A Solution Methodology for Exact Design Space Exploration in a Three-Dimensional Design Space[1]

Samit Chaudhuri[2]
Cadence Design Systems
San Jose, CA 95134

Stephen A. Blythe[3]
Rensselaer Polytechnic Institute
Troy, NY 12180

Robert A. Walker[4]
Kent State University
Kent, OH 44242

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 a 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 or on the schedule length. Both chaining and multi-cycle operations are supported.***

I. Introduction

High-level synthesis is the design task of converting a behavioral description of a digital system into a register-transfer level design that implements that behavior. One of the central problems in high-level synthesis is the *scheduling* problem – the problem of mapping operations onto control steps (csteps) in the proper order. The scheduling problem is usually formulated in one of three ways, depending on the goal: (1) Time-Constrained Scheduling (TCS), which minimizes the number of resources when the number of control steps is fixed, (2) Resource-Constrained Scheduling (RCS), which minimizes the number of control steps when the number of functional units is fixed, or (3) Time- and Resource-Constrained Scheduling (TRCS), which determines whether or not a feasible schedule exists when both the number of functional units and the number of control steps are fixed.

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, and infeasible designs lie in the white region. Optimal designs lie on the curve between the two regions, and represent the tradeoff between time and area.

[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 a graduate student in the Department of Electrical, Computer, and Systems Engineering at Rensselaer.

[3]Department of Computer Science.

[4]Department of Mathematics and Computer Science. This work was performed when R. Walker was on the faculty of the Department of Computer Science at Rensselaer.

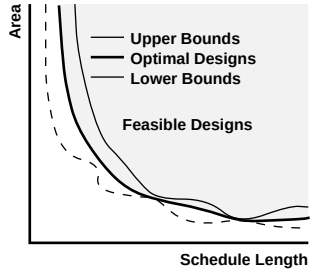

Fig. 1. The Usual 2-Dimensional (2D) Design Space

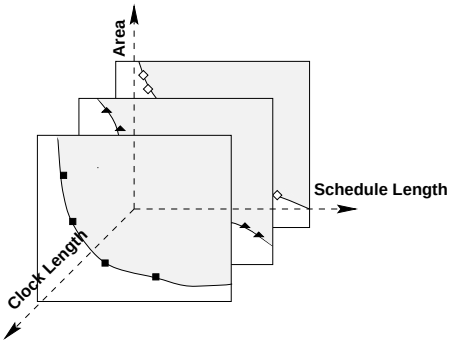

Fig. 2. The Larger 3-Dimensional (3D) Design Space

A. The 3D Design Space

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 [1], and is illustrated in Figure 2. 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 controller delays. Some of those values are unknown before scheduling, and can therefore only be estimated at this stage in the design process.

Unfortunately, the designer *must* specify a clock length (or at least, the data path component of the clock length) before scheduling. Lacking detailed information, the designer is forced to make an *ad hoc* and frequently arbitrary guess at the clock length[1]. Unfortunately, this *ad hoc*

[1]The designer may be relying on retiming [2], clock skew optimiza-

Clock	3 Mult, 3 Add		2 Mult, 2 Add		1 Mult, 2 Add	
	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 I
RESOURCE-CONSTRAINED SCHEDULING RESULTS FOR THE EWF

choice eliminates an entire dimension of the search space, so even an optimal scheduler will explore only the corresponding 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 will not be found.

To motivate the need to explore this 3D design space, consider the problem of scheduling the well-known Elliptic Wave Filter [6, p.206] (EWF) benchmark, under a variety of resource constraints, to find the fastest possible schedule. Assume that the VDP100 module library [7], [8] is used, which has a multiplication delay of 163ns, and an addition delay of 48ns.

Forced to select a clock length for the scheduling algorithm, 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 [9] in our Voyager design space exploration system) would produce the results shown on the rows labeled "48" and "163" in Table I.

Now consider the other rows of Table I, 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. [2] Thus it is important to explore a number of candidate clock lengths to find the globally optimal solution.

B. Exploring the 3D Design Space

A variety of methodologies can be used for design space exploration. The methodologies may use exact algorithms to find *optimal solutions*, may use heuristic algorithms to find *lower and upper bounds* on the optimal solution, or may use heuristic algorithms to *estimate* the optimal solution. In general, the tradeoff between these three types of methodologies is one of solution quality versus computation time. This paper is concerned with finding optimal (exact) solutions to the scheduling problem in the 3D design space.

Exhaustive Search:
read in DFG, module library, and any constraints
for each clock length
    optimally schedule the DFG
present the best result(s) to the user for evaluation

Fig. 3. Exhaustive Search of the 3D Design Space (Impractical)

One exact methodology for optimally solving this 3D-scheduling problem shown in Figure 3. This methodology exhaustively explores all potential clock lengths and all feasible schedules, and guarantees a globally optimal solution. Unfortunately, the computation time for this methodology is too high to be practical for all but the simplest examples.

In contrast, this paper presents a more efficient exact methodology, implemented in the Voyager design space exploration system, for optimally solving this 3D-scheduling problem. 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.

However, even this solution methodology is only the first step toward the larger design space exploration problem that eventually needs to be solved. As described here, our methodology does not consider the module selection or type mapping problems, and does not support loops or conditionals[3]. It also does not incorporate register, wiring, or controller area, and only partially incorporates the delays associated with the controller and wiring. Nevertheless, the work described here can serve as a foundation for an exact solution methodology that incorporates each of these factors, either by adding extra dimensions to the search space or by adding other stages to the methodology.

