# An Exact Methodology for Scheduling in a 3D Design Space*

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

## *Abstract*

*This paper describes an exact solution methodology, implemented in Rensselaer's Voyager design space exploration system, for solving the scheduling problem in a 3-dimensional (3D) design space: the usual 2D design space (which trades off area and schedule length), plus a third dimension representing clock length. Unlike design space exploration methodologies which rely on bounds or estimates, this methodology is guaranteed to find the globally optimal solution to the 3D scheduling problem. Furthermore, this methodology efficiently prunes the search space, eliminating provably inferior design points through: (1) a careful selection of candidate clock lengths, and (2) tight bounds on the number of functional units of each type or on the schedule length.*

## *1 Introduction*

In high-level synthesis, 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). In reality, however, this 2D design space is only a small part of a much larger design space. One such larger design space is presented by De Micheli in [15], and is illustrated in Figure 1. Here the design space for high-level synthesis is viewed as a 3-dimensional (3D) space, with axes not only representing schedule length and area, but clock (cycle) length as well.

A typical scheduling algorithm explores only one 2D slice of this larger 3D design space – the 2D slice corresponding to a fixed clock length chosen *a priori* by the designer. This clock length depends on many factors, including the delays of the functional units, storage elements, glue logic, and wiring, as well as clock skew. Some of those values are unknown before scheduling, and can therefore only be estimated at this stage in the design process.

Given this lack of detailed information, the designer is forced to make an *ad hoc* and frequently arbitrary guess at the clock length, unfortunately eliminating an entire dimension of the search space. Thus even an optimal scheduler will explore only that one 2D slice of the design space, and will produce a schedule that is optimal only for that one clock length. A better schedule may exist for a different clock length, but that better schedule will not be found.

To motivate the need to explore this larger design space, consider the problem of scheduling the well-known Elliptic Wave Filter [23, p.206] (EWF) benchmark, under a variety of resource constraints, to find the fastest possible schedule. Assume that the VDP100 module library [16, 24] is used, which has a multiplication delay of 163ns, and an addition delay of 48ns.

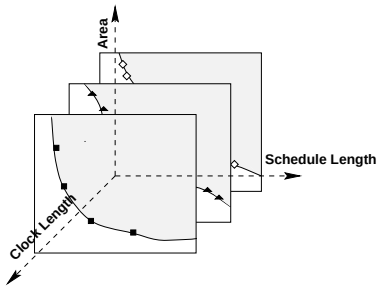


**Figure 1**: The Larger 3-Dimensional (3D) Design Space

| | 3 Mult, 3 Add | | 2 Mult, 2 Add | | 1 Mult, 2 Add | |
|---|---|---|---|---|---|---|
| Clock | Csteps | ns | Csteps | ns | Csteps | ns |
| 163 | 14 | 2282 | 16 | 2608 | 16 | 2608 |
| 82 | 17 | 1394 | 18 | 1476 | 21 | 1722 |
| 55 | 21 | 1155 | 22 | 1210 | 29 | 1595 |
| 48 | 25 | 1200 | 26 | 1248 | 37 | 1776 |
| 24 | 46 | 1104 | 48 | 1152 | 66 | 1584 |

**Table 1**: Resource-Constrained Scheduling Results for the EWF

Forced to choose a clock length for the scheduling algorithm to use, the designer would probably choose either a clock length of 48ns or 163ns – the execution delay of either addition or multiplication. Given those clock lengths, an optimal scheduler that supports multi-cycle operations (such as the ILP-based scheduler [5] in our Voyager design space exploration system) would produce the results shown on the lines labeled "48" and "163" in Table 1.

Now consider the other lines of Table 1, which represent other, perhaps less obvious, choices for the clock length. For each resource constraint, the fastest design corresponds to a clock length of 24ns – a design that would not be found by a scheduling methodology limited by *ad hoc* guesses. [1] Thus it is important to explore a number of candidate clock lengths to find the globally optimal solution.

This paper presents an exact solution methodology, implemented in the Voyager design space exploration system, to find the globally optimal solution to the scheduling problem in this 3D design space. This methodology makes the problem tractable through: (1) careful pruning of provably inferior points from the design space, and (2) provably efficient exact algorithms for solving the individual problems.

* 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.

[1] Of course, this small clock length also results in a larger number of control steps, and thus a larger and more complex control unit. However, note that a clock length of 55ns – more comparable to the *ad hoc* guesses – results in a schedule almost as fast as the one corresponding to a 24ns clock, and faster than those corresponding to the *ad hoc* guesses.

