Computing Lower Bounds on Functional Units before Scheduling[*]

Samit Chaudhuri[†] ***Robert A. Walker***[‡†]
Rensselaer Polytechnic Institute
Troy, NY 12180

Abstract

This paper presents a new algorithm for computing lower bounds on the number of functional units (FU's) required to schedule a data flow graph in a specified number of control steps. We use a formal approach to compute the bounds that can can be proven to be tighter than those produced by existing methods, and that considers the interdependencies of the bounds on the different FU-types. This quick yet accurate estimation of the number of FU's is used to generate resource constraints for a design, and thus reduce the design space.

1 Introduction

In this paper, we discuss the problem of computing the minimum number of functional units (FU) required to schedule a data flow graph in a given number of control steps. The general problem is NP-hard; however, we present an efficient methodology that produces provably tight lower bounds on the number of FU's.

Computing tight lower bounds on the number of FU's is important for several reasons. Time-constrained scheduling algorithms have to explore a large search space in order to find the schedule with minimum area. Tight bounds on the number of FU's can be used as resource constraints to quickly narrow down the design space, thus enabling the scheduling algorithm to produce faster results. Furthermore, the lower bounds provide information on the absolute quality of a design produced by heuristic algorithms [7].

Previous work on computing lower bounds in high-level synthesis can be found in [5] and in [7] (which reports the tightest bound so far). Our method differs from [7] in two aspects. First, we present a formal approach to the problem and compute bounds that are provably tighter. Second, the bounds given in [7] are computed independently for each type of FU, eg. a for the adders and m for the multipliers. It is then concluded that any feasible design has less than a adders and m multipliers, leading to a lower bound curve illustrated using dotted lines in Figure 1. However, due to precedence constraints between operators, the lower bounds on the adders and the multipliers may not be independent. More realistically, there are two bounds (a, m') and (a', m), which will lead to a tighter lower bound curve denoted by the solid line in Figure 1. Our approach considers such interdependencies of bounds, and thus produces a more accurate bound on the feasible design space.

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

[†]Dept. of Electrical, Computer, and Systems Engineering.
[‡]Department of Computer Science.

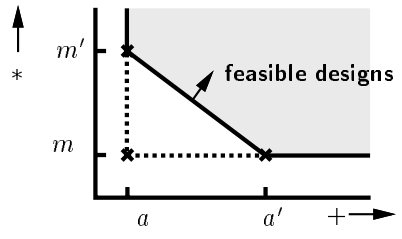

Figure 1: Lower-Bounding the Design Space. The Solid Line Gives Tighter Bounds than the Dotted Line

2 Formal Description of the Lower-Bounding Problem

In this section we present an integer linear programming (ILP) formulation of the lower-bounding problem. The constraints of the ILP are similar to that of the scheduling problem, and have been used by several ILP-based schedulers [4, 3, 2].

Given a cdfg, let I be the index set of all operators, and let $o_i \rightarrow o_j$ indicate a precedence relation, meaning operator i must finish execution before operator j can start. Suppose the cdfg is to be scheduled onto a set S of control steps. As-soon-as-possible (ASAP) and as-late-as-possible (ALAP) schedules give a continuous range S_i of control steps, called the *schedule interval*, over which an operator o_i can be scheduled.

The *type* of a functional unit (FU) indicates 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 type of the operators are determined by the type function $\tau : I \rightarrow K$, where $\tau(i) = k$ means operator o_i is executed on a functional unit of type k. By using the function τ, we have implicitly assumed that each operator can be scheduled on only one type of FU. Thus each FU-type k must execute all operators with index set $I_k = \{\, i | i \in I;\ \tau(i) = k \,\}$, and $\{I_k\}$ for $k \in K$ is a partition of I.