II. 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 4), while the other solves the Resource-Constrained 3D Scheduling (RCS-3D) problem (Figure 5). Both methodologies are implemented in Rensselaer's Voyager design space exploration system.

The core of each methodology is based roughly on the exhaustive search of Figure 3. 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

tion [3], or reclocking [4] to determine the final clock length. However, these techniques generally do not change the relative scheduling of the operations, and do not perform tradeoffs involving resource sharing, so they do not explore the high-level design space as fully as scheduling techniques. Nevertheless, the later use of these techniques, possibly in conjunction with other transformations [5], can serve as a valuable complement to our methodologies.

[2] 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.

[3] However, module selection has since been incorporated [10], and conditionals and register area are currently being investigated.

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

Fig. 4. 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

Fig. 5. Voyager's Resource-Constrained 3D Scheduling (RCS-3D) Methodology

problems as *efficiently* as possible so that an optimal solution is found in a reasonable amount of time.

To solve the scheduling problem, both methodologies use an Integer Linear Programming (ILP) formulation (Section IV) that was developed after a careful formal analysis of that problem. This analysis was presented earlier in [9], where we proved that this formulation, in particular the formulation of the TRCS problem, was well-structured and can be solved efficiently.

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.

A. Time-Constrained 3D Scheduling (TCS-3D)

Voyager's methodology for solving the time-constrained 3D scheduling problem is outlined in Figure 4. 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 III), based on the execution delays of the relevant functional units in the module library, and for chained designs, on the structure of the DFG. For the EWF and the module library described earlier, 10 candidate clock lengths would be generated (in the absence of chaining). 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 IV) 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 V). 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 [9]. Finally, the functional unit lower bounds are highly accurate [11] (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 VII).

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.

B. Resource-Constrained 3D Scheduling (RCS-3D)

Voyager's methodology for solving the resource-constrained 3D scheduling problem is similar (see Figure 5). This methodology reads in a resource constraint, and generates a minimal set of candidate clocks using the clock length determination algorithm described in Section III. 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 IV) is used as follows. First, a tight lower bound on the overall length of the schedule is computed (Section VI). 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 VII).

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.

C. 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 is well-structured [9], and therefore usually finds an optimal solution with few branches.

III. Determining Candidate Clock Lengths

One of the most important parameters needed by any scheduling algorithm is the length of the system clock [4].

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. [13]. However, all such quantities are 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 [14]. Here the clock length is assumed to have 3 components: datapath delay, control delay, and wire delay. For the moment, we will use only the datapath delays to determine the clock length, and will ignore the control and wire delays, realizing that the actual clock length will be longer due to those delays; this limitation will be addressed later in Section III-C. We will also assume 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 $t_s(reg)$ and $t_p(reg)$ be the setup time and propagation delay of the registers, and let $t_p(interconnect)$ be the interconnect propagation delay. 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

$$d_k = delay\,(k) + t_s(reg) + t_p(reg) + t_p(interconnect).$$

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.

The remainder of this section describes Voyager's methodology for choosing a set of provably non-inferior candidate clock lengths. Section III-A describes the methodology in the absence of chaining, and then Section III-B describes the extensions necessary to support chaining. Finally, Section III-C discusses how controller delay could be included as well.

A. Determining Candidate Clock Lengths in the Absence of Chaining

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, and $\bar{c}$, the largest d_k (or maximum chain length when chaining is considered). One of the goals of the Voyager 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 [15], [16] use the delay of the slowest functional unit as the estimated clock length. A

[4] In this section, we will assume that a fixed clock length is used for every control step, in contrast to the method presented by Rouzeyre in [12] in which the clock length is dynamically changed during execution to reduce the slack in the current control step.

more realistic approach is used in [7], 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 [17], 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 [18], 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.

The new clock period c' can be found as $c' = c - \epsilon$, where $\epsilon = \min\{s_k(c)/\lceil d_k/c \rceil \mid k \in K\}$. Substituting Definition 2 for $s_k(c)$, $\epsilon = \min\{c - d_k/\lceil d_k/c \rceil \mid k \in K\}$, which in turn implies $c' = \max\{d_k/\lceil d_k/c \rceil \mid k \in K\}$. This value of c' can be used to derive the following relation:

$$\lceil d_k/c' \rceil = \min\{\lceil d_k/c \rceil \mid k \in K\} \leq \lceil d_k/c \rceil.$$

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, since c' is less than c, it will lead to 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 $[\underline{c}, \bar{c}]$.

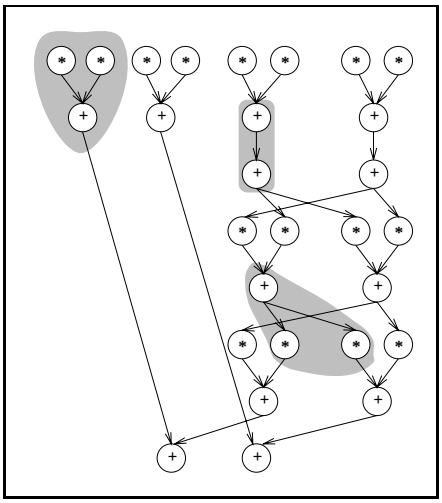

Fig. 6. The AR-Lattice Filter DFG

For example, consider the AR lattice filter [19] benchmark (see Figure 6), the VDP100 module library [7], [8] (with datapath delays of 48ns for addition and 163ns for multiplication), and a technology lower bound on the clock length of 19ns. In the absence of chaining, Voyager's design space exploration methodologies explore those integer clock lengths in the range [19,163] that are ceilings of the integral divisors of 163 and 48, i.e. $CK = \{163, 82, 55, 48, 41, 33, 28, 24, 21, 19\}$.