**Time-Constrained 3D Scheduling (TCS-3D):**
read in DFG, module library, and time constraint
compute minimal set of candidate clock lengths
**for** each clock length
    perform **Time-Constrained Scheduling (TCS)**
**end**
present the results to the user for evaluation

**Time-Constrained Scheduling (TCS):**
compute tight lower bounds on the number of functional units of each type
use these lower bounds as resource constraints, and solve **TRCS** as a decision problem
**while** no feasible schedule is found
    increase the resource constraints
    solve **TRCS** as a decision problem
**end**

**Figure 2**: Voyager's Time-Constrained 3D Scheduling (TCS-3D) Methodology

**Resource-Constrained 3D Scheduling (RCS-3D):**
read in DFG, module library, and resource constraints
compute minimal set of candidate clock lengths
**for** each clock length
    perform **Resource-Constrained Scheduling (RCS)**
**end**
present the results to the user for evaluation

**Resource-Constrained Scheduling (RCS):**
compute a tight lower bound on the schedule length
use this lower bound as a time constraint, and solve **TRCS** as a decision problem
**while** no feasible schedule is found
    increase the time constraint
    solve **TRCS** as a decision problem
**end**

**Figure 3**: Voyager's Resource-Constrained 3D Scheduling (RCS-3D) Methodology

## *2 Methodology Overview*

This paper presents two methodologies to solve the clock determination and scheduling problem, that are guaranteed to find the globally optimal design, and that are far more efficient than an exhaustive search of the design space. One methodology solves the Time-Constrained 3D Scheduling (TCS-3D) problem (Figure 2), while the other solves the Resource-Constrained 3D Scheduling (RCS-3D) problem (Figure 3). Both methodologies are implemented in Rensselaer's Voyager design space exploration system.

The core of each methodology is roughly the same – each methodology computes a set of candidate clock lengths, and then, for each candidate clock length, optimally solves the scheduling problem. However, a straightforward implementation of this core methodology takes much too long to solve, even for small benchmarks. Thus it is important to (1) solve the scheduling problem for only a small, *provably minimal* set of candidate clock lengths, and (2) solve the scheduling problems as *efficiently* as possible so that an optimal solution is found in a reasonable amount of time.

Since the search spaces for the TCS and RCS problems are each larger than that of the TRCS problem, these methodologies solve the TCS and RCS problems by generating the missing constraints, in effect converting each into an easier-to-solve TRCS problem. For the TCS problem, the methodology computes constraints on the number of functional units of each type; for the RCS problem, it computes a time constraint on the length of the schedule. Since these constraints can also be found efficiently, the entire methodology is efficient.

### *2.1 Time-Constrained 3D Scheduling (TCS-3D)*

Voyager's methodology for solving the time-constrained 3D scheduling problem is outlined in Figure 2. This methodology begins by reading in the data flow graph (DFG), the execution delays for the relevant functional units in the module library, and the overall time constraint.

The minimal set of candidate clock lengths is then determined (see Section 3), based on the execution delays of the relevant functional units in the module library. For the EWF and the module library described earlier, 10 candidate clock lengths would be generated. For each of these clock lengths, time-constrained scheduling is then performed, and the results are presented to the user for evaluation.

To solve the TCS problem efficiently, Voyager's ILP formulation of the TRCS problem (described in Section 4) is used as follows. First, tight lower bounds on the number of functional units of each type are computed (using a method sketched out in Section 5). These bounds are then used as resource constraints, and the TRCS problem is solved as a decision problem. If TRCS produces a feasible schedule, then that schedule is guaranteed to be optimal; if not, the resource constraints are increased, and this process is repeated.

This TCS-3D solution methodology is relatively efficient for the following reasons. First, the functional unit lower bounds can be computed in polynomial time, by solving at most two Linear Programs (LPs). Second, TRCS is solved as a decision problem, rather than an optimization problem, using a formulation that is well-structured, and requires few, if any, branches in a branch-and-bound search [5]. Finally, the functional unit lower bounds are highly accurate [4] (in almost every case they lead immediately to a feasible solution), so in practice the lower bounds seldom have to be increased to solve TRCS again. Thus the TCS-3D problem can be solved quickly, even for medium-sized benchmarks (see Section 7).

The efficiency of the methodology can be further increased if the goal is to find the schedule with the fewest number of functional units. In this case, before each TCS problem is solved as a TRCS problem, the FU lower bounds are compared to the number of FUs required in the best previous schedule. If the new bounds are smaller, then the TRCS problem is solved as explained above; if the new bounds are larger, then there is no need to solve the TRCS problem since it would require more functional units than the best solution found so far.