The lower-bounding subproblem finds a lower bound on m_k for a particular $k \in K$, and can be formulated as:

$$min \quad m_k$$

$$\sum_{s \in S_i} x_{is} = 1 \qquad \forall\, i \in I \qquad \text{(A)}$$

$$\sum_{i \in I_k} x_{is} \leq m_k \qquad \forall\, s \in S,\ k \in K \qquad \text{(R)}$$

$$\sum_{\substack{s_i \geq s,\ s_i \in S_i \\ s_j \leq s,\ s_j \in S_j}} x_{is_i} + x_{js_j} \leq 1 \qquad \begin{array}{l} \forall\, s \in S_i \cap S_j \\ \forall\, o_i \rightarrow o_j \end{array} \qquad \text{(P)}$$

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

Note that, unlike previous formulations, the interdependencies between the bounds are captured by our model. Although the bounds on the FU's are computed one type at a time, the bound on one type can affect the bounds on other types. For example, while computing the bound on $m_{k'}$, if we fix the values of m_k, $k \neq k'$ in constraint (R), then those values will affect the bound indirectly through the precedence constraints in (P).

Let M_a be the coefficient matrix due to the assignment constraints (A), M_r be the coefficient matrix due to the resource constraints (R), and M_p be the coefficient matrix due to the precedence constraints (P). Then the above problem can be represented more concisely in the following form:

$$(\mathsf{IP}_k) \qquad m_k^* = \min\{m_k \mid M_p x \leq 1\,;\; x \in Q\}$$

where

$$Q = \{\, x \in \mathbb{Z}_+^n \mid M_a x = 1\,;\; M_r x \leq m \,\}$$

The problem (IP_k) is similar to the time-constrained-scheduling problem, so it is NP-hard [8]. In order to find the lower bound efficiently, we have to consider a relaxation [6] of (IP_k). A number of different relaxations of (IP_k) are possible, each of which produces a lower bound on m_k^*. We want to find the *tightest* lower bound that can be found from all different relaxations.

In the following paragraphs, we first present a problem $\mathsf{RP}_k(\lambda)$ which produces a lower bound on m_k^* for each nonnegative value of λ:

$$\mathsf{RP}_k(\lambda) \qquad r_k(\lambda) = \min\{m_k + \lambda(M_p x - 1) \mid x \in Q\}$$

where λ is a vector of positive real numbers. Note that the above problem does not contain the precedence constraints. Instead, we have included them in the objective function with the penalty term $\lambda(M_p x - 1)$. Since $\lambda \geq 0$, violations of the precedence constraints will make the penalty term positive, and thus intuitively $M_p x \leq 1$ will be satisfied if λ is suitably large.

It can be easily seen that $r_k(\lambda) \leq m_k^*$ for all $\lambda \geq 0$; in other words, $r_k(\lambda)$ provides a lower bound on m_k^*. If we solve the following problem:

$$(\mathsf{LD}_k) \qquad \underline{m}_k^* = \max_{\lambda \geq 0} r_k(\lambda)$$

then we can find the greatest lower bound $\underline{m}_k^*$ available from the infinite number of lower bounds $\{r_k(\lambda) \mid \lambda \geq 0\}$. Therefore, $\underline{m}_k^*$ will produce the tightest lower bound that can be found by relaxing the precedence constraints.

In the solution of $\mathsf{RP}_k(\lambda)$, although the elements of x have to be integral, the elements of λ can be fractional. Therefore, it is possible that the value of $\underline{m}_k^*$ will be fractional, in which case we can use $\lceil \underline{m}_k^* \rceil$ as the lower bound on the number of resources in any feasible schedule.

Note that, we have not ignored the precedence constraints; instead, we have moved them to the objective function. Therefore, our model will still consider the interdependencies of the bounds. It can be proven that $\lceil \underline{m}_k^* \rceil$ gives a tighter bound than the bound reported in [7], or the bound given by the LP-relaxation of (IP_k), but we will not do so here in the interest of space. The complete proof is given in [1].

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 λ, the relaxation problem $\mathsf{RP}_k(\lambda)$, in its present form, is not solvable in polynomial time. Therefore, the optimization problem (LD_k) is even harder to solve.

Fortunately, it is possible to compute $\lceil \underline{m}_k^* \rceil$ in polynomial time, without explicitly solving $\mathsf{RP}_k(\lambda)$ or (LD_k). The method is described in Section 6, and it computes $\lceil \underline{m}_k^* \rceil$ indirectly by solving a different problem (E_k).

Although our method is straightforward, the proof of its correctness requires an elaborate theoretical development which is presented in the following three sections: Section 3 introduces the problem (E_k), Section 4 discusses how to solve (E_k) in polynomial time, and Section 5 indicates an important property of the solution of (E_k) that leads to the method described in Section 6.

