil$.

However, (E_k) does not contain the variable m_k, moreover it minimizes a new cost function $f(\mathbf{y})$. Therefore $f(\mathbf{y})$ must be defined in such a way that minimizing $f(\mathbf{y})$ has a similar effect as minimizing m_k in the original problem (AP_k). In this section, we first define the cost function $f(\mathbf{y})$, and then discuss two important implications of choosing such a cost function for the problem (E_k). Those implications are then used in the Section V to prove that the optimal solution of (E_k) can be used to compute $\lceil \underline{m}_k^* \rceil$.

First, we define the cost function $f(\mathbf{y})$ as

$$f(\mathbf{y}) = \sum_{j=1}^{\overline{m}_k} c_j \sum_{s \in S} y_{s,j}, \tag{4}$$

where

$$c_j = \begin{cases} 0, & j = 1, \ldots, \underline{m}_k, \\ 1 + (\frac{|I_k|}{j-1} - 1) \sum_{i=1}^{j-1} c_i, & j = \underline{m}_k + 1, \ldots, \overline{m}_k \end{cases} \tag{5}$$

The quantity $\underline{m}_k$ in the above equation serves as a preliminary lower bound, and can be computed as $\lceil \frac{|I_k|}{|S|} \rceil$.

The first important implication of $f(\mathbf{y})$ is that when $f(\mathbf{y})$ is used as the cost function, it is sufficient to study only those optimal solutions of (E_k) that satisfy a special form. This special form is given in Lemma 1, and is crucial for proving that an optimal solution of (E_k) leads to the desired bound $\lceil \underline{m}_k^* \rceil$. For a concise description of the special form, we use the notation $[p]$ to denote, for any real number $p \geq 1$, the vector

$$[p]_i = \begin{cases} 1, & i = 1, \ldots, \lfloor p \rfloor, \\ p + 1 - \lceil p \rceil, & i = \lceil p \rceil, \end{cases} \tag{6}$$

where $p \geq 1$.

The second term in the above equation, i.e., the value of $[p]_i$ at $i = \lceil p \rceil$, evaluates to 1 when p is an integer, and to the fractional part of p when p is a fraction.

The special form of a feasible solution of (E_k) is denoted as $[\mathbf{x}, \mathbf{z}(\mathbf{x})]$, and is described in the following Lemma.

Lemma 1: If $[\mathbf{x}, \mathbf{y}]$ is a feasible solution of (E_k), then $[\mathbf{x}, \mathbf{z}(\mathbf{x})]$ is also feasible in (E_k), and $f(\mathbf{z}(\mathbf{x})) \leq f(\mathbf{y})$, where $\mathbf{z}(\mathbf{x})$ is defined as

$$z_{s,j}(\mathbf{x}) = \begin{cases} [u_s(\mathbf{x})]_j, & j = 1, \ldots, \lceil u_s(\mathbf{x}) \rceil, \\ 0, & \text{otherwise}, \end{cases} \tag{7}$$

where

$$u_s(\mathbf{x}) = \sum_{i \in I_k} x_{i,s}. \tag{8}$$

Proof: Given in [16] and [10].

Corollary 1: If $[\tilde{\mathbf{x}}, \tilde{\mathbf{y}}]$ is an optimal solution of (E_k), then $[\tilde{\mathbf{x}}, \mathbf{z}(\tilde{\mathbf{x}})]$ is also an optimal solution of (E_k).

Informally, Lemma 1 and Corollary 1 suggest that, when we are interested in the optimum solutions of (E_k), it is sufficient to consider only the feasible solutions of the form $[\mathbf{x}, \mathbf{z}(\mathbf{x})]$.

Next we present the second important implication of the cost function $f(\mathbf{y})$: the costs of the feasible solutions (of the special form) are not arbitrary, instead they follow a nicely ordered pattern according to their *FU-usage*. The *FU-usage* is defined to be the maximum number of k-type operations performed in any control step, and is expressed as $m(\mathbf{x})$ in the equation

$$m(\mathbf{x}) = \max\{u_s(\mathbf{x}) \mid s \in S\}, \tag{9}$$

where $u_s(\mathbf{x})$ is the number of k-type operations performed in control step s, and is defined in (8).