### *2.2 Resource-Constrained 3D Scheduling (RCS-3D)*

Voyager's methodology for solving the resource-constrained 3D scheduling problem is similar (see Figure 3). This methodology reads in a resource constraint, and generates a minimal set of candidate clocks using the clock length determination algorithm described in Section 3. For each of these clock lengths, resource-constrained scheduling is then performed.

To solve the RCS problem efficiently, Voyager's ILP formulation of the TRCS problem (Section 4) is used as follows. First, a tight lower bound on the overall length of the schedule is computed (Section 6). This bound is then used as a time constraint, and the TRCS problem is solved as a decision problem. If TRCS produces a feasible schedule, then that schedule is guaranteed to be optimal; if not, the time constraint is increased, and this process is repeated. The RCS problem can be solved quickly, even for medium-sized benchmarks (see Section 7).

The efficiency of the methodology can be further increased if the goal is to find the shortest schedule. In this case, before each RCS problem is solved as a TRCS problem, the schedule length lower bound is compared to the length of best previous schedule. If the new bound is smaller, then the TRCS problem is solved as explained above; if the new bound is larger, then there is no need to solve the TRCS problem since it would result in a longer schedule than the best solution found so far.

### *2.3 Advantages of this Solution Methodology*

In summary, Voyager's exact solution methodology has a two-fold advantage over previous methodologies: (1) guaranteed optimal results, and (2) solution techniques based on efficient pruning of the search space.

Unlike other design space exploration methodologies which rely on bounds or estimates to make the problem tractable, this methodology generates the minimal set of candidate clock lengths that could possibly correspond to the optimal design, and then optimally solves either the TCS or RCS problem for each of those clock lengths. Thus it is *guaranteed* to find the globally optimal result.

Furthermore, although this methodology may appear at first glance to perform exhaustive scheduling, in reality it is quite *efficient* for three reasons. First, a minimal set of candidate clock lengths is generated, and scheduling is performed for only those few values. Second, instead of directly solving the TCS or RCS problem, the missing constraints are generated, converting that problem into a TRCS problem with a smaller search space; moreover, those constraints are tight, and are also generated efficiently. Finally, a TRCS formulation is used that was carefully designed and is well-structured [5], and therefore usually finds an optimal solution with few branches.

## *3 Determining Candidate Clock Lengths*

One of the most important parameters needed by any scheduling algorithm is the length of the system clock. However, determining this clock length requires a detailed analysis of the clock skew, wire delays, glue logic delays, setup and propagation delays of the storage elements, etc. [1] – quantities largely unknown during high-level synthesis. Fortunately, although such a detailed analysis is necessary later in the design process, it is not needed during high-level synthesis, where only the macroscopic structure of the circuit is determined.

One appropriate model of the clock length during high-level synthesis is presented by Chaiyakul and Gajski in [3]. Here the clock length is assumed to have 3 components: datapath delay, control delay, and wire delay. Of these three, the control delays and the wire delays cannot be meaningfully estimated before scheduling, but fortunately those delays do not play a major role in the scheduling problem (because resource utilization in the datapath is primarily affected by the delays of the datapath units). Thus for scheduling, we use only the datapath delays to determine the clock length, and we ignore the control and wire delays, realizing that the actual clock length (determined later) will be longer due to those delays. Furthermore, these operation execution delays are computed assuming a bus-based architecture with a point-to-point interconnection topology, meaning there exists only one bus between any two functional unit and/or storage unit ports.

**Definition 1** *Let $t_s$ and $t_p$ be the setup time and propagation delay of the registers, and let $t_\triangleright$ be the delay of a tri-state driver. If the delay of a functional unit of type $k$ is denoted as $delay\,(k)$, the execution delay $d_k$ for a register-to-register transfer executing an operation of type $k$ is given as:*

$$d_k = t_s + t_p + 2t_\triangleright + delay\,(k)$$

Throughout the remainder of this section, the set $D$ will be used to denote the set of all $d_k$'s found in the given DFG.