3 *Transforming Problem* (LD_k)

The previous section has stated that $\mathsf{RP}_k(\lambda)$, in its present form, is not solvable in polynomial time, which implies that (LD_k) is not solvable in polynomial time either. However, instead of directly solving (LD_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 (LD_k). This procedure is explained in a step-by-step manner in the following sections.

3.1 *Step 1: Forming the Alternative Problem*

The description of (LD_k), as presented in Section 2, can be viewed as the *Lagrangian Dual* [6] of IP_k. Therefore, we can use Proposition 6.2, Chapter II.3 in [6] to conclude that the value of $r_k(\lambda^*)$ can be computed by solving an alternative problem as shown below:

$$(\mathsf{AP}_k) \qquad \underline{m}_k^* = \min\{\, m_k \mid M_p x \leq 1\,;\; x \in \mathit{conv}(Q)\,\}$$

where *conv* indicates convex hull. 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.

3.2 *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. Let:

> $\bar{m}_k$ be the largest number of type-k FU's that will ever be required for any feasible schedule. For correctness of formulation, $\bar{m}_k$ has to be equal to an upper bound for m_k; this can always be ensured by making $\bar{m}_k = max_{s \in S} |\{\, i \mid s \in S_i\,;\; \tau(i) = k \,\}|$.

In the extended problem, we introduce additional variables, modify the resource constraints, and use a different objective function. The additional binary variables y_{sj}, $s \in S$, $j = 1, \ldots, \bar{m}_k$ are used to denote that FU j is occupied at control step s. The modified resource constraints are shown below:

$$\sum_{i \in I_k} x_{is} - \sum_{j=1}^{\bar{m}_k} y_{sj} = 0\,, \quad s \in S,\ j \in \bar{m}_k \qquad (\mathrm{R}')$$

The above constraints can be described in terms of their coefficient matrices as $M_r x - M_y y = 0$. The extended problem is now defined as:

$$(\mathsf{E}_k) \qquad \min \{ f(y) \mid M_p x \leq 1\,;\ x \in conv(Q') \}$$

where

$$\begin{aligned} Q' &= \{ [x, y] \in Q'' | x, y \text{ integer} \} \\ Q'' &= \{ x \in \mathbb{R}^n_+,\ y \in \mathbb{R}^p_+ \\ &\quad \mid M_a x = 1\,;\ y \leq 1\,;\ M_r x - M_y y = 0 \} \end{aligned}$$

The cost function f is is given below:

$$f(y) = \sum_{j=1}^{\bar{m}_k} c_j \sum_{s \in S} y_{sj} \qquad (1)$$

$$\text{where } c_j = \begin{cases} 0 & j = 1, \ldots, l-1 \\ 1 & j = l \\ \displaystyle\sum_{i=l+1}^{j} (|S|+1)^i & j > l \end{cases} \qquad (2)$$

$$\text{where } l = \left\lceil \frac{|I_k|}{|S|} \right\rceil$$

In the following sections we will show that (E_k) can be solved in polynomial time, and can be used to compute the value of $\lceil \underline{m}^*_k \rceil$.

4 Solving the Extended Problem (E_k) in Polynomial Time

In this section, we will show that the extended problem (E_k) is solvable in polynomial time. The proof relies on demonstrating that (E_k) is a *linear program* (LP). Since an LP can be solved in polynomial time, such a demonstration will imply polynomial time solvability of (E_k).

The constraints of an LP must consist of only linear equality and inequality constraints (i.e., no integrality restrictions on the variables). Therefore, in order to solve problem (E_k) as an LP, we must describe $conv(Q')$ using only equality and inequality constraints.

In the following we will prove that $conv(Q') = Q''$. Since Q'' is described only using linear equality and inequality constraints, this result will imply that (E_k) can be solved as an LP.

