Efficient Optimal Design Space Characterization Methodologies

STEPHEN A. BLYTHE
Parks College of Saint Louis University
and
ROBERT A. WALKER
Kent State University

One of the primary advantages of a high-level synthesis system is its ability to explore the design space. This paper presents several methodologies for design space exploration that compute **all** optimal tradeoff points for the combined problem of scheduling, clock length determination, and module selection. We discuss how each methodology takes advantage of both the structure within the design space itself as well as the structure of, and interaction between, each of the three subproblems.

Categories and Subject Descriptors: J.6 [**COMPUTER-AIDED ENGINEERING**]: Computer-aided design (CAD)

General Terms: High-Level Synthesis, Design Space Exploration

Additional Key Words and Phrases: scheduling, module selection, clock length determination, efficient searching, bounding

1. INTRODUCTION

For many years, one of the most compelling reasons for developing high-level synthesis systems [Gajski et al. 1994] [De Micheli 1994] has been the desire to quickly explore a wide range of designs for the same behavioral description. Given a set of designs, two metrics are commonly used to evaluate their quality: area (ideally total area, but often only functional unit area), and time (the schedule length, or latency). Finding the optimal tradeoff curve between these two metrics is called *design space exploration*. This problem is generally considered too difficult and time consuming to solve optimally , so the problem is usually limited to computing either lower bounds [Timmer et al. 1993] or estimates [Chen and Jeng 1991] on the optimal tradeoff curve for some set of time or resource constraints.

The design space exploration methodology described here goes beyond traditional design space exploration in several ways. First, *all* optimal tradeoff points are computed so that the design space is *completely* characterized. Second, these optimal tradeoff points represent optimal *solutions* to the time-constrained scheduling (TCS) and resource-constrained scheduling (RCS) problems, rather than lower bounds or estimates. Third, the tradeoff points are computed in a manner that supports more realistic module libraries by incorporating clock length determination and module selection into the methodology. Finally, these tradeoff points are computed in an efficient manner through careful pruning of the search space during the design cycle. Although the work presented here does not consider issues associated with the controller, registers, or interconnect, they could be incorporated (at

---

This material is based upon work supported by the National Science Foundation under Grant No. MIP-9423953 while the authors were with the Department of Computer Science at Rensselaer Polytechnic Institute.

Authors' addresses: Stephen A. Blythe, Department of Computer Science, Parks College of Saint Louis University, PO Box 56907, St. Louis, MO 63156-0907 ; Robert A. Walker, Department of Mathematics and Computer Science, Kent State University, Kent, OH 44242.

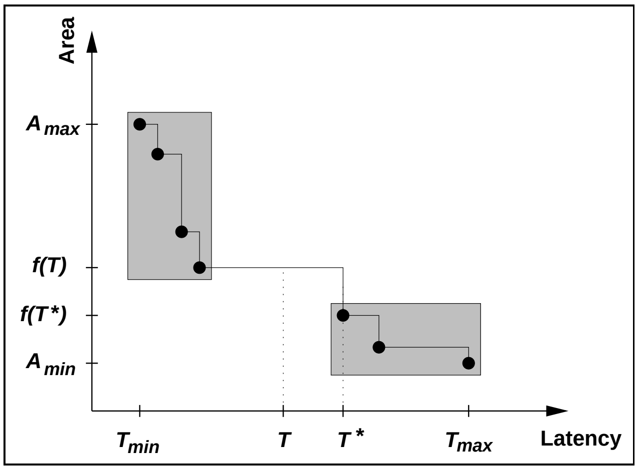

Fig. 1. Example design space showing Pareto points. The shaded regions show the two distinct clusters of Pareto points that many tradeoff curves exhibit.

the expense of optimality) using techniques similar to those described by Ohm et. al. [Ohm et al. 1995].

1.1 The Design Space

The process of exploring the design space can be viewed as solving either the time-constrained scheduling (TCS) problem (minimizing the functional unit area) for a range of time constraints, or the resource-constrained scheduling (RCS) problem (minimizing the latency) for a range of resource constraints. Although there is a tradeoff between latency and area, the tradeoff curve is not smooth due to the finite combinations of the library modules available.

Consider the design space shown in Figure 1 – this curve can be described by the set of points $\{(T, f(T))\}$, where $f(T)$ is the minimum area required for a given time constraint T (i.e., the optimal solution to that TCS problem). To ensure that this curve is completely characterized, one could exhaustively solve the TCS problem optimally for every time constraint T from $T_{\min}$ (the critical path length) to $T_{\max}$ (the time constraint corresponding to the module selection / allocation with the minimum area). However, this brute-force approach is not very efficient.