Although the details will be omitted here in the interest of space, we show in [16] and [10] that when the FU-usages are integers, a higher FU-usage results in a strictly higher cost, and intuitively, minimizing the cost function should have a similar effect as minimizing the FU-usage.

In this section, we have presented the cost function $f(\mathbf{y})$, and have informally explained why minimizing $f(\mathbf{y})$ also produces the effect of minimizing FU-usage $m(\mathbf{x})$. Therefore we can expect to minimize m_k by solving (E_k), and the next section will prove that it is indeed possible to compute $\lceil \underline{m}_k^* \rceil$ by solving (E_k) at most two times.

V. Computing the Lower Bound $\lceil \underline{m}_k^* \rceil$ Using the Extended Problem (E_k)

In this section, we discuss how to compute the intended lower bound $\lceil \underline{m}_k^* \rceil$ by solving the extended problem (E_k).

The bound $\lceil \underline{m}_k^* \rceil$ is defined as the ceiling on the optimal solution of problem (LBP_k) or its equivalent alternative problem (AP_k). It is possible to compute this bound indirectly by solving problem (E_k) only if there exists a correspondence between the feasible solutions of (E_k) and those of (AP_k). This correspondence is established in the following lemmas (proofs are given in [16] and [10]).

Lemma 2: If $[\mathbf{x}, m_k]$ is a feasible solution of (AP_k), then $[\mathbf{x}, \mathbf{z}(\mathbf{x})]$ is a feasible solution of (E_k).

Corollary 2: If $[\mathbf{x}, m_k]$ is a feasible solution of (AP_k), then $f(\mathbf{z}(\mathbf{x})) \geq f(\mathbf{z}(\tilde{\mathbf{x}}))$, where $[\tilde{\mathbf{x}}, \tilde{\mathbf{y}}]$ denotes an optimal solution of (E_k).

Lemma 3: If $[\mathbf{x}, \mathbf{y}]$ is a feasible solution of (E_k), then $[\mathbf{x}, m(\mathbf{x})]$ is a feasible solution of (AP_k).

Corollary 3: If $[\mathbf{x}, \mathbf{y}]$ is a feasible solution of (E_k), then $m(\mathbf{x}) \geq m(\mathbf{x}^*)$, where $[\mathbf{x}^*, \underline{m}_k^*]$ denotes an optimal solution of (AP_k).

Informally, Lemma 2 and 3 suggest: from a feasible solution of (AP_k), it is always possible to construct a feasible solution of (E_k), and vice versa.

Although we have established a correspondence between the *feasible* solutions of (AP_k) and (E_k), it remains to determine the relation between the *optimal* solutions of those problems. This relation is described in the following proposition.

Proposition 4: Let $[\tilde{\mathbf{x}}, \tilde{\mathbf{y}}]$ be an optimal solution of (E_k), and $[\mathbf{x}^*, \underline{m}_k^*]$ be an optimal solution of (AP_k). Then

$$\lceil m(\tilde{\mathbf{x}}) \rceil \geq \lceil \underline{m}_k^* \rceil \geq \lfloor m(\tilde{\mathbf{x}}) \rfloor. \tag{10}$$

Proof: Given in [16] and [10]. It should be mentioned that Corollary 2 and 3, as well as the implications of the cost function presented in Section IV, were used to prove the above result.

It would be useful to make one further observation before we present our method for computing $\lceil \underline{m}_k^* \rceil$. It is easy to see that for an optimal solution $[\mathbf{x}^*, \underline{m}_k^*]$ of (AP_k), at least $m(\mathbf{x}^*)$ FUs are needed to satisfy the resource constraints (R). Therefore we can write

$$m(\mathbf{x}^*) = \underline{m}_k^*. \tag{11}$$

We are now in a position to explain how to compute $\lceil \underline{m}_k^* \rceil$ using the solution of the extended problem (E_k). The method is described in Figure 5; a detailed explanation of each step, as well as the proof of the method's correctness, are presented in the following.

1. Solve (E_k) with an LP-solver in order to compute $[\tilde{\mathbf{x}}, \tilde{\mathbf{y}}]$. It then immediately follows from Proposition 4 that $\lceil \underline{m}_k^* \rceil$ is either $\lceil m(\tilde{\mathbf{x}}) \rceil$ or $\lfloor m(\tilde{\mathbf{x}}) \rfloor$.