Lemma 1 *The coefficient matrix* $A = \begin{bmatrix} M_a & 0 \\ 0 & I \\ M_r & -M_y \end{bmatrix}$ *describing the constraints of* Q'' *is totally unimodular (TU).*

Proof: The rows of A can be partitioned in the following manner:

$$\left[\begin{array}{cc} M_a & 0 \\ \hline 0 & I \\ M_r & -M_y \end{array}\right]$$

Each column of the above matrix contains exactly two nonzero entries. If the nonzero entries are of the same sign then the corresponding rows are contained in different partitions. If the nonzero entries are of the same sign then the corresponding rows are contained in the same partition. This is a sufficient condition for a matrix to be TU [6]. □

Proposition 1 *The polyhedron* Q'' *is integral, i.e.* $Q'' = conv(Q')$

Proof: It is a well-known result that if A is TU then $P = \{x \in \mathbb{R}^n_+ | Ax \leq b\}$ is integral when the elements of b are integral [6]. When this fact is applied to the description of Q'' along with the result of the previous lemma, the proof is obvious. □

In the above proposition we have shown that $conv(Q')$ can be described using only linear equality and inequality constraints. Therefore, problem (E_k) can be solved as a linear program in polynomial time.

5 Properties of the Solution of the Extended Problem (E_k)

In the previous section, we have proven that an optimal solution of (E_k) can be found in polynomial time. In this section we will present an important property satisfied by the optimal solutions of (E_k), which allows us to use such a solution to compute $\lceil \underline{m}^*_k \rceil$, the lower bound on the number of FU's of type k in any feasible solution. The main result is presented in Section 5.2; but first we need to introduce some definitions and preliminary results.

5.1 FU-patterns and their Properties

In this section, we introduce some definitions and preliminary results that we will use in the next section to present a method for computing $\lceil \underline{m}^*_k \rceil$.

Definition 1 An *FU-pattern* is a vector $u = [u_1, \ldots, u_{|S|}]$, where u_s, $s \in S$ denotes the number of operations performed on type-k FU's at control step s. Formally, the set U of all possible FU-patterns is given by:

$$U = \{ u \in \mathbb{R}^{|S|}_+ \mid \sum_{s \in S} u_s = |I_k| \}$$

The cost of an FU-pattern is computed, using the cost function defined in (1), as $f(y(u))$ where $y(u)$ is defined as follows:

$$y_{s,j}(u) = \begin{cases} 1 & j = 1, \ldots, \lfloor u_s \rfloor \\ u_s - \lfloor u_s \rfloor & j = \lfloor u_s \rfloor + 1 \\ 0 & \text{otherwise} \end{cases} \qquad (3)$$

$$m(u) = \max_{s \in S} u_s \qquad (4)$$

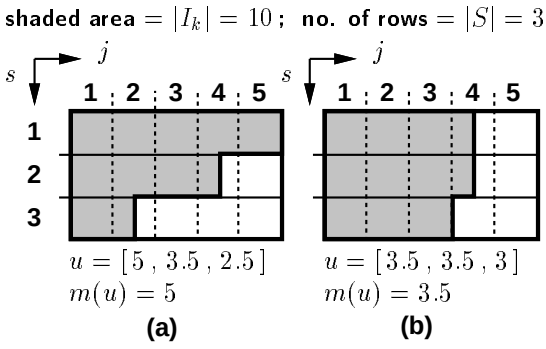

Figure 2: FU-pattern

If we divide a rectangular area into rows and columns of unit width and index them with s and j respectively, then an FU-pattern can be visualized as a shaded region of area $|I_k|$ occupying at most $|S|$ rows. In Figure 2 (a) and (b) we show two possible FU-patterns with $|I_k| = 10$ and $|S| = 3$. The value of u_s is denoted by the area of the shaded region in row s, the value of $m(u)$ is the area of the widest row, and the value of $y_{s,j}(u)$ is denoted by the area of the shaded region in the corresponding (s, j) block. For example, in Figure 2 (a), we can see $y_{2,3}(u) = 1$ because it is completely shaded, and $y_{2,4}(u) = 0.5$ because half of its area is shaded.

The cost for column j is c_j, and the cost of an FU-pattern is computed by multiplying the area of the shaded region in each column with the corresponding cost, and then summing those products. For example, the cost of the FU-pattern in Figure 2 (a) is $3c_1 + 2.5c_2 + 2c_3 + 1.5c_4 + c_5$ where the values of c_j can be found from (2).