Fortunately, the number of points needed to fully characterize the optimal tradeoff curve is much smaller. The curve can be completely characterized by the set $\{(T^*, f(T^*))\}$ of *optimal tradeoff points* (shown by black dots in Figure 1) – those points for which there is no design with a smaller latency and the same area, and no design with a smaller area and the same latency. Such points are called *Pareto points* [De Micheli 1994] [Brayton and Spence 1984], and can be formally defined as follows:

$$f(T^*) < f(T^* - k), 1 \leq k \leq T^* - T_{\min}$$

$$f(T^*) \geq f(T^* + k), 1 \leq k \leq T_{\max} - T^*$$

Therefore, the design space exploration problem can be solved more efficiently by finding only the Pareto points in the design space. Furthermore, many optimal tradeoff curves contain two distinct clusters of such Pareto points, as shown by the shaded regions in Figure 1: one where the latency is small and the area is large, and another where the area is small and the latency is large.

This paper explores several approaches to find the Pareto points in an efficient manner. First, Section 2 describes a basic methodology that explores the latency axis to find the Pareto points in the design space through repeatedly applying a TCS methodology. Section 3 discusses how to extend this problem to incorporate the clock length determination problem, and Section 4 discusses the incorporation of module selection. Section 5 presents improvements that improve the run-time efficiency of the basic methodology. Lastly, Section 6 gives a summary of this work and suggests some future directions that it may take.

2. LATENCY-AXIS EXPLORATION

One approach to finding the Pareto points is to repeatedly solve the TCS problem[1], leading to two subproblems: (1) determining which time constraints to explore, and (2) determining how to efficiently explore the design space at each time constraint.

Ideally, we want to avoid exhaustively searching all time constraints in the feasible range $[T_{\min}, T_{\max}]$. If the module set and clock length are specified *a priori*, then the TCS problem need only be solved for those time constraints that are a multiple of the clock length, since any other time constraint could be replaced by the smaller of the two time constraints that it would lie between without any increase in area.

As a simple example, consider the design space exploration problem for the DIFFEQ example [Paulin and Knight 1989], using library A from Table I (the "trivial" library 1 found in [Timmer et al. 1993]) and a clock length of 100. The minimum time constraint is 600 (the length of the critical path), the maximum time constraint is 1300 (the latency required for a feasible schedule with 1 mult and 1 alu1), so the only time constraints that must be explored are those in the set $\{600, 700, 800, 900, 1000, 1100, 1200, 1300\}$.

Given that set of time constraints, our Voyager design space exploration system [Chaudhuri and Walker 1996] can be used to efficiently characterize the design space as follows. The main loop (see Figure 2) scans the time constraints in the direction of increasing latency. At each time constraint, an ASAP schedule is first calculated to determine if a feasible schedule exists for that time constraint and clock length. If so, then it uses a polynomial time algorithm to compute a lower bound on the functional unit (FU) area; if this area is the same as or larger[2] than the previous area, then that solution is not a Pareto point. This is the case for time constraints 900, 1000, 1100, and 1200 in Figure 3.

However, if the lower bound is smaller than the previous area, then it is a potential Pareto point. The methodology then computes an upper bound on the area, and

---

[1] Note that, although we are solving only the TCS problem, this methodology is not limited to solving only that problem, and could be extended to include register allocation, interconnect allocation, control unit design, etc.

[2] In the problem as specified so far, the area will never be larger. However, it may be larger when the clock length determination and module selection problems are incorporated into the methodology as described in Sections 3 and 4.

```
Design Space Exploration:

$area_{cur} \leftarrow$ MAXINT
compute $T_{\min}$ and $T_{\max}$
compute all candidate time constraints $T_i$ in $[T_{\min}, T_{\max}]$
for each $T_i$ from $T_{\min}$ to $T_{\max}$
  if (latency(ASAP) $> f(T_i)$)
    $(T_i, f(T_i))$ is not a feasible schedule
  else
    compute the lower bound $lb$ on $f(T_i)$
    if ($lb >= area_{cur}$)
      $(T_i, f(T_i))$ is not a Pareto point          /* nP-lb */
    else
      compute the upper bound $ub$ on $f(T_i)$
      if ($lb = ub$)
        $(T_i, f(T_i))$ is a Pareto point            /* P-lbub */
        $area_{cur} = lb$
      else
        compute the LP-relaxed lower bound $rlb$ on $f(T_i)$
        if ($rlb >= area_{cur}$)
          $(T_i, f(T_i))$ is not a Pareto point      /* nP-rlb */
        else if ($rlb = ub$)
          $(T_i, f(T_i))$ is a Pareto point          /* P-rlbub */
          $area_{cur} = rlb$
        else
          calculate IP solution $ip = f(T_i)$.
          if ($ip < area_{cur}$)
            $(T_i, f(T_i))$ is a Pareto point        /* P-ILP */
            $area_{cur} = ip$
          else
            $(T_i, f(T_i))$ is not a Pareto point/* nP-ILP */
```