Before discussing Voyager's methodology for determining candidate clock lengths, it is necessary to have a measure of the quality of one clock length with respect to other clock lengths for a particular operation. One such measure that is commonly used is operation *slack*, defined as follows:

**Definition 2** *For a given clock length $c$, the slack $s_k$ of an operation of type $k$ is given by:*

$$s_k(c) = c \cdot \lceil d_k/c \rceil - d_k$$

Voyager's methodology determines a minimal set of candidate clock lengths in a range $[\underline{c}, \bar{c}]$. This range is bounded by $\underline{c}$, the minimum clock length possible for implementing the design's controller. One of the goals of the Voyager's 3D design space exploration methodology is to find the minimal set of non-inferior clock lengths $c^*$ in this range that need to be examined in order to find the globally optimal solution.

Unfortunately, the clock determination problem is usually ignored in favor of *ad hoc* decisions or estimates, which, as demonstrated later, can ignore much of the design space and lead to an inferior design. For example, several previous clock estimation schemes [19, 11] use the delay of the slowest functional unit as the estimated clock length. A more realistic approach is used in [16], in which a contiguous range of integer candidate clock lengths is heuristically evaluated in an attempt to provide some guidance as to the "best" clock length to choose.

However, all of these approaches choose the clock length before, and independent of, scheduling. Thus they are at best *estimates*, since it is never possible to guarantee that a better schedule with a different clock length does not exist. Therefore it may seem at first that the globally optimal solution to the 3D scheduling problem cannot be found without optimally solving the scheduling problem for every possible clock length – a prohibitively expensive exhaustive search.

Fortunately, this exhaustive search is not necessary. In [7], Corazao *et al.* combined clock length determination with the problem of operation template matching, and made

some suggestions to reduce the number of candidate clock lengths. However, the number of candidate clock lengths can be reduced even further, as shown in our Theorem 1 below (a similar observation was made by Chen *et al.* in [6], but presented without proof).

The following theorem shows that only certain clock lengths in the range $[\underline{c}, \bar{c}]$ must be explored to find the globally optimal clock length $c^*$, when chaining is not considered, and when clock lengths are *not* assumed to be integers:

**Theorem 1** *$c^*$ integrally divides at least one of the register transfer delays. More formally, $s_k(c^*) = 0$ for at least one $k \in K$.*

*Proof:* Consider any clock period $c$ such that $s_k(c) > 0 \; \forall \; k$, and an optimal basic schedule generated using $c$ as the clock length. We will show that $c$ is not optimal because there can be found another clock period $c'$, that leads to a faster schedule with the same number of functional units and csteps as the original schedule.

Let $\epsilon = \min_{k \in K}\{s_k(c)/\lceil d_k/c \rceil\}$. The new clock period $c'$ can then be found as $c' = c - \epsilon$. Using the definition of $s_k(c)$ (Definition 2), the value of $c'$ can be derived as $\max_{k \in K}\{d_k/\lceil d_k/c \rceil\}$, which can be substituted for $c'$ in $\lceil d_k/c' \rceil$ giving:

$$\lceil d_k/c' \rceil = min_{k \in K}\{\lceil d_k/c \rceil\} \leq \lceil d_k/c \rceil$$

Furthermore, since $c' \leq c$, it also follows that $\lceil d_k/c' \rceil \geq \lceil d_k/c \rceil$. These two relations imply $\lceil d_k/c' \rceil = \lceil d_k/c \rceil$, i.e., each register transfer takes the same number of control steps with the new clock $c'$ as with the original clock $c$.

Hence the original schedule will still be valid with the new clock $c'$. However, $c'$ being less than $c$, will result in a faster execution time, while the number of csteps and functional units remain the same. □

**Corollary 1** *When using integer clock lengths, any non-integer clock $c$ generated through application of theorem 1 can be replaced by $c' = \lceil c \rceil$. More formally, $\lfloor s_k(c')/\lceil d_k/c' \rceil \rfloor = 0$ for at least one $k \in K$.*

In summary, Theorem 1 and Corollary 1 give a method for determining a small set of candidate clock lengths $CK$, that provably contains the optimum clock length $c^*$. This set is computed as $CK = div(D)$, where $div(D)$ denotes the ceilings of all integral divisors of the delays $d_k$ that fall in the range $[\underline{c}, \bar{c}]$. In practice, the size of $CK$ is less than 10% of that of the integer range $[\underline{c}, \bar{c}]$.

Similar theorems can be developed for a scheduler that supports chaining, or a scheduler that supports operation templates. However, those theorems are not presented here due to lack of space.

## 4 Optimally Solving the Scheduling Problem

In high-level synthesis, the basic scheduling problem is the problem of determining the control step in which each operation will execute. After a careful formal analysis of the scheduling problem [5], we were able to develop well-structured Integer Linear Programming (ILP) formulations of the scheduling problems, in particular the TRCS problem.