B. Determining Candidate Clock Lengths When Chaining

First mentioned in high-level synthesis literature in [20], *chaining* refers to the technique of scheduling two or more data-dependent operations into the same control step, using the otherwise wasted "slack" time that remains in the clock period after the first operation finishes. Commonly used in industry, these chains of two or more operations may include a variety of arithmetic operations, logic operations, etc. At the register-transfer level, the chain is implemented by connecting the output of one functional unit directly to the input of the following functional unit (i.e., without an intervening register).

Definition 3: Let $t_s(reg)$ and $t_p(reg)$ be the setup time and propagation delay of the registers, and let $t_p(interconnect)$ be the interconnect propagation delay. If the total delay of the functional units involved in chain ch is denoted as $delay\,(ch)$, the *chain delay* d_{ch} for a register-to-register transfer executing a chain ch is given as

$$d_{ch} = delay\,(ch) + t_s(reg) + t_p(reg) + t_p(interconnect).$$

In the discussion of chaining that follows, the *length* of the chain will denote the number of operations that are chained together in sequence. For simplicity, the discussion will be limited to chains of length 2, although the methodologies can be extended to handle longer chains at the cost of a larger solution space and a corresponding increase in execution time.

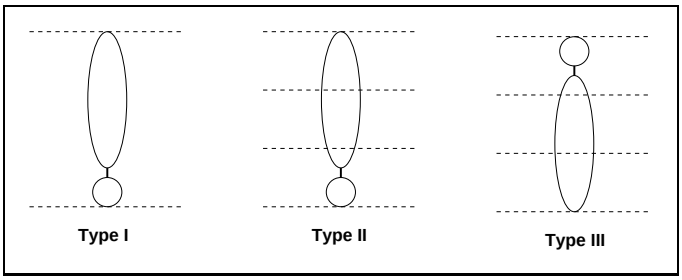

Fig. 7. Types of Chaining

```
chain_clocks(node, chain_len, curr_delay):
increment chain_len
if chain_len < MAX_CHAIN_LEN then
      new_chain = curr_delay + d_k(node)
      append new_chain to set of possible chain delays
      for each successor j of node in the DFG
          call chain_clocks(j,chain_len,new_chain)
      end
```

Fig. 8. Algorithm for Finding All Possible Chain Delays in a DFG

We categorize chaining into three types, since treating each each type differently allows us to more fully reduce the set of candidate clock lengths. These three types [5] are summarized as follows, and are illustrated in Figure 7:

- *type I chaining* – the entire chain must execute within a single control step
- *type II chaining* – the chain may execute over multiple control steps (as may one or more of the operations), but the final operation in the chain must start and finish within the last control step
- *type III chaining* – the chain may execute over multiple control steps (as may one or more of the operations), but the first operation in the chain must start and finish within the first control step.

Note that type I chaining (the classical form of chaining) is a special case of type II and type III chaining.

To determine the minimal set of candidate clock lengths CK when chaining is allowed, the set of operation execution delays D must be considered in conjunction with the set D_{ch} of possible chain delays in a given DFG. Thus, the task of determining the candidate clock lengths for resource-constrained scheduling with chaining consists of two steps: (1) finding the set D_{ch} of all possible chain delays due to potential chains in the DFG, and (2) using D and D_{ch} to determine a set CK' that represents the the minimal set of candidate clock lengths that must be explored for the given type of chaining.

B.1 Finding All Possible Chain Delays in the DFG

To find all possible chain delays due to potential chains in the DFG, Voyager performs a depth-first search of the DFG, using a lookahead equal to the maximum chain length allowed. This recursive algorithm, shown in Figure 8, finds all the chain delays in a DFG when invoked with the call **chain_clocks(***source***, 0, 0)**. The algorithm runs in $O(n^l)$ time, where n is the number of operations in the DFG, and l is the maximum allowable chain length.

For the AR-lattice filter benchmark [19] and the VDP100 [8] module library, the algorithm returns the set $D_{ch} = \{211, 96\}$, corresponding to the operation sequences $\{*, +\}$ and $\{+, +\}$.

[5] Note that we do not consider type IV chaining, in which all operations in the chain are multicycled, since this is not really chaining – one operation is not completed within the wasted slack left by another.

B.2 Candidate Clock Lengths for Type I Chaining

To determine the set of candidate clock lengths for type I chaining, the algorithm initially sets $CK' = div(D) \cup D_{ch}$, and then removes those clock lengths that cannot lead to type I chains. [6] For type I chaining, the entire chain must be scheduled into a single control step, and none of the chained operations may be multi-cycled, which implies that all clock lengths in CK' must be at least as long as the shortest chain length ($\geq min(D_{ch})$), but can be no longer than the maximum chain delay ($\bar{c}$). Continuing the example, the algorithm therefore initially sets $CK' = \{211, 163, 96, 82, 55, 48, 33, 28, 24, 21, 19\}$, and then removes all clock lengths shorter than 96ns. This process leaves the final set of candidate clock lengths to explore as $CK' = \{211, 163, 96\}$.

B.3 Candidate Clock Lengths for Type II Chaining

In the case of type II chaining, the algorithm first sets $CK' = div(D \cup D_{ch})$, and then removes those clock lengths that cannot lead to type II chains. For a given clock length, a type II chain can only exist if the second operation in the chain can be scheduled in the slack remaining after the first operation in the chain, which implies that all clock lengths in CK' must be longer than shortest operation delay ($> min(D)$). Moreover, the longest chain delay ($max(D_{ch})$) can form only a type I chain, and can also be removed from CK'. Using the AR-lattice filter example, this means that the algorithm initially sets $CK' = div(\{211, 163, 96, 48\})$, and then removes 211ns and all clock lengths less than or equal to 48ns from this set, leaving the set $CK' = \{163, 106, 96, 82, 71, 55, 53\}$.