2a. If $m(\tilde{\mathbf{x}})$ is integer, since $\lceil m(\tilde{\mathbf{x}}) \rceil = \lfloor m(\tilde{\mathbf{x}}) \rfloor = m(\tilde{\mathbf{x}})$, we can be certain that $m(\tilde{\mathbf{x}}) = \lceil \underline{m}_k^* \rceil$, and is the desired bound.

```
1)   Compute [x̃, ỹ] by solving (E_k);
2a)  if m(x̃) integer then
         ⌈m*_k⌉ = m(x̃);
     else
2b)      set y_{s,j} = 0, for j = ⌊m(x̃)⌋ + 1, ..., m̄_k, ∀ s ∈ S
           and solve (E_k) again;
         if (E_k) yields a solution then
             ⌈m*_k⌉ = ⌊m(x̃)⌋;
         else
             ⌈m*_k⌉ = ⌈m(x̃)⌉;
     end;
```

Fig. 5. Method for Computing $\lceil \underline{m}_k^* \rceil$

2b. If $m(\tilde{\mathbf{x}})$ is fractional, set $y_{s,j} = 0$, for $j = \lfloor m(\tilde{\mathbf{x}}) \rfloor + 1, \ldots, \overline{m}_k$, $\forall\, s \in S$ and solve (E_k) one more time. It can be easily verified from (7) and (9), that in this case (E_k) will yield a solution only if it has a feasible solution $[\mathbf{x}, \mathbf{y}]$ such that $\lceil m(\mathbf{x}) \rceil \leq \lfloor m(\tilde{\mathbf{x}}) \rfloor$.

Suppose (E_k) yields a solution $[\mathbf{x}', \mathbf{y}']$. The argument presented in the previous paragraph implies $\lceil m(\mathbf{x}') \rceil \leq \lfloor m(\tilde{\mathbf{x}}) \rfloor$. Furthermore by Lemma (3), $\lceil m(\mathbf{x}') \rceil \geq \lceil \underline{m}_k^* \rceil$. Combining these two inequalities: $\lceil \underline{m}_k^* \rceil \leq \lfloor m(\tilde{\mathbf{x}}) \rfloor$, which in conjunction with (10) leads to the conclusion $\lceil \underline{m}_k^* \rceil = \lfloor m(\tilde{\mathbf{x}}) \rfloor$.

On the contrary, if (E_k) does not yield a solution, then we conclude $\lceil \underline{m}_k^* \rceil = \lceil m(\tilde{\mathbf{x}}) \rceil$. The proof can be given by contradiction. Let us assume $\lceil \underline{m}_k^* \rceil = \lfloor m(\tilde{\mathbf{x}}) \rfloor$; substituting (11), we obtain $\lceil m(\mathbf{x}^*) \rceil = \lfloor m(\tilde{\mathbf{x}}) \rfloor$. At this point we can invoke Lemma 2 and conclude that (E_k) has a feasible solution $[\mathbf{x}^*, \mathbf{z}(\mathbf{x}^*)]$ such that $\lceil m(\mathbf{x}^*) \rceil = \lfloor m(\tilde{\mathbf{x}}) \rfloor$. Then, however, ($E_k$) must yield a solution in the second run, which contradicts our initial assumption.

Thus, in both cases where $m(\tilde{\mathbf{x}})$ is fractional, the value of $\lceil \underline{m}_k^* \rceil$ is determined after solving (E_k) a second time. The method above demonstrates that the lower bound $\lceil \underline{m}_k^* \rceil$ on the number of FUs of type k can be computed in polynomial time by solving the linear program (E_k) at most 2 times.