For our problem, we are interested in some properties satisfied by the maximum and minimum costs of any possible FU-pattern with $m(u) = p$. Formally, the costs can be defined as follows:

$$\begin{aligned} f_{max}(p) &= \max\{\, f(y(u)) \mid u \in U \,;\; m(u) = p\} \\ f_{min}(p) &= \min\{\, f(y(u)) \mid u \in U \,;\; m(u) = p\,\} \end{aligned}$$

These maximum and minimum costs denote bounds on the cost of an FU-pattern u as follows:

$$f_{max}(m(u)) \geq f(y(u)) \geq f_{min}(m(u)) \,, \quad \forall\, u \in U \qquad (5)$$

In the following Lemma we present some properties satisfied by $f_{max}(p)$ and $f_{min}(p)$:

Lemma 2 *$f_{min}(p)$ and $f_{max}(p)$, $p \geq \left\lceil \frac{|I_k|}{|S|} \right\rceil$, satisfy the following properties:*

$$\begin{aligned} f_{min}(p) &\geq f_{min}(p') && \text{for } p \geq p' && (6) \\ f_{max}(p) &\geq f_{max}(p') && \text{for } p \geq p' && (7) \\ f_{min}(p) &> f_{max}(p-1) && \text{for integer } p && (8) \end{aligned}$$

Proof: In proving the properties above, we will view the FU-patterns as shaded regions in the manner explained earlier in this section.

Property (6): Let u be an FU-pattern of cost $f_{min}(p)$. We will show in the following paragraph, that from u, we can construct a new FU-pattern u' of no greater cost, such that $m(u') = p'$, $p' \leq p$. For the construction to be valid, the shaded region corresponding to u' should have the same area and same number of rows as u.

The rows in u that are wider than p' are first made narrower so that, in the new FU-pattern u', they all have width p'. Thus the shaded area due to these rows will be smaller in u' than in u. To compensate for this decrease in area, we have to increase the width of the rows in u that are narrower than p'. Choose one such row at a time and make it wider (no more than p') in the new FU-pattern u' until the increase in area is sufficient to keep the total area the same. The remaining rows have the same width in u and u'. For example, the FU-pattern in Figure 2 (b) can be constructed from Figure 2 (a) using the procedure described above.

It can be easily seen, that while constructing u' from u, we could have only increased the area in columns 1 to $\lfloor p' \rfloor$, and could have only decreased the area in columns higher than $\lfloor p' \rfloor$, while the amount of increase and the amount of decrease were the same to keep the total area unchanged. Since the cost of the columns increases with the index, the cost of the new FU-pattern u' can be no greater than the cost of u, which is $f_{min}(p)$. Therefore, we can conclude that $f_{min}(p') \leq f_{min}(p)$.

Property (7): Consider an FU-pattern u' of cost $f_{max}(p')$ and then construct another FU-pattern u with $m(u) = p$, $p \geq p'$ in the following way. Increase the widest row of u' to a width of p and then decrease the widths of the remaining rows until the area becomes the same. Using an argument similar as in the previous paragraph, we can conclude the cost of the new FU-pattern u will be no less than the cost $f_{max}(p')$ of u'. Thus $f_{max}(p) \geq f_{max}(p')$.

Property (8): First note that the values of c_j as given in (2) satisfy the following recursion:

$$c_j = \begin{cases} 0 & j = 1, \ldots, l-1 \\ 1 + |S| \sum_{i=1}^{j-1} c_i & j \geq l \end{cases} \qquad (9)$$

$$\text{where } l = \left\lceil \frac{|I_k|}{|S|} \right\rceil$$

Let u be an FU-pattern of cost $f_{min}(p)$. Therefore, $m(u) = p$, and at least one row of u has width $u_s = p$. This implies that the area of the shaded region in column p is at least 1, and the cost of u is at least c_p. Formally:

$$f_{min}(p) \geq c_p \qquad (10)$$

It is not hard to see that $f_{max}(p-1)$ can not be greater than the cost of the FU-pattern whose shaded region covers the entire s, j space through column $p-1$. This

can be mathematically written as follows:

$$\begin{aligned} f_{max}(p-1) &\leq |S| \sum_{i=1}^{p-1} c_i \\ &= c_{p-1} && \text{from (9)} \\ &< c_p && \text{from (2)} \\ &\leq f_{min}(p) && \text{from (10)} \end{aligned}$$

□

So far, we have presented some properties of the FU-patterns, which we will use in the next section to prove a property satisfied by the optimal solutions of the extended problem (E_k).