Voyager's formulation of the TRCS problem can be summarized as follows. If $\{o_i \mid i \in I\}$ denotes the set of all operations, and $S_i$ denotes the schedule interval $[asap_i, alap_i]$ for operation $o_i$, then binary variables $x_{i,s}$, $s \in S_i$ can be used to indicate whether or not operation $o_i$ is scheduled in cstep $s$. In any feasible schedule, the values of these variables must satisfy three types of constraints: (1) assignment constraints (A), which ensure that each operation is scheduled onto exactly one cstep; (2) precedence constraints (P), which ensure that each operation is always scheduled after all of its predecessors; and (3) resource constraints (R), which ensure that the schedule does not use more than the available number of functional units of each type.

The TRCS problem is the problem of determining whether or not a feasible schedule exists that satisfies these constraints, and can be written succinctly as:

$$\min\{\, \mathbf{0}^T\mathbf{x} \mid \mathbf{M}_a\mathbf{x} = \mathbf{1} \;;\; \mathbf{M}_t\mathbf{x} \leq \mathbf{1} \;;\; \mathbf{M}_r\mathbf{x} \leq \mathbf{m} \;;\; \mathbf{x} \text{ integer} \,\}$$

where $\mathbf{0}$ is a vector of zeros, and $\mathbf{M}_a$, $\mathbf{M}_t$ and $\mathbf{M}_r$ are the coefficient matrices due to the assignment constraints, precedence constraints, and resource constraints, respectively. Further details on Voyager's scheduling ILP formulations can be found in [5].

## 5 Bounding the Number of Functional Units

As discussed earlier in Section 2, it is important to generate tight lower bounds on the number of functional units (FUs) of each type, so that those bounds can be used as resource constraints to convert the TCS problem into an easier-to-solve TRCS problem. Furthermore, those bounds must be computed efficiently, preferably with a polynomial-time algorithm.

This functional unit lower-bounding problem can be viewed as a relaxation of the functional unit minimization problem. Unfortunately, there are many possible relaxations of that problem, and therefore many possible functional unit lower-bounding problems. Ideally, we would like to find the tightest bounds of all possible relaxations, but with an efficient solution methodology.

One approach to forming the FU lower-bounding problem from the FU minimization problem is to relax the precedence constraints between operations. Other than our work [4], we are aware of only two methodologies to compute FU lower bounds in this manner – the relaxations considered by Jain [12] (and a similar relaxation in [13]), and the tighter relaxations in [22, 18, 10], and [20] based on a method originally proposed by Fernández and Bussell in [8, Theorem 1].

Our work, described more fully in [4], takes a different approach. We start with a formal description of the FU minimization problem, which finds the minimum value $m_k$ of the number of functional units of type $k \in K$. We then relax this problem to form a generic description of an entire class of FU lower-bounding problems (the problems above are special cases of this generic class). Finally, we select the one lower-bounding problem that produces the tightest possible bound, and solve that problem. Thus our approach formalizes an entire class of FU lower-bounding problems, and is guaranteed to produce the tightest possible bounds. In practice, we have verified that the bounds produced are exact in most cases.

Furthermore, the solution to this functional unit lower-bounding problem can be found by solving at most two LP's (which can be done in polynomial time). Thus we have developed a functional unit lower-bounding methodology [4]

that is guaranteed to produce the tightest bounds of all possible precedence relaxations and that does so in polynomial time, even though our original formulation was an ILP formulation.

## 6 Bounding the Length of the Schedule

The previous section briefly described Voyager's method to generate tight lower bounds on the number of functional units, so that the search space for TCS problem could be reduced to solve that problem more efficiently, as described in Section 2. This section presents a similar method to generate a tight lower bound on the schedule length, so that the RCS problem can be solved more efficiently.

One early formulation of the schedule length lower-bounding problem in presence of resource constraints is presented in [12]; however, the bounds produced by that approach are very loose. More recent algorithms that produce tighter bounds are those in [21] and [20], based on Jackson's earliest deadline rule (ED-Rule) [2], and those in [22] and [10], based on a theorem originally given by Fernández and Bussell in [8, Theorem 2]. Furthermore, those algorithms can be applied iteratively (Hu *et al.* apply Fernández' Theorem 2 in [9], and Langevin applies ED-Rule in [14]), producing even tighter bounds, although at the cost of increased algorithmic complexity.