However, this set of candidate clock lengths can be pruned even further, because many of these candidate clock lengths will actually not support a type II chain. For example, consider the clock length of 55ns, which is too long to multi-cycle an addition, and thus can only multi-cycle a multiplication. However, the slack left after a multi-cycle multiplication is $55 * \lceil 163/55 \rceil - 163 = 2ns$, which is less than d_{add}, so the chain can not be completed and 55 should be removed from CK'. Applying this process further to the AR-lattice filter example gives the minimal set of candidate clock lengths to explore as $CK' = \{106, 71, 53\}$.

[6] Note that we only consider D_{ch}, rather than $div(D_{ch})$, since type I chaining does not allow chains to be scheduled over multiple cycles.

B.4 Candidate Clock Lengths for Type III Chaining

As with type II chaining, for type III chaining the algorithm first sets $CK' = div(D \cup D_{ch})$, and then removes those clock lengths that cannot lead to type III. Under type III chaining, the only restriction is that the first operation must begin and end within a single control step, which implies that all clock lengths in CK' must be longer than the shortest operation delay ($> min(D)$). Again, the longest chain delay ($max(D_{ch})$) can also be removed, as it can form only a type I chain. Using the AR-lattice filter example, this process gives the final set of candidate clock lengths to explore as $CK' = \{163, 106, 96, 82, 71, 55, 53\}$.

C. Considering Controller Delay

In Definition 1, the register-to-register delay d_k of a type-k functional unit was defined as consisting of the functional unit delay, the register setup and propagation delay, and the interconnect propagation delay. However, the final system clock length depends not only on data path components, but also on the controller delays. Therefore the definition of d_k should be modified to include the state register setup and propagation delays, and the propagation delays through the control logic and next state logic. For a non-pipelined PLA-based controller, d_k can be redefined as:

$$\begin{aligned} d_k = delay\,(k) \quad & + t_s(reg) + t_p(reg) + t_p(interconnect) \\ & + t_s(statereg) + t_p(statereg) \\ & + t_p(controllogic) + t_p(nextstatelogic). \end{aligned}$$

In the presence of chaining, the chain-delay d_{ch} can be redefined in a similar manner. A similar definition of register-transfer delay is given by Gajski *et al.* [14], [21]. Modifications can also be made for other (e.g., microcoded or pipelined) types of controllers.

Since the additional delays are not available at the beginning of the design process, most of the previous work in high-level synthesis has concentrated solely on the functional unit delay ($delay(k)$), or possibly the functional unit and register delays, arguing that the interconnect and controller delays can not be accurately determined before scheduling. However, even rough estimates of those additional delays, determined from a previous iteration of the design process and treated as constants in the current iteration, can sometimes be exploited to produce a better schedule.

The additional controller-related delays can be accurately determined after logic synthesis [22], or estimated from an RT-level design [23], and then used in the current design iteration. For example, the controller delays determined in the previous iteration can be used in the current iteration [24], or the system can attempt to predict the incremental change over the previous delays (the work presented in [25] is a first step in this direction). Similarly, the interconnect delays can be determined from the maximum value in the previous iteration [24], or from models of the layout tools [26], and the register delays can be treated as constants, or measured more accurately in conjunction with a detailed retiming model [27].

Once the controller and interconnect related delays have been determined, the new values of d_k and d_{ch} can be used with the techniques described in III-A and III-B to more accurately calculate the candidate clock lengths. For pipelined controllers, care must be taken to prevent chaining over conditional statements.

IV. 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 [9], we were able to develop well-structured formulations of those problems, in particular the TRCS problem. We began by characterizing the set of feasible schedules in terms of assignment, precedence, and resource constraints. We then used polyhedral theory to analyze these constraints to determine the structure of the corresponding scheduling polytope, and we proved that our precedence constraints lead to the tightest possible description of that polytope. Finally, once this analysis was complete, that structure was exploited to develop a provably well-structured Integer Linear Programming (ILP) formulation of the TRCS problem.

This section briefly introduces Voyager's ILP formulation of the scheduling problem, and describes the modifications necessary to support chaining [28]. Most previous ILP formulations have considered only type I chaining [29], [30], or a combination of types I and II [31]. An ILP formulation that supports all three types of chaining is presented in [32], but the encompassing methodology does not allow for multicycling of non-chained operations due to clock length restrictions in the ILP formulation. In contrast, this section describes how all three types of chaining can be incorporated into our ILP formulation while still allowing multicycling of non-chained operations. For a more detailed description of this formulation in the absence of chaining, see [9].

A. ILP Formulation of the Scheduling 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

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

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.

The TCS and RCS problems can be defined similarly. Since the RCSs problem minimizes the number of control steps, a *sink* operation o_d is introduced, and the formulation ensures that it is scheduled in the last control step by making it the successor of all operations that had no successors in the original DFG. The total number of control steps can then computed as $\sum_{s \in S_d} s x_{d,s}$, and this quantity can be minimized in the objective function of the ILP formulation

$$\min \{ \sum_{s \in S_d} s x_{d,s} \mid \mathbf{M}_a \mathbf{x} = \mathbf{1} \; ; \; \mathbf{M}_t \mathbf{x} \leq \mathbf{1} \; ; \; \mathbf{M}_r \mathbf{x} \leq \mathbf{m} \; ; \; \mathbf{x} \text{ integer} \}.$$

To support chaining in these formulations, the precedence constraints described above must be modified. Section IV-B first discusses some modifications to the schedule intervals that are necessary, and then Section IV-C presents the new precedence constraints.