Fig. 2. Voyager's main design space exploration loop

MODULE	AREA	DELAY (ns)	OPERATIONS
mult	1440	200	$\{*\}$
alu1	160	100	$\{+, -, <\}$

Table I. Library A – Timmer's "trivial" library 1

compares it to the lower bound. If the two are equal, then the point is an optimal solution, and a Pareto point (e.g., time constraint 800 in Figure 3); if not, then the results are still inconclusive (e.g., time constraint 700). It then uses a tighter (but more computationally-intensive) FU lower-bounding method based on LP-relaxation, and tries this procedure again (in this example, determining that time

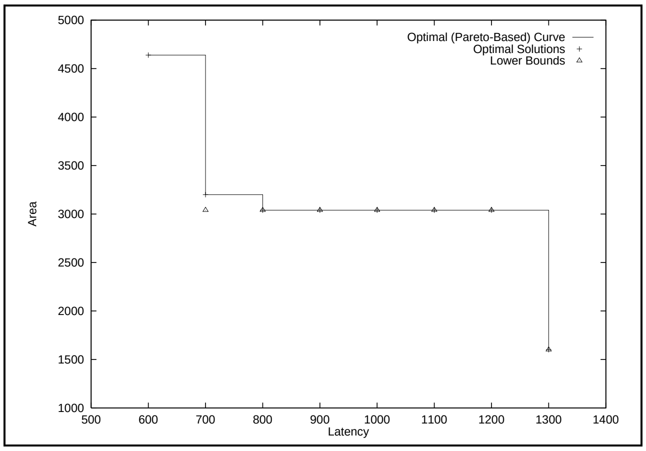

Fig. 3. Results from DIFFEQ using library A

constraint 700 corresponds to a Pareto point). If this method also fails, then it solves a carefully-developed ILP formulation [Chaudhuri et al. 1994] to determine the optimal solution. In this manner, our base methodology quickly determines whether or not each time constraint corresponds to a Pareto point by carefully pruning the search space.

3. ADDING CLOCK DETERMINATION

As described so far, the problem is simplified by assuming the clock length was known *a priori*, whereas recent work has shown that not only is determining the system clock length a difficult problem [Chen and Jeng 1991; Chaiyakul et al. 1992; Narayan and Gajski 1992; Corazao et al. 1993; Jha et al. 1995; Chaudhuri and Walker 1996]), but the choice of the clock length has a significant impact on the resulting design. Therefore, the problem of clock length determination must be folded into the design space exploration problem.

3.1 Prior Work

As described in [Chaudhuri and Walker 1996], the clock determination problem is usually ignored in favor of *ad hoc* decisions or estimates. For example, several early synthesis systems used the delay of the slowest functional unit as the estimated clock length, a choice which favored the use of chaining and disallowed multi-cycling. A heuristic method for finding the clock length was given in [Narayan and Gajski 1992], but the result may not be optimal.

To guarantee that the optimal clock length is chosen,[3] the scheduling problem could be solved repeatedly for every possible clock length – a very computationally-intensive task. Fortunately, such an exhaustive search is not necessary, as the set of candidate clock lengths to be scheduled can be reduced. In [Corazao et al. 1993], one method for reducing that set is given. A tighter method was introduced in [Chen

---

[3] Actually, this is only the data path component of the system clock length; the final clock length includes controller and interconnect delays as well.

MODULE	AREA	DELAY (ns)	OPERATIONS
mul1	150	163	$\{*\}$
alu1	100	48	$\{+, -, <\}$

Table II. Library B – Narayan's library

Clock Length	slack(*)	slack(+)	replaced by
163	0	115	–
82	1	38	–
55	2	7	–
48	29	0	24
41	1	38	82
33	2	18	55
28	5	8	24/55
24	5	0	–
21	5	15	24/55
19	8	9	24/55
17	7	3	24

Table III. Slack values found in library B

and Jeng 1991], and later proven correct in [Jha et al. 1995] and [Chaudhuri and Walker 1996] – this method computes a small set of candidate clock lengths (one of which must be the optimal clock length) by taking the ceiling of the integral divisors of each of the functional unit delays.