5.2 Property of the Solution of (E_k)

In this section, we present a property satisfied by the optimal solutions of (E_k). The main result is presented in Proposition 2, which leads to a simple method of computing $\lceil \underline{m}_k^* \rceil$.

Consider the problems (AP_k) and (E_k). We will use $[x, m_k]$ and $[x, y]$ to denote any feasible solution of (AP_k) and (E_k) respectively. For any x corresponding to a feasible solution of (AP_k) or (E_k), we can compute an FU-pattern $u(x) \in U$ defined as follows:

$$u_s(x) = \sum_{i \in I_k} x_{is} \quad \forall\, s \in S \tag{11}$$

For the sake of notational simplicity, let us define:

$$y(x) = y(u(x)) \tag{12}$$

$$m(x) = m(u(x)) \tag{13}$$

Lemma 3 *For any x, if either $[x, m_k]$ is feasible in (AP_k) or $[x, y]$ is feasible in (E_k), then $[x, y(x)]$ is feasible in (E_k) and $f(y(x)) \leq f(y)$.*

Proof: From the constraints of (E_k) it can be easily verified that $[x, y(x)]$ is a feasible solution. We will only show that the cost $f(y(x))$ of $[x, y(x)]$ is no greater than the cost $f(y)$ of $[x, y]$.

Consider a particular $s \in S$. Let $d_j = y_{s,j}(x) - y_{s,j}$, and $a = \lfloor u_s \rfloor + 1$. The values of $y_{s,j}(x)$ can be found using (12), (11), and (3), and it can be seen that $d_j \geq 0$, for $j = 1, \ldots a-1$, and $d_j \leq 0$, for $j = a+1, \ldots \bar{m}_k$. Furthermore, according to (2) $c_j \leq c_{j+1}$, $j = 1, \ldots, \bar{m}_k - 1$. In the following we use these facts to show that, for each row $s \in S$, the difference in cost between $[x, y]$ and $[x, y(x)]$ is no greater than zero:

$$\begin{aligned} \sum_{j=1}^{\bar{m}_k} c_j d_j &= \sum_{j=1}^{a-1} c_j d_j + c_a d_a + \sum_{j=a+1}^{\bar{m}_k} c_j d_j \\ &\leq c_a \sum_{j=1}^{a-1} d_j + c_a d_a + c_a \sum_{j=a+1}^{\bar{m}_k} d_j \\ &= c_a \sum_{j=1}^{\bar{m}_k} d_j \\ &= 0 \end{aligned}$$

□

Proposition 2 *Let $[x^*, \underline{m}_k^*]$ and $[\tilde{x}, \tilde{y}]$ be the optimum solutions of (AP_k) and (E_k) respectively. Then:*

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

Proof: From Lemma 3, it can be concluded that $[x^*, y(x^*)]$ is a feasible solution of (E_k) and $[\tilde{x}, y(\tilde{x})]$ is an optimal solution of (E_k). Therefore:

$$f(y(x^*)) \geq f(y(\tilde{x})) \tag{14}$$

We can compute $m(x^*)$ from (13) and (11), and it can be easily seen that $\underline{m}_k^* = m(x^*)$. Since $\underline{m}_k^*$ represents the minimum number of FU's, we must have:

$$m(\tilde{x}) \geq m(x^*) \tag{15}$$

$$\begin{aligned} f_{max}(\lceil m(x^*) \rceil) &\geq f(y(x^*)) && \text{from (5)} \\ &\geq f(y(\tilde{x})) && \text{from (14)} \\ &\geq f_{min}(m(\tilde{x})) && \text{from (5)} \\ &\geq f_{min}(\lfloor m(\tilde{x}) \rfloor) && \text{from (6)} \\ &> f_{max}(\lfloor m(\tilde{x}) \rfloor - 1) && \text{from (8)} \end{aligned}$$

It can be easily verified that $m(\tilde{x}) \geq \lceil \frac{|I_k|}{|S|} \rceil$, which justifies our use of (6) and (8) in the above derivation. From the above relation we can write:

$$\begin{aligned} \lceil m(x^*) \rceil &> \lfloor m(\tilde{x}) \rfloor - 1 \\ &\geq \lfloor m(\tilde{x}) \rfloor \end{aligned}$$

If we combine the above relation with (15) and note that $\underline{m}_k^* = m(x^*)$, then we can conclude:

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

□

The result presented in the above proposition will be used in the following section to compute $\lceil \underline{\mathbf{m}}_k^* \rceil$ in polynomial time.