B. Modifying Schedule Intervals to Support Chaining

Before the new precedence constraints can be formulated, new schedule intervals S_i must be determined for each operation o_i, $i \in I$ in the presence of chaining. In the absence of chaining, given the resource constraints on the number of FUs of each type, a heuristic such as list scheduling can be used to find an upper bound on the total number of csteps, and then the schedule interval S_i can be determined for each operation o_i, $i \in I$ as $S_i = [asap_i, alap_i]$.

However, these ASAP and ALAP times must be determined differently when chaining is allowed. This section discusses only the modifications necessary to determine the ASAP values when chaining is supported; the modifications for the ALAP values are analogous.

In the case of type I chaining, the ASAP cstep for each operation can be determined by placing as many operations as the maximum chain length will allow into the current cstep, while ensuring that the sum of the delays of these chained operations does not exceed the chosen clock.

However, for type II or type III chaining, the situation becomes more complex due to the interplay between the maximum chain length and the delays of chained operations. Given a DFG, consider an operation sequence $i \rightarrow j \rightarrow k$ in which i and j could be chained, as could j and k; the ASAP time for k would then be calculated as:

$$asap_k = asap_i + \min\{\lceil d_i/c \rceil + \lceil d_j/c \rceil - 1, \lceil (d_i + d_j)/c \rceil\}$$

where $\lceil d_i/c \rceil + \lceil d_j/c \rceil - 1$ represents the case of not chaining operations i and j, while $\lceil (d_i + d_j)/c \rceil$ represents the case where i and j are chained (thus not allowing j and k to be chained). The minimum value of this pair then indicates whether or not to chain i and j (note that this minimum is determined independent of k) when generating an ASAP schedule.

C. Modifying Precedence Constraints to Support Chaining

Once the new schedule intervals have been determined, the precedence constraints can be modified to support all three types of chaining, as described in this section.

In the original DFG, each arc a_{ij} indicates a precedence relation

$$a_{ij} \Rightarrow o_i \xrightarrow{d_i} o_j.$$

The quantity d_i denotes a continuous-time delay d_i that implies a continuous-time relation $t(j) \geq t(i) + d_i$, where $t(i)$ and $t(j)$ denote the instants in continuous time at which operation o_i and o_j respectively start execution.

However, during scheduling, time is measured in discrete control steps, or clocks, rather than in continuous time. In particular, when a clock of length c is used without chaining, the previous continuous-time relation is replaced with the following discrete-time relation:

$$s(j) \geq s(i) + \lceil d_i/c \rceil,$$

where $s(i)$ and $s(j)$ denote the control steps in which o_i and o_j respectively start execution. Note that $s(j) \geq s(i) + 1$, which is consistent with the assumption that o_j can not be chained with o_i.

When chaining is supported, the above discrete-time relation is modified to

$$s(j) \geq s(i) + \lfloor d_i/c \rfloor. \tag{1}$$

to allow o_j to begin in the same control step as the one in which o_i finishes execution.

Now consider a sequence of three operations i, j, and k that indicates the following precedence relation:

$$o_i \xrightarrow{d_i} o_j \text{ and } o_j \xrightarrow{d_j} o_k.$$

If $d_i + d_j \leq c$ and $d_j + d_k \leq c$, then chaining requires that $s(j) \geq s(i)$ and $s(k) \geq s(j)$, which implies that $s(k) \geq s(i)$. However, if $d_i + d_j + d_k > c$, then the above implication will lead to an infeasible schedule.

Therefore, to ensure correctness, the following new precedence relation must be added to the DFG:

$$a_{ik} \Rightarrow o_i \xrightarrow{d_i + d_j} o_k.$$

In other words, from each operation o_i, arcs a_{ik} must be added to its nearest successors o_k that can not be chained with it. Determining whether o_k can be chained with o_i depends on the type of chaining and the maximum chain-length; so the decision of adding arc a_{ik} is also affected by the type of chaining and the maximum chain-length.

After adding the new arcs for chaining, we use the modified discrete-time relation (1) to generate the precedence constraints in a similar manner as the non-chained precedence constraints. A detailed description of the non-chained precedence constraints can be found in [9].

V. Bounding the Number of Functional Units to Solve TCS More Efficiently

As discussed earlier in Section II, 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, these bounds must be computed efficiently.

This FU lower-bounding problem can be viewed as a relaxation of the FU minimization problem. While many different FU lower-bounding problems can be formed by relaxing the minimization problem in different ways, most are

formed by relaxing the precedence constraints between operations. Precedence constraints have been relaxed in [19] (and a similar relaxation in [33]), and in the tighter relaxations described in [34], [35], [36], and [37] (based on a method originally proposed by Fernández and Bussell in [38, Theorem 1]).

A different approach, described more formally in [11], is used in Voyager. This approach starts with an ILP formulation of the FU minimization problem (minimize the number m_k of FUs of type $k \in K$). The problem is then relaxed to a generic description of an entire class of FU lower-bounding problems (the problems above are special cases of this generic class). From this class, the FU lower-bounding problem that produces the tightest possible bound is selected and solved. This approach formalizes an entire class of FU lower-bounding problems and is guaranteed to produce the tightest possible bound in that class; this bound was verified to be exact in most of our experiments.