Let us now consider the theoretical quality of the lower bounds $\lceil \underline{m}_k^* \rceil$ as compared to previous approaches based on precedence relaxation. As discussed in Section II, the FU lower-bounding problem (LBP_k) was formulated in such a way that it produces the tightest lower bound $\underline{m}_k^*$ of all possible lower bounds $\{\underline{m}_k(\boldsymbol{\lambda}) \mid \boldsymbol{\lambda} \geq \mathbf{0}\}$ that can be found by relaxing the precedence constraints. The lower bounds by Sharma and Jain [4] are also computed by relaxing the precedence constraints, and we have shown in [10] that the algorithm in [4] computes $\underline{m}_k(\mathbf{0})$ by solving the problem $GLBP_k(\boldsymbol{\lambda})$ for $\boldsymbol{\lambda} = \mathbf{0}$. Since $\lceil \underline{m}_k^* \rceil \geq \underline{m}_k(\mathbf{0})$, our approach will produce bounds that are at least as tight, if not tighter, than the bounds in [4]. We have also shown in [10] that the bounds produced by Rabaey and Potkonjak's approach [11] are the same as those of Sharma and Jain [4], so again, our approach will produce bounds that

Schedule Length	Loop Length	(+, ∗)		
		Optimal	Voyager	Sharma
17	17	(3,3)	(3,3)	(3,3)
18	18	(2,2)	(2,2)	(2,2)
18	16	(3,2)	(3,2)	(2,2)
19	19	(2,2)	(2,2)	(2,2)
19	17	(2,2)	(2,2)	(2,2)
21	21	(2,1)	(2,1)	(2,1)
21	19	(2,1)	(2,1)	(2,1)

TABLE II
NUMBER OF FUS FOR THE EWF (USING NON-PIPELINED MULTIPLIER)

Schedule Length	Loop Length	(+, ⊛)		
		Optimal	Voyager	Sharma
17	17	(3,2)	(3,2)	(3,2)
18	18	(3,1)	(3,1)	(2,1)
18	16	(3,1)	(3,1)	(2,1)
19	19	(2,1)	(2,1)	(2,1)
19	17	(2,1)	(2,1)	(2,1)
21	21	(2,1)	(2,1)	(2,1)
21	19	(2,1)	(2,1)	(2,1)

TABLE III
NUMBER OF FUS FOR THE EWF (USING PIPELINED MULTIPLIER)

are at least as tight, if not tighter, than the bounds in [11].

Next we consider the solution of the linear programming relaxation of (MP_k), which also provides a lower bound (m_k^{LP}) on m_k^*. The LP relaxation of (MP_k) is described as

$$m_k^{LP} = \min\{\, m_k \mid \mathbf{M}_t\mathbf{x} \leq \mathbf{1};\ \mathbf{x} \in P_F(\mathcal{R}_k)\,\}. \qquad (\mathrm{LP}_k)$$

The feasible region, $\{\, \mathbf{M}_t\mathbf{x} \leq \mathbf{1};\ \mathbf{x} \in P_F(\mathcal{R}_k)\,\}$, of ($\mathrm{LP}_k$) looks very similar to the feasible region, $\{\, \mathbf{M}_t\mathbf{x} \leq \mathbf{1};\ \mathbf{x} \in conv(\mathcal{R}_k)\,\}$, of the alternative problem (AP_k). However, since $conv(\mathcal{R}_k) \subseteq P_F(\mathcal{R}_k)$ (from (2) and (3)), the feasible region of (AP_k) is a subset of the feasible region (LP_k). Therefore, the bound produced by (AP_k) must be as tight or tighter than the bound produced by (LP_k), i.e., $m_k^{LP} \leq \underline{m}_k^*$. This implies $\lceil m_k^{LP} \rceil \leq \lceil \underline{m}_k^* \rceil$, which means that the bound obtained by rounding up the solution of the LP relaxation of (MP_k) may be as tight as the bound produced by Voyager's approach, but is not guaranteed to always be that tight.

VI. EXPERIMENTS AND RESULTS

Experiments were run on the Elliptic Wave Filter [17, p.206] (EWF) and Discrete Cosine Transform [18] (DCT) benchmarks. The results are listed in Table II, III, and IV, respectively. Given the time constraint c listed in the first column, the bounds computed by Voyager's FU lower-bounding method are reported in the column labeled **Voyager**, and the minimum number of FUs for which a feasible schedule was found by Voyager's scheduler [12], [2] are reported in the column labeled **Optimal**. These results are compared to the lower bounds produced by applying the technique of [4], which are reported under the heading **Sharma** [4].