3.2 Pruning the Candidate Clock Lengths

Even integral-divisor methods can lead to a set of candidate clock lengths so large that it becomes too time-consuming to solve the TCS problem for each clock length at each time constraint. Fortunately, the set of candidate clock lengths can be reduced even further, as described below.

Definition 1. *For a given clock length c, the slack s_k of a module of type k with execution delay d_k is given by*

$$s_k(c) = c \cdot \left\lceil \frac{d_k}{c} \right\rceil - d_k,$$

Theorem 1. *Given a clock length c, if there exists a clock length c^* such that $s_k(c^*) \leq s_k(c)$ for all module types k in the current module selection, then c can be replaced by c^* without lengthening the schedule.*

Proof: see [Blythe and Walker 1996].□

To demonstrate the use of this theorem, consider library B, shown in Table II (the VDP100 library from [Narayan and Gajski 1992], augmented with areas similar to those of library A). Assuming a technology limit of 17ns on the shortest clock length, integral divisor methods give the set $CK = \{163, 82, 55, 48, 41, 33, 28, 24, 21, 19, 17\}$ of candidate clock lengths, with the corresponding slack values shown in Table III.

When Theorem 1 is applied to this set of candidate clock lengths, the set is reduced to $CK' = \{163, 82, 55, 24\}$. Note that when two sets of slack values are equivalent, the shorter clock length is dropped since it would tend to result in a larger controller.

```
Design Space Exploration w/ Clock Determination:

$area_{cur} \leftarrow$ MAXINT
compute pruned set $CK'$ of candidate clock lengths
compute $T_{\min}$ and $T_{\max}$
for each $c_j$ in $CK'$
    compute all candidate time constraints $T_i$ in $[T_{\min}, T_{\max}]$
for each $T_i$ from $T_{\min}$ to $T_{\max}$
    using each $c_j$ in $CK'$ inducing $T_i$
        determine if $(T_i, f(T_i))$ is a Pareto point (see Figure 2)
```

Fig. 4. Voyager's design space exploration loop with clock determination

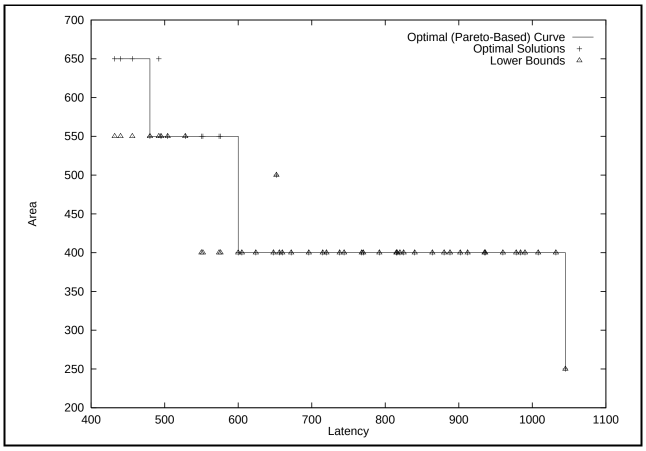

Fig. 5. Results from DIFFEQ using library B

3.3 Exploring the Candidate Clock Lengths

Once the pruned set CK' of candidate clock lengths has been computed, the integral multiples of each of those clock lengths give the time constraints to explore. Then, for each such time constraint and candidate clock length, the methodology outlined in Figure 2 can be applied. The efficiency of the search at each time constraint can be improved by observing that each time constraint was derived as an integral multiple of one or more clock lengths, so only those inducing clock lengths need be explored at that time constraint. The resulting methodology is outlined in Figure 4.

Using library B and the DIFFEQ example, this methodology generates the design space shown in Figure 5. From the pruned set $CK' = \{163, 82, 55, 24\}$ of candidate clock lengths, 49 time constraints were generated, and 50 time constraint / clock length pairs (i.e. design points) were explored (note that there was only a single time constraint with more than one candidate clock length). Two corresponded to infeasible schedules, while the other 48 had to be examined to determine if they were Pareto points. As Table IV shows (the headings of the last six columns correspond

clocks	time constr.	design points	nP-lb	nP-rlb	nP-ILP	P-lb	P -rlb	P-ILP
4	49	50	38	6	0	2	2	0

Table IV. Statistics from solving DIFFEQ using library B

to labels in Figure 2), the vast majority of the solutions were determined to be either Pareto or non-Pareto points using the bounding heuristics – only two were solved using the tighter LP-relaxation lower bounding method and no solutions required an ILP approach!