Furthermore, the solution to this FU lower-bounding problem can be found in polynomial time by solving at most two LP's, even though the original formulation was an ILP formulation. Such an LP-based relaxation is chosen because we want as tight as possible bounds to increase the efficiency of solving the TCS problem. However, Voyager also has a suite of more efficient heuristic algorithms [10] that may produce less accurate bounds and are suitable for a quick first pass over the design space.

VI. Bounding the Length of the Schedule to Solve RCS More Efficiently

This section presents a method of generating a tight lower bound on the schedule length, so that the RCS problem can be solved more efficiently. The method is similar in principle to the method presented in the previous section for solving the FU lower-bounding problem.

One early formulation of the schedule length lower-bounding problem in presence of resource constraints is presented in [19]; however, the bounds produced by that approach are very loose. More recent algorithms that produce tighter bounds are those in [39] and [37] (based on Jackson's earliest deadline rule (ED-Rule) [40]), and those in [34] and [36] (based on a theorem originally given by Fernández and Bussell in [38, Theorem 2]). Furthermore, those algorithms can be applied iteratively (Hu *et al.* apply Fernández's Theorem 2 in [41], and Langevin applies ED-Rule in [42]), producing even tighter bounds, although at the cost of increased algorithmic complexity.

In much the same manner as FU-lower bounding, the ILP formulation of the schedule-length minimization problem can be relaxed to a generic description of an entire class of schedule-length lower-bounding problems. From this class, the lower-bounding problem that produces the tightest possible bound (the problems above are special cases of this generic class) is chosen and solved. This 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.

Clock	Time		(Mult, Add)
	Csteps	ns	
Time Constraint = 902ns			
82	11	902	(4, 2)
55	15	825	(4, 2)
48	18	864	(5, 2)
41	22	902	(4, 2)
33	26	858	(4, 2)
28	30	840	(4, 2)
24	34	816	(4, 2)
21	41	861	(4, 2)
19	45	855	(4, 2)
Time Constraint = 760ns (tightest)			
24	31	744	(6, 2)

TABLE II
AR – TCS-3D Results

Clock	2 *, 1 +		2 *, 4 +		4 *, 2 +		6 *, 3 +		6 *, 4 +	
	Cs	ns	Cs	ns	Cs	ns	Cs	ns	Cs	ns
163	13	2119	10	1630	8	1304	8	1304	8	1304
82	18	1476	18	1476	11	902	11	902	11	902
55	26	1430	26	**1430**	15	825	14	770	14	**770**
48	LB	1440	LB	1440	LB	864	LB	816	LB	816
41	LB	1435	LB	1435	LB	861	LB	779	LB	779
33	LB	1452	LB	1452	LB	858	LB	792	LB	792
28	LB	1456	LB	1456	LB	840	LB	784	LB	784
24	LB	1440	LB	1440	34	816	31	744	31	**744**
21	LB	1449	LB	1449	LB	819	LB	756	LB	756
19	LB	1444	LB	1444	LB	817	LB	760	LB	760

TABLE III
AR – RCS-3D Results Without Chaining

VII. Experimental Results

To demonstrate the accuracy and performance of Voyager's 3D scheduling methodology, we conducted a series of experiments using the well-known AR-lattice Filter (AR) [19], Elliptic Wave Filter (EWF) [6, p.206], and Discrete Cosine Transform (DCT) [43] benchmarks. We used the VDP100 module library from [7], [8], 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) with and without chaining, using the methodologies presented in Section II.

A. AR Filter

The TCS-3D results for the AR filter are presented in Table II. 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 902ns, eight clock lengths (82ns, 55ns, 41ns, 33ns, 28ns, 24ns, 21ns, and 19ns) led to the minimum number of functional units. Of these, the schedule for the 82ns clock ($\lceil d_{mult}/2 \rceil$) requires the fewest control steps (and thus potentially the smallest controller), and so

Clock	2 *, 1 +		2 *, 4 +		4 *, 2 +		6 *, 3 +		6 *, 4 +	
	Cs	ns	Cs	ns	Cs	ns	Cs	ns	Cs	ns
211	12	2532	9	1899	6	1266	5	1055	4	**844**
163	13	2119	9	**1467**	LB	1304	LB	1141	LB	978
96	18	1728	LB	1536	11	1056	LB	1056	LB	864

TABLE IV

AR – RCS-3D Results for Type I Chaining

Clock	2 *, 1 +		2*, 4 +		4 *, 2 +		6 *, 3 +		6 *, 4 +	
	Cs	ns	Cs	ns	Cs	ns	Cs	ns	Cs	ns
106	17	1802	17	1802	9	954	8	848	8	848
71	LB	1491	LB	1491	LB	923	11	781	11	781
53	LB	1431	LB	1431	LB	848	14	742	14	**742**

TABLE V

AR – RCS-3D Results for Type II Chaining

Clock	2 *, 1 +		2 *, 4 +		4 *, 2 +		6 *, 3 +		6 *, 4 +	
	Cs	ns	Cs	ns	Cs	ns	Cs	ns	Cs	ns
163	13	2119	9	**1467**	7	1141	7	1141	6	978
106	17	1802	LB	1484	10	1060	9	954	LB	848
96	18	1728	LB	1536	11	1056	LB	960	LB	864
82	18	1476	LB	1476	11	902	11	902	LB	902
71	LB	1491	LB	1491	LB	923	12	852	11	781
55	26	1430	26	1430	15	825	14	770	14	**770**
53	LB	1431	LB	1431	LB	848	LB	795	LB	795

TABLE VI

AR – RCS-3D Results for Type III Chaining

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 VII

EWF – TCS-3D Results

would be preferable. Note that the 48ns clock – one of the "obvious" *ad hoc* guesses (d_{add}) – requires an additional multiplier.