6 Computing the Lower Bound $\lceil \underline{\mathbf{m}}_k^* \rceil$ using the Extended Problem (E_k)

Section 4 has proven that the extended problem (E_k) can be solved as a linear program (LP). Thus we can compute $[\tilde{x}, \tilde{y}]$ by solving (E_k) with an LP-solver. If $m(\tilde{x})$ is integral, then it can be easily deduced from Proposition 2 that $\lceil \underline{m}_k^* \rceil = m(\tilde{x})$, giving us our desired lower bound. If $m(\tilde{x})$ is fractional, then there are at most two choices for $\lceil \underline{m}_k^* \rceil$, namely $\lceil \underline{m}_k^* \rceil$ or $\lfloor m(\tilde{x}) \rfloor$. In this case, the result can be found in the following manner.

As explained in the proof of Proposition 2, $[x^*, y(x^*)]$ is a feasible solution of (E_k), and $u_s(x^*) \leq m^*$, $\forall\, s \in S$. If $\lceil m^* \rceil = \lfloor m(\tilde{x}) \rfloor$, then it can be concluded from (12) that $y_{s,j}(x^*) = 0$, for $j = \lfloor m(\tilde{x}) \rfloor + 1, \ldots, \bar{m}_k$, $\forall\, s \in S$. To see whether such a feasible solution exists, set $y_{s,j} = 0$, for $j = \lfloor m(\tilde{x}) \rfloor + 1, \ldots, \bar{m}_k$, $\forall\, s \in S$ and solve (E_k) one more time. If there exists a feasible solution, then $\lceil \underline{m}_k^* \rceil = \lfloor m(\tilde{x}) \rfloor$; otherwise $\lceil \underline{m}_k^* \rceil = \lceil m(\tilde{x}) \rceil$.

The above method demonstrates that $\lceil \underline{m}_k^* \rceil$, the lower bound on the number of FU's of type k, can be computed by solving the linear program (E_k) at most 2 times.

	Schedule Length	Loop Length	EXACT (m_k^*)	LBND ($\lceil \underline{m}_k^* \rceil$)	SHARMA [7]
(±, ∗)	17	17	(3,3)	(3,3)	(3,3)
(±, ⊛)	17	17	(3,2)	(3,2)	(3,2)
(±, ∗)	18	18	(2,2)	(2,2)	(2,2)
(±, ⊛)	18	18	(3,1)	(3,1)	(2,1)
(±, ∗)	18	16	(3,2)	(3,2)	(2,2)
(±, ⊛)	18	16	(3,1)	(3,1)	(2,1)
(±, ∗)	19	19	(2,2)	(2,2)	(2,2)
(±, ⊛)	19	19	(2,1)	(2,1)	(2,1)
(±, ∗)	19	17	(2,2)	(2,2)	(2,2)
(±, ⊛)	19	17	(2,1)	(2,1)	(2,1)
(±, ∗)	21	21	(2,1)	(2,1)	(2,1)
(±, ⊛)	21	21	(2,1)	(2,1)	(2,1)
(±, ∗)	21	19	(2,1)	(2,1)	(2,1)
(±, ⊛)	21	19	(2,1)	(2,1)	(2,1)

Table 1: Number of FU's for the Elliptic Wave Filter

7 Experiments and Results

We compared the accuracy of the bounds obtained by our method against the actual schedules produced by RPI-ILP [2], an integer programming based scheduling algorithm. The experiments were conducted in two steps. First, for a given schedule length, we used our method as reported in Section 6 to compute the lower bounds on the number of FU's. In the next step, we constrained the number of FU's to the bounds produced in the first step, and then ran RPI-ILP to try to find a feasible schedule with those bounds.