Note also that in several cases (time constraints in the range 420-600), the lower bound differed from the optimal solution, so methods based solely on lower-bounding would incorrectly characterize the design space.

Finally, Figure 5 also demonstrates the importance of systematically examining all relevant clock lengths in the design space. At a time constraint of 652, the inducing clock length of 163ns leads to a solution with an area of 500, whereas the previous time constraint had a lower area of 400. Although the point (652, 500) is optimal with respect to its time constraint and a fixed clock length of 163 ns, it is not a Pareto point, and is thus rejected by the line labeled */* nP-lb */* in Figure 2.

4. ADDING MODULE SELECTION

While adding clock length determination to the base methodology is an important step toward supporting more complex libraries, the methodology must also be extended to cover libraries that offer a number of possible module sets. Again, we would prefer to avoid an exhaustive search of all possible module sets, yet we must ensure that we do not miss a time constraint, clock length, and module set combination (i.e a design point) that corresponds to a Pareto point.

4.1 Prior Work

Over the years, a variety of methods have been employed to determine the appropriate module set. One method, described in [Jain et al. 1988], generates a number of module sets, and then selects the best one. Another method, presented in [Timmer et al. 1993], computes an initial module set through a MILP formulation, and determines its validity by scheduling; if no viable schedule is found, then the set (and its allocation) are updated, and the scheduling process is repeated.[4] As with the previous work on clock length determination, using such techniques to determine a single module set before (and independently of) scheduling cannot guarantee a globally optimal solution. Instead of trying to find a single module set, the method found in [Chen and Jeng 1991] exhaustively explores all possible module sets. Since this method also exhaustively explores all integral divisor based clock lengths, its computational complexity is too large for optimal scheduling, so only estimates are computed.

---

[4] This method also incorporates the type mapping problem into the MILP formulation – something our methodology does not yet handle.

MODULE	AREA	DELAY (ns)	OPERATIONS
alu	100	125	$\{*, +, -, <, >\}$
mul	80	100	{*}
add	50	50	{+}
sub	60	60	{-}
cmp	65	60	{<,>}

Table V. Library D – an artificial complex library

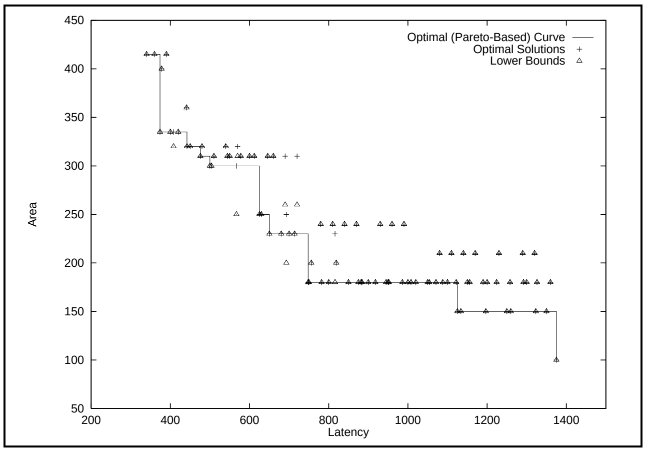

Fig. 6. Results from DIFFEQ using library D

4.2 Exploring Different Module Sets

Fortunately, such an exhaustive search is not necessary. The number of module sets that must be explored at each time constraint can be reduced (as was the number of candidate clock lengths) by observing that each time constraint was derived as an integral multiple of a clock length derived from one or more specific modules. Therefore, only those module sets that contain at least one of those modules must be explored at that time constraint.

As an example, consider library D, shown in Table V (an artificial library slightly less complex than library C, but with more realistic module delays). Using that library, and the DIFFEQ example, the methodology described above generates the design space shown in Figure 6. Here there were 16 possible module sets, 9 integral-divisor candidate clock lengths, and 131 time constraints – almost 19,000 combinations. Even after pruning the candidate clock lengths, there were 7 pruned candidate clock lengths, and 93 feasible time constraints – almost 9,000 combinations.