To find the fastest possible design, the critical path length was used to derive the tightest possible time constraint of 760ns. For this time constraint, only one clock length – 24ns – led to a feasible schedule, and thus to the optimal 3D schedule.

The RCS-3D results for the AR filter are presented in Tables III-VI. 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 II-B, 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.

In the absence of chaining, schedule lengths are shown for every candidate clock length in Table III. In this table, the fastest schedules correspond to clock lengths of 55ns, and when sufficient resources are available, 24ns. 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 optimal 3D schedule, because the smaller clock lengths tend to reduce the operation slack.

Similarly, in the presence of chaining, schedule lengths are shown for every candidate clock length in Tables IV-VI. For a given clock length (such as 163ns), chaining usually improved the schedule when there were sufficient resources available, but provided no improvement when the number of resources was small. Furthermore, for some clock lengths and resource constraints, one type of chaining provided the largest improvement, while for other clock lengths and resource constraints, another type provided the largest improvement.

Finally, looking at all these results over all candidate clock lengths, note that type II chaining gave the fastest overall chained schedule (742ns), slightly faster than the fastest unchained schedule (744ns), and with half the number of control steps. Type III chaining's performance was poorer (770ns), but at least tied with the unchained schedule at that same clock length (55ns). The best schedule from type I chaining (the "standard" form of chaining), however, was considerably worse than the best unchained schedule (844ns vs. 744ns).

B. Elliptic Wave Filter (EWF)

The TCS-3D results for the EWF are presented in Table VII. Again, they show, for each of two time constraints, those clock lengths from the candidate set that lead to a feasible schedule.

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 this is a different clock length than the one chosen for the AR filter, illustrating the importance of taking the structure of the DFG into account.

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 optimal 3D schedule.

The RCS-3D results for the EWF are shown in Tables VIII-XI. In the absence of chaining, the 24ns and 55ns clock lengths correspond to the fastest schedules. Again, for a given clock length, chaining usually improved the schedule when there were sufficient resources available. However, neither form of chaining was able able to find an

Clock	ASAP Cs	ASAP ns	1 *, 2 + Cs	1 *, 2 + ns	2 *, 2 + Cs	2 *, 2 + ns	3 *, 3 + Cs	3 *, 3 + 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 VIII
EWF – RCS-3D Results Without Chaining

Clock	ASAP Cs	ASAP ns	1 *, 2 + Cs	1 *, 2 + ns	2 *, 2 + Cs	2 *, 2 + ns	3 *, 3 + Cs	3 *, 3 + ns
211	7	1477	14	2954	14	2954	10	2110
163	9	1467	15	2445	15	2445	10	1630
96	12	1152	20	**1920**	16	1536	13	**1248**

TABLE IX
EWF – RCS-3D Results for Type I Chaining

Clock	ASAP Cs	ASAP ns	1 *, 2 + Cs	1 *, 2 + ns	2 *, 2 + Cs	2 *, 2 + ns	3 *, 3 + Cs	3 *, 3 + ns
106	11	1166	19	**2014**	15	1590	12	**1272**
71	17	1207	LB	1775	19	1349	LB	1278
53	20	1060	LB	1643	LB	1219	LB	1166

TABLE X
EWF – RCS-3D Results for Type II Chaining

Clock	ASAP Cs	ASAP ns	1 *, 2 + Cs	1 *, 2 + ns	2 *, 2 + Cs	2 *, 2 + ns	3 *, 3 + Cs	3 *, 3 + ns
163	9	1467	15	2445	15	2445	10	1630
106	11	1166	19	2014	15	1590	13	1378
96	12	1152	20	1920	16	1536	13	1248
82	17	1394	21	1722	18	1476	LB	1394
71	17	1207	LB	1775	19	1349	LB	1420
55	20	1100	29	**1595**	22	1210	21	**1155**
53	20	1060	LB	1643	LB	1219	LB	1166

TABLE XI
EWF – RCS-3D Results for Type III Chaining

overall faster schedule than the fastest unchained schedule.

C. Discrete Cosine Transform (DCT)

The TCS-3D results for the DCT are presented in Table XII. 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 optimal 3D schedule.

The RCS-3D results for the DCT are presented in Table XIII. In the absence of chaining, the 56ns clock length (d_{sub}) corresponds to the fastest schedule. This time, not only could type I chaining not find an overall faster schedule than the fastest unchained schedule, but it could not even improve the schedule for a given clock length over the unchained schedule, probably due to the severe resource constraints.

D. 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 V and VI. Finally, the TRCS formulation used in Voyager 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 II-A and II-B.

VIII. Summary and Future Work

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. It can optimally solve medium-sized problems in seconds, as opposed to more conventional techniques that might require hours. Thus it eliminates the

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 XII
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

5 Mult, 3 Add, 2 Sub		
Clock	Csteps	ns
219	9	1971
211	9	1899
163	9	1467
104	10	1040
96	10	**960**

TABLE XIII
DCT – RCS-3D RESULTS WITHOUT CHAINING, WITH TYPE I CHAINING

need to replace exact techniques with heuristic algorithms or estimates in an effort to obtain acceptable performance – a claim not possible with any existing solution methodology.

Although this methodology is a step toward solving the scheduling problem in the context of a larger problem, much work remains for the future. The next step is to carefully extend the methodology to support conditionals, module selection and type mapping, and wiring and register costs.

REFERENCES

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

[2] C. E. Leiserson and J. B. Saxe, "Retiming synchronous circuitry," *Algorithmica*, no. 6, pp. 5–35, 1991.