Time=500ns		(+, −, ∗)		
Clock	Csteps	EXACT	LBND	SHARMA
56	8	(7, 4, 11)	(7, 4, 11)	(5, 4, 11)
55	9	(5, 6, 15) (6, 4, 13)	(5, 6, 15) (6, 4, 13)	(5, 4, 13)
48	10	(5, 6, 16)	(5, 6, 16)	(5, 4, 16)
33	15	(7, 4, 11) (5, 4, 15)	(7, 4, 11) (5, 4, 15)	(5, 4, 11)
28	15	(7, 4, 11) (5, 4, 15)	(7, 4, 11) (5, 4, 15)	(5, 4, 11)
24	20	(4, 4, 11)	(4, 4, 11)	(4, 4, 11)
21	23	(7, 4, 11) (5, 4, 15)	(7, 4, 11) (5, 4, 15)	(5, 4, 11)
19	26	(7, 4, 11) (5, 4, 15)	(7, 4, 11) (5, 4, 15)	(5,4, 11)

TABLE IV
NUMBER OF FUS FOR THE DCT (USING A TIME CONSTRAINT OF 500NS)

For the EWF benchmark, the standard assumptions were made, i.e., that an adder (denoted as +) takes one control step, a multiplier (denoted as ∗) takes two control steps, and a pipelined multiplier (denoted as ⊛) has a latency 1. The bounds for non-pipelined multipliers are given in Table II, and for pipelined multipliers are reported in Table III. In all cases, the bounds on the number of FUs produced by our method (**Voyager**) were as tight as possible (i.e., Voyager's scheduler could find feasible schedules that satisfy those bounds). However, the bounds produced by the algorithm of [4] were occasionally loose, as illustrated in the shaded rows.

A different kind of experiment was run on the DCT benchmark. The VDP100 module library [19] was used, where a register transfer involving an adder (denoted as +), a subtracter (denoted as −), and a multiplier (denoted as ∗) takes 48ns, 56ns, and 163ns respectively. For a time constraint of 500ns, the bounds were computed for eight different clock lengths as shown in Table IV. Again, the bounds produced by Voyager's approach were always as tight as possible, and the bounds produced by Sharma's approach were loose in several cases.

For all these experiments, we used the LINDO LP-solver on a SPARCstation 2. For the EWF example, each set of bounds in Table II took between 2.2 and 9.7 CPU seconds to generate, and each set of bounds in Table III was generated in less than 1.5 CPU seconds. The DCT experiment, reported in Table IV, comprised much larger problem instances (particularly as the clock length became smaller), and required between 3.3 and 23.03 CPU seconds to generate each set of bounds. Although Sharma's algorithm is theoretically faster , with average run times [4] of 1.68 CPU seconds for examples such as the EWF, we have shown, both theoretically and experimentally, that our method produces bounds with higher accuracy, and does so with acceptable run times.

In Section V, we proved the bound obtained by rounding up the solution of the LP relaxation of the FU-Min problem may be as tight as the bound produced by Voyager's approach, but is not guaranteed to always be that tight.

However, in all the experiments that we have run so far, both approaches always found exact bounds.

VII. Summary and Future Work

This paper has presented a formal description of the FU lower-bounding problem that produces the tightest possible bounds that can be found by relaxing either the precedence constraints or the integrality constraints on the scheduling problem. Although this problem is not directly solvable in polynomial time, we have presented an extended problem that can be solved in polynomial time and can be used to indirectly find the same bounds as the original problem.

References

[1] C. H. Gebotys and M. I. Elmasry, *Optimal VLSI Architectural Synthesis*. Kluwer international series in engineering and computer science; VLSI, computer architecture, and digital signal processing, 101 Philip Drive, Assinippi Park, Norwell, MA 02061: Kluwer Academic Publishers Group, 1992.

[2] S. Chaudhuri, R. A. Walker, and J. E. Mitchell, "Analyzing and Exploiting the Structure of the Constraints in the ILP Approach to the Scheduling Problem," *IEEE Transactions on VLSI Systems*, vol. 2, pp. 456–471, Dec. 1994.