Our investigations into this problem have shown that we can relax our ILP formulation of the RCS problem to form a generic description of an entire class of schedule length lower-bounding problems, in much the same manner as we did for FU lower-bounding. As with functional unit lower-bounding, we select the one lower-bounding problem that produces the tightest possible bound (the problems above are special cases of this generic class), and solve that problem. Thus our approach formalizes an entire class of schedule length lower-bounding problems, and is guaranteed to produce the tightest bound of all possible precedence relaxations in polynomial time.

## 7 Experimental Results

To demonstrate the accuracy and performance of Voyager's 3D scheduling methodology, we conducted a series of experiments using the well-known Elliptic Wave Filter (EWF) [23, p.206] and Discrete Cosine Transform [17] (DCT) benchmarks. We used the VDP100 module library from [16, 24], giving a datapath delay of 48ns for addition, 56ns for subtraction, and 163ns for multiplication. For each benchmark, we performed Time-Constrained 3D Scheduling (TCS-3D) and Resource-Constrained 3D Scheduling (RCS-3D) using the methodologies presented in Section 2.

### 7.1 Elliptic Wave Filter (EWF)

The TCS-3D results for the EWF are presented in Table 2. They show, for each of two time constraints, those clock lengths from the candidate set that lead to a feasible schedule (the other clock lengths lead to infeasible schedules regardless of the number of functional units available).

For a time constraint of 1394ns, three clock lengths (55ns, 48ns, and 24ns) led to the minimum number of functional units. Of these, the schedule for the 55ns clock ($\lceil d_{mult}/3 \rceil$) requires the fewest control steps (and thus potentially a smaller controller), so would be preferable. Note that the 48ns clock – one of the "obvious" *ad hoc* guesses ($d_{add}$) – may require an additional multiplier.

| Clock | Time | | (Mult, Add) |
|---|---|---|---|
| | Csteps | ns | |
| ***Time Constraint = 1394ns*** | | | |
| 82 | 17 | 1394 | (3, 3) |
| 55 | 25 | 1375 | (2, 2) |
| 48 | 29 | 1392 | (2, 2) |
| 24 | 58 | 1392 | (2, 2) |
| ***Time Constraint = 1035ns (tightest)*** | | | |
| 24 | 43 | 1032 | (4, 3) |

**Table 2**: EWF – TCS-3D Results

| Clock | ASAP | | 1 *, 2 + | | 2 *, 2 + | | 3 *, 3 + | |
|---|---|---|---|---|---|---|---|---|
| | Cs | ns | Cs | ns | Cs | ns | Cs | ns |
| 163 | 14 | 2282 | 16 | 2608 | 16 | 2608 | 14 | 2282 |
| 82 | 17 | 1394 | 21 | 1722 | 18 | 1476 | 17 | 1394 |
| 55 | 20 | 1100 | 29 | **1595** | 22 | 1210 | 21 | **1155** |
| 48 | 23 | 1104 | LB | 1632 | 26 | 1248 | 25 | 1200 |
| 41 | 34 | 1394 | LB | 1599 | LB | 1230 | LB | 1189 |
| 33 | 37 | 1221 | LB | 1617 | LB | 1221 | LB | 1190 |
| 28 | 40 | 1120 | LB | 1596 | 44 | 1232 | 42 | 1176 |
| 24 | 43 | 1032 | 66 | **1584** | 48 | 1152 | 46 | **1104** |
| 21 | 57 | 1197 | LB | 1596 | LB | 1155 | LB | 1113 |
| 19 | 60 | 1140 | LB | 1596 | LB | 1159 | LB | 1121 |

**Table 3**: EWF – RCS-3D Results

To find the fastest possible design, the critical path length was used to derive the tightest possible time constraint of 1035ns. For this time constraint, only one clock length – 24ns – led to a feasible schedule, and thus to the guaranteed fastest design.

The RCS-3D results for the EWF are shown in Table 3. Some schedule lengths of interest are shown in boldface, and those that were lower-bounded by the RCS-3D methodology are shown in gray along with the lower-bounded schedule length. As described in Section 2.2, the TRCS problem was not solved for those clock lengths, since each would result in a schedule that was longer than the shortest schedule found for the previous clock lengths.

The 55ns and 24ns clock lengths correspond to the fastest schedules. Again, it is interesting to note that neither of these clock lengths is an obvious *ad hoc* guess (55 is $\lceil d_{mult}/3 \rceil$, and 24 is both $\lceil d_{mult}/7 \rceil$ and $\lceil d_{add}/2 \rceil$), which means that the fastest schedule might be missed using more conventional methodologies. Furthermore, although the clock length of 24ns would correspond to a larger number of control steps (and perhaps a larger controller), that small clock length does result in the overall fastest schedule, because the small clock granularity tends to reduce the operation slack.