[3] R. B. Deokar and S. S. Sapatnekar, "A fresh look at retiming via clock skew optimization," in [44], pp. 310–315.

[4] P. Jha, S. Parameswaran, and N. Dutt, "Reclocking for High Level Synthesis," in *Proc. of the Asia-South Pacific Conference on Design Automation (ASP-DAC)*, (Makuhari Messe, Chiba, Japan), IEEE Computer Society Press, Aug. 29-Sept. 1 1995.

[5] M. Potkonjak and J. M. Rabaey, "Optimizing resource utilization using transformations," *IEEE Transactions on Computer-Aided Design*, vol. 13, pp. 277–292, Mar. 1994.

[6] 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.

[7] S. Narayan and D. D. Gajski, "System Clock Estimation based on Clock Slack Minimization," in [45], pp. 66–71.

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

[9] 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.

[10] S. A. Blythe and R. A. Walker, "Towards a Practical Methodology for Completely Characterizing the Optimal Design Space," in *Proc. of the 9th International Symposium on System-Level Synthesis*, (La Jolla, California), IEEE Computer Society Press, Nov. 6-8 1996.

[11] S. Chaudhuri and R. A. Walker, "Computing Lower Bounds on Functional Units before Scheduling," *IEEE Transactions on VLSI Systems*, vol. 4, pp. 273–279, June 1996.

[12] B. Rouzeyre, D. Dupont, and G. Sagnes, "Component Selection, Scheduling and Control Schemes for High Level Synthesis," in [46].

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

[14] V. Chaiyakul, A. C.-H. Wu, and D. D. Gajski, "Timing Models for High Level Synthesis," in [45], pp. 60–65.

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

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

[17] 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*, (Santa Clara, California), pp. 518–521, IEEE Computer Society Press, Nov. 7-11 1993.

[18] 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.*, (Singapore), pp. 2200–2203, IEEE Computer Society Press, June 11-14 1991.

[19] 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.

[20] B. M. Pangrle and D. D. Gajski, "Design Tools for Intelligent Silicon Compilation," *IEEE Transactions on Computer-Aided Design*, vol. 6, pp. 1098–112, Nov. 1987.

[21] D. D. Gajski, N. D. Dutt, A. Wu, and S. Lin, *High-Level Synthesis: Introduction to Chip and System Design*. 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.

[22] B. Gregory, D. MacMillen, and D. Fogg, "ISIS: A System for Performance Driven Resource Sharing," in [47], pp. 285–290.

[23] C. Ramachandran, F. J. Kurdahi, D. D. Gajski, V. Chaiyakul, and A. C.-H. Wu, "Accurate layout area and delay modelling for system level design," in *Proc. of the IEEE/ACM International Conference on Computer-Aided Design*, (Santa Clara, California), pp. 355–361, IEEE Computer Society Press, Nov. 8-12 1992.

[24] D. Mintz and C. Dangelo, "Timing Estimation for Behavioral Descriptions," in *Proc. of the 7th International Symposium on High-Level Synthesis*, (Niagara-on-the-Lake, Canada), pp. 42–47, IEEE Computer Society Press, May 18-20 1994.

[25] C. Ramachandran and F. J. Kurdahi, "Incorporating the controller effects during register transfer level synthesis," in [46], pp. 308–313.

[26] D. Gelosh and D. E. Setliff, "Deriving Efficient Area and Delay Estimates by Modelling Layout Tools," in [44], pp. 402–407.

[27] K. N. Lalgudi and M. C. Papaefthymiou, "DELAY: An Efficient Tool for Retiming with Realistic Delay Modeling," in [44], pp. 304 – 309.

[28] R. M. Russell, "The CRAY-1 Computer System," *Communications of the ACM*, vol. 21, pp. 63–72, Jan. 1978.

[29] 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.

[30] B. Landwehr, P. Marwedel, and R. Dömer, "OSCAR: Optimum Simultaneous Scheduling, Allocation adn Resource Binding Based on Integer Programing," in [46].

[31] M. Rim and R. Jain, "Estimating lower-bound performance of schedules using a relaxation technique," in *Proc. of the IEEE International Conference on Computer Design*, (Cambridge, Massachusetts), pp. 290–294, IEEE Computer Society Press, Oct. 11-14 1992.

[32] C. H. Gebotys, "Optimal Scheduling and Allocation of Embedded VLSI Chips," in [47], pp. 116–119.

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

[34] 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.

[35] 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.

[36] 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 [48], pp. 21–24.

[37] 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.

[38] 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.

[39] M. Rim and R. Jain, "Lower-Bound Performance Estimation for the High-Level Synthesis Scheduling Problem," *IEEE Transactions on Computer-Aided Design*, vol. 13, pp. 451–458, Apr. 1994.

[40] J. Błażewicz, "Simple Algorithms for Multiprocessor Scheduling to Meet Deadlines," *Information Processing Letters*, vol. 6, pp. 162 – 164, Oct. 1977.

[41] 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.

[42] M. Langevin and E. Cerny, "A Recursive Technique for Computing Lower-Bound Performance of Schedules," in [48], pp. 16–20.

[43] 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.

[44] *Proc. of the 32nd ACM/IEEE Design Automation Conf.*, (San Fransisco, California), IEEE Computer Society Press, June 12-16 1995.

[45] *Proc. of the European Design Automation Conference (Euro-DAC)*, (Hamburg, Germany), IEEE Computer Society Press, Feb. 1992.

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

[47] *Proc. of the 29th ACM/IEEE Design Automation Conf.*, (Anaheim, California), IEEE Computer Society Press, June 8-12 1992.

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