[3] R. Jain, A. C. Parker, and N. Park, "Predicting System-Level Area and Delay for Pipelined and Nonpipelined Designs," *IEEE Transactions on Computer-Aided Design*, vol. 11, pp. 955–965, Aug. 1992.

[4] A. Sharma and R. Jain, "Estimating Architectural Resources and Performance for High-Level Synthesis Applications," *IEEE Transactions on VLSI Systems*, vol. 1, pp. 175–190, June 1993.

[5] K. Küçükçakar, *System-Level Synthesis Techiques with Emphasis on Partitioning and Design Timing*. PhD thesis, Electrical Engineering – Systems Department, University of Southern California, 1991.

[6] E. B. Fernández and B. Bussell, "Bounds on the number of Processors and Time for Multiprocessor Optimal Schedule," *IEEE Transactions on Computers*, vol. C-22, pp. 745–751, Aug. 1973.

[7] S. Y. Ohm, F. J. Kurdahi, and N. Dutt, "Comprehensive Lower Bound Estimation from Behavioral Descriptions," in *Proc. of the IEEE/ACM International Conference on Computer-Aided Design*, (San Jose, California), pp. 182–187, IEEE Computer Society Press, Nov. 6-10 1994.

[8] Y. Hu, A. Ghouse, and B. S. Carlson, "Lower Bounds on the Iteration Time and the number of Resources for Functional Pipelined Data Flow Graphs," in *Proc. of the IEEE International Conference on Computer Design*, (Cambridge, Massachusetts), pp. 21–24, IEEE Computer Society Press, Oct. 3-6 1993.

[9] Y. Hu and B. S. Carlson, "Improved Lower Bounds for the Scheduling Optimization Problem," in *Proc. of 1994 IEEE International Symp. on Circuits and Systems.*, (London, England), pp. 295–298, IEEE Computer Society Press, May 30-June 2 1994.

[10] S. Chaudhuri, *Scheduling and Design Space Exploration in High-Level Synthesis*. PhD thesis, Electrical, Computer, and Systems Engineering – Rensselaer Polytechnic Institute, 1995.

[11] J. M. Rabaey and M. Potkonjak, "Estimating Implementation Bounds for Real Time DSP Application Specific Circuits," *IEEE Transactions on Computer-Aided Design*, vol. 13, pp. 669–683, June 1994.

[12] S. Chaudhuri, S. A. Blythe, and R. A. Walker, "An Exact Methodology for Scheduling in a 3D Design Space," in *Proc. of the 8th International Symposium on System-Level Synthesis*, (Cannes, France), p. to appear, IEEE Computer Society Press, Sept. 13-15 1995.

[13] C.-T. Hwang, J.-H. Lee, and Y.-C. Hsu, "A Formal Approach to the Scheduling Problem in High-Level Synthesis," *IEEE Transactions on Computer-Aided Design*, vol. 10, pp. 464–475, Apr. 1991.

[14] J. D. Ullman, "NP-Complete Scheduling Problems," *J. Comput. System Sci*, vol. 10, no. 10, pp. 384–393, 1975.

[15] G. L. Nemhauser and L. A. Wolsey, *Integer and Combinatorial Optimization*. Wiley-Interscience series in discrete mathematics, 605 Third Avenue, New York, NY 10158-0012, USA: Wiley-Interscience, 1988.

[16] S. Chaudhuri and R. A. Walker, "Computing Lower Bounds on Functional Units before Scheduling," Tech. Rep. 95–13, CS Dept, Rensselaer Polytechnic Institute, September 1995.

[17] D. E. Thomas, E. D. Lagnese, R. A. Walker, J. A. Nestor, J. V. Rajan, and R. L. Blackburn, *Algorithmic and Register Transfer Level Synthesis: The System Architect's Workbench*. 101 Philip Drive, Assinippi Park, Norwell, MA 02061: Kluwer Academic Publishers Group, 1990.

[18] J. A. Nestor and G. Krishnamoorthy, "SALSA: A New Approach to Scheduling with Timing Constraints," *IEEE Transactions on Computer-Aided Design*, vol. 12, pp. 1107–1122, Aug. 1993.

[19] VLSI Technologies Inc., *VDP100 1.5 Micron CMOS Datapath Cell Library*, 1988.