### 7.2 Discrete Cosine Transform (DCT)

The TCS-3D results for the DCT are presented in Table 4. The first set of results are for a time constraint of 500ns, corresponding to a design will run at 2MHz. Eight clock lengths produced feasible schedules, but only one – 24ns – led to the minimum number of functional units. To find the fastest possible design, the critical path length was used to derive the tightest possible time constraint of 434ns, and only one clock length – 24ns – led to a feasible schedule and thus to the guaranteed fastest design.

The RCS-3D results for the DCT are presented in Table 5. The 56ns clock length ($d_{sub}$) corresponds to the fastest schedule.

| Clock | Time | | (Mult, Add, Sub) |
|---|---|---|---|
| | Csteps | ns | |
| ***Time Constraint = 500ns (2MHz)*** | | | |
| 56 | 8 | 448 | (11, 7, 4) |
| 55 | 9 | 495 | (15, 5, 6), (13, 6 4) |
| 48 | 10 | 480 | (16, 5, 6) |
| 33 | 15 | 495 | (11, 7, 4), (15, 5, 4) |
| 28 | 17 | 476 | (11, 7, 4), (15, 5, 4) |
| 24 | 20 | 480 | **(11, 4, 4)** |
| 21 | 23 | 483 | (11, 7, 4), (15, 5, 4) |
| 19 | 26 | 494 | (11, 5, 4), (15, 7, 4) |
| ***Time Constraint = 434ns (tightest)*** | | | |
| 24 | 18 | 432 | (16, 5, 6) |

**Table 4**: DCT – TCS-3D Results

| 5 Mult, 3 Add, 2 Sub | | |
|---|---|---|
| Clock | Csteps | ns |
| 163 | 9 | 1467 |
| 82 | 10 | 820 |
| 56 | 14 | **784** |
| 55 | LB | 825 |
| 48 | LB | 816 |
| 41 | LB | 820 |
| 33 | LB | 792 |
| 28 | 28 | **784** |
| 24 | LB | 792 |
| 21 | LB | 798 |
| 19 | LB | 798 |

**Table 5**: DCT – RCS-3D Results

### *7.3 Methodology Run Times*

Voyager's design space exploration methodologies consists of three main tasks: computing the minimal set of candidate clock lengths, computing tight bounds on the number of functional units or on the schedule length, and solving the TRCS problem. The minimal set of candidate clock lengths can be computed quickly, and the bounds can be computed by solving at most two linear programs in polynomial time, as discussed in Sections 5 and 6. Finally, the TRCS formulation used in Voyager was carefully constructed and is well-structured, meaning that it converges on the optimal solution faster than an arbitrary formulation.

To motivate the need for solving the TCS or RCS problem by first computing bounds and then solving the resulting TRCS problem, consider the result of solving the TCS problem directly for a time constraint of 1394ns and a 24ns clock on the EWF benchmark. Even with a well-structured formulation such as Voyager's, solving this problem directly took over an hour of CPU time (using LINDO on a Sun SPARCstation 2). In contrast, we spent only 1.51 sec to compute the lower bounds on the number of functional units, and only 7.75 sec to solve the TRCS problem – solving the same problem in two orders of magnitude less time!

On a larger benchmark – the DCT – for a time constraint of 500ns and a 24ns clock, we spent 8.28 sec to compute the lower bounds on the number of functional units, and 2.62 sec to solve the TRCS problem. Again, directly solving the TCS problem for this case took over an hour.

In general, the best designs for each example were generated within seconds. However, for very small clock lengths (e.g. 19ns), the ILP for the TRCS problem becomes quite large, and in some cases would have taken hours to find the exact solution. Fortunately, even in those cases the bounds were produced fairly quickly, and could often obviate the need to solve the TRCS problem for those clock lengths as described in Sections 2.1 and 2.2.

## *8 Summary*

This paper has defined a new problem – the 3D scheduling problem – and has presented an exact solution methodology to solve that problem without resorting to a time-consuming exhaustive search. This solution methodology is *exact* – it is *guaranteed* to find the optimal clock length and schedule. Furthermore, it is *efficient* – it *prunes inferior points* in the design space through a careful selection of candidate clock lengths (an important design parameter too often determined by guesswork or estimates), and through tight bounds on the number of functional units or the length of the schedule.