If RPI-ILP could find a feasible schedule, then we could conclude that the bounds produced by our method are as tight as possible. If no feasible schedule was found (which means that one doesn't exist), then we increased the number of FU's until RPI-ILP was able to find a feasible schedule. In this case, the bound produced by our method was not as tight as possible, and the amount by which the number of FU's had to be increased gives a measure looseness of the bounds produced.

Experiments were run on the elliptical wave filter (ewf), and the discrete cosine transform (dct). The results are listed in Table 1 and 2, respectively. The bounds computed by our method are reported in the column labeled LBND, and the number of FU's for which a feasible schedule was found by RPI-ILP are reported in the column labeled EXACT. Our results are compared to the lower bound estimates produced by applying the technique of [7], which are reported under the heading SHARMA [7]. In these examples, FU ± (adder/subtractor) was assumed to take one control step, FU ∗ (multiplier) was assumed to take two control steps, and FU ⊛ (pipelined multiplier) was assumed to have a latency 1.

For ewf, the bounds on the number of FU's produced by our method (LBND) were as tight as possible in all cases (i.e. RPI-ILP could find feasible schedules that satisfy the bounds with equality). For dct, the bounds were as tight as possible in all cases except the one in row 2 of Table 2, where our bound is off by one adder/subtractor from the exact design. It can also be seen from the tables that our bounds are consistently tighter than those generated by the algorithm reported in [7].

	Schedule Length	EXACT (m_k^*)	LBND ($\lceil \underline{m}_k^* \rceil$)	SHARMA [7]
(±, ⊛)	7	(6,5)	(6,5)	(6,4)
(±, ⊛)	7	(8,4)	(7,4)	
(±, ⊛)	8	(5,4)	(5,4)	(4,4)
(±, ⊛)	9	(4,3)	(4,3)	(4,3)

Table 2: Number of FU's for the Discrete Cosine Transform

To demonstrate how our algorithm considers interdependencies between the bounds, consider Table 2. For a schedule length of 7, there exists two design points (6,5) and (8,4) as was verified by running RPI-ILP. In this case, our method produced two sets of bounds (6,5) and (7,4), one for each design point, whereas previous techniques like the algorithm in [7] considers the bound on each type of FU as independent of the other and produced only one set of bounds (6,4).

8 Summary and Future Work

In this paper we have presented a formal description of the lower-bounding problem that produces tighter bounds on FU's than existing methods, and considers the interdependencies of the bounds on different FU-types. Although the 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.

Our formulation assumes that each FU can accept one operator at every control step. When the FU's have a latency greater than 1, each multicycle operator is considered as a combination of many unicycle operators. However, it can not be ensured that the unicycle components belonging to the same multicycle operator are allocated in consecutive control steps of the same FU. This might affect the tightness of the bound produced. A polynomial time solution of a more general formulation is yet to be found.

References

[1] Samit Chaudhuri and Robert A. Walker. Computing Lower Bounds on Functional Units before Scheduling. Technical Report 94–6, CS Dept, Rensselaer Polytechnic Institute, March 1994.

[2] Samit Chaudhuri, Robert A. Walker, and John Mitchell. The Structure of Assignment, Precedence and Resource Constraints in the ILP Approach to the Scheduling Problem. In *IEEE International Conference on Computer Design*, pages 25–29, 1993.

[3] Catherine H. Gebotys and Mohamed I. Elmasry. *Optimal VLSI Architectural Synthesis*. Kluwer Academic Publishers, 1991.

[4] Cheng-Tsung Hwang, Jiahn-Hurng Lee, and Yu-Chin Hsu. A Formal Approach to the Scheduling Problem in High Level Synthesis. *IEEE Trans. on Computer-Aided Design*, 10(4):464–475, April 1991.

[5] Rajiv Jain, Alice Parker, and Nohbyung Park. Predicting System-Level Area and Delay for Pipelined and Nonpipelined Designs. *IEEE Trans. on Computer-Aided Design*, 11(8):955–965, Aug 1992.

[6] George L. Nemhauser and Laurence A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, 1988.

[7] Alok Sharma and Rajiv Jain. Estimating Architectural Resources and Performance for High-Level Synthesis Applications. *IEEE Trans. on VLSI Systems*, 1(2):175–190, Jun 1993.

[8] J. D. Ullman. NP-Complete Scheduling Problems. *J. Comput. System Sci*, 10(10):384–393, 1975.