## *References*


[1] H. B. Bakoglu. *Circuits, Interconnections, and Packaging for VLSI*. Addison-Wesley VLSI systems series. Addison-Wesley, Reading, MA, USA, 1990.

[2] J. Blażewicz. Simple Algorithms for Multiprocessor Scheduling to Meet Deadlines. *Information Processing Letters*, 6(5):162 – 164, Oct. 1977.

[3] V. Chaiyakul, A. C.-H. Wu, and D. D. Gajski. Timing Models for High Level Synthesis. In [25], pages 60–65.

[4] S. Chaudhuri and R. A. Walker. Computing Lower Bounds on Functional Units before Scheduling. In *Proc. of the 7th International Symposium on High-Level Synthesis*, pages 36–41, Niagara-on-the-Lake, Canada, May 18-20 1994. IEEE Computer Society Press.

[5] 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*, 2(4):456–471, Dec. 1994.

[6] L.-G. Chen and L.-G. Jeng. Optimal Module Set and Clock Cycle Selection for DSP Synthesis. In *Proc. of 1991 IEEE International Symp. on Circuits and Systems.*, pages 2200–2203, Singapore, June 11-14 1991. IEEE Computer Society Press.

[7] M. Corazao, M. Khalaf, L. Guerra, M. Potkonjak, and J. M. Rabaey. Instruction Set Mapping for Performance Optimization . In *Proc. of the IEEE/ACM International Conference on Computer-Aided Design*, pages 518–521, Santa Clara, California, Nov. 7-11 1993. IEEE Computer Society Press.

[8] E. B. Fernández and B. Bussell. Bounds on the number of Processors and Time for Multiprocessor Optimal Schedule. *IEEE Transactions on Computers*, C-22(8):745–751, Aug. 1973.

[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.*, pages 295–298, London, England, May 30-June 2 1994. IEEE Computer Society Press.

[10] 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 [26], pages 21–24.

[11] R. Jain, A. C. Parker, and N. Park. Module Selection for Pipeline Synthesis. In *Proc. of the 25th ACM/IEEE Design Automation Conf.*, pages 542–547, Anaheim, California, June 12-15 1988. IEEE Computer Society Press.

[12] 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*, 11(8):955–965, Aug. 1992.

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

[14] M. Langevin and E. Cerny. A Recursive Technique for Computing Lower-Bound Performance of Schedules. In [26], pages 16–20.

[15] G. D. Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill series in electrical and computer engineering. McGraw-Hill, New York, NY, USA, 1994.

[16] S. Narayan and D. D. Gajski. System Clock Estimation based on Clock Slack Minimization. In [25], pages 66–71.

[17] J. A. Nestor and G. Krishnamoorthy. SALSA: A New Approach to Scheduling with Timing Constraints. *IEEE Transactions on Computer-Aided Design*, 12(8):1107–1122, Aug. 1993.

[18] 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*, pages 182–187, San Jose, California, Nov. 6-10 1994. IEEE Computer Society Press.

[19] N. Park and A. C. Parker. Synthesis of Optimal Clocking Schemes. In *Proc. of the 23rd ACM/IEEE Design Automation Conf.*, pages 454–460, Las Vegas, June 1986. IEEE Computer Society Press.

[20] J. M. Rabaey and M. Potkonjak. Estimating Implementation Bounds for Real Time DSP Application Specific Circuits. *IEEE Transactions on Computer-Aided Design*, 13(6):669–683, June 1994.

[21] M. Rim and R. Jain. Lower-Bound Performance Estimation for the High-Level Synthesis Scheduling Problem. *IEEE Transactions on Computer-Aided Design*, 13(4):451–458, Apr. 1994.

[22] A. Sharma and R. Jain. Estimating Architectural Resources and Performance for High-Level Synthesis Applications. *IEEE Transactions on VLSI Systems*, 1(2):175–190, June 1993.

[23] 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*. Kluwer Academic Publishers Group, 101 Philip Drive, Assinippi Park, Norwell, MA 02061, 1990.

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

[25] *Proc. of the European Design Automation Conference (EuroDAC)*, Hamburg, Germany, Feb. 1992. IEEE Computer Society Press.

[26] *Proc. of the IEEE International Conference on Computer Design*, Cambridge, Massachusetts, Oct. 3-6 1993. IEEE Computer Society Press.

[27] *Proc. of the European Design and Test Conference*, Paris, France, Feb. 8Mar. 3 1994. IEEE Computer Society Press.