However, the methodology had to solve only 1522 TCS problems (an average of 1.35 clock lengths and 11.27 module selections at each time constraint). 183 of those were infeasible, and again, the vast majority of the solutions were determined to be either Pareto or non-Pareto points using the bounding methods. Moreover, this entire procedure took only 1.5 hours of wall-clock time. Without such a careful pruning of the search space, this problem could not have been solved optimally in a reasonable amount of time.

library	clocks	time constr.	design points	nP-lb	nP-rlb	nP-ILP	P-lb	P-rlb	P-ILP
C	1	9	288	251	1	0	0	2	2
D	7	99	1522	1325	1	3	9	0	1

Table VI. Statistics from solving DIFFEQ using libraries C and D

Furthermore, with these 4 module delays, there are many resulting designs that lie above the optimal tradeoff curve. Although these designs are optimal solutions for a particular clock length and module set, they are not Pareto points, so it is very important that the methodology correctly and completely explores the design space.

5. IMPROVING METHODOLOGY EFFICIENCY

Thus far, it has been shown that carefull pruning of the design space along the latency axis makes design space exploration a tractable problem. However, it is possible to further increase the effieciency of design space exploration by considering the area axis in a similar manner. Again, it is necessary to reduce the number of constraints to explore, as searching the entire integral range along the area axis would be prohibitively expensive. Given such a reduced set of area constraints, the algorithm outlined in Figure 2 can be modified to explore the area axis instead – starting with the point that has the smallest possible area (and the largest latency) and continuing until it reaches the point with the largest possible area (and minimal latency).

In much the same way that any one time constraint can be induced by more than one clock length, each area constraint can correspond to more than one module set / resource allocation. Enumerating all possible allocations whose resulting area is less than $A_{\max}$ for each module set gives an initial set of candidate area / resource constraints that can be used in the area-axis methodology. The size of this initial set can then be reduced by noting that it may contain overly loose resource constraints – for example, a resource constraint of 3 adders would be too loose for a behavioral description with only one addition operation. In general, we can reduce the set of candidate resource constraints by upper bounding the number of independent paths in the DFG that could require resources of type T and could possibly be executed in parallel. The resulting upper bound would then be the maximal number of type T resources needed in any allocation for *any* schedule using *any* clock length. Although greatly dependent on the module library and DFG in use, applying even such simple heuristics usually reduces the number of area constraints to explore by 80% - 85%.

Results for both the latency-axis and area-axis methodologies are given in Table VII, which shows results for three different benchmarks (DIFFEQ, AR-lattice, and Elliptic Wave Filter). In each cell of the table, the execution time[5] is given (as *minutes*:*seconds*), along with the total number of points explored (i.e., the number of TCS or RCS problems solved).

---

[5] The executions times are based on a Sun SPARC-20 running the Solaris Operating System.

	Latency Axis	Area Axis
DIFF	14:07 94	2:36 72
AR	48:08 198	8:50 49
EWF	55:45 230	223:22 42

Table VII. Results from axis-based exploration using the library from Table V

Note that neither approach is universally better than the other. Most of the time, it was faster to use area-axis exploration, but for the EWF example, several of the RCS problems were quite time-consuming to solve optimally. However, expressing those problems as TCS problems tended to be significantly faster, resulting in faster latency-axis exploration. Furthermore, the simplicity of the module library used tends to lend itself to giving fewer area constraints, which is a significant factor in the speed of either method. Thus it is unclear how to determine, *a priori*, which axis to choose for exploration. In order to increase the efficiency of a design space exploration methodology, it would be advisable to utilize both axes when performing design space exploration. One such methodlogy is described in [Timmer et al. 1993], where exploration is performed by alternating between axes to find the Pareto points. However, this methodology fails to incorporate the clock length determination problem. When this subproblem is included, the methodology's efficiency is frequently worse than either of the axis based methods outlined here.

5.1 Pivoting Between Latency-Axis and Area-Axis Exploration

Another approach to combining latency-axis and area-axis exploration is to consider the structure of the tradeoff curve. As shown in Figure 1, a large number of the Pareto points are clustered into two regions: one where the latency is small and the area is large, and another where the area is small and the latency is large. This phenomenon is also illustrated in Figure 7. Here, the latency-axis methodology (exploring the latency axis in the direction of increasing latency) would find many Pareto points fairly quickly, but would then waste a considerable amount of time exploring time constraints that do not correspond to Pareto points until the high-latency cluster of Pareto points is reached. Using the area-axis methodology (exploring the area axis in the direction of increasing area) has a similar shortcoming.

This shortcoming can be overcome by *pivoting* between the two axis-based methods – using the latency-axis methodology to explore the high-area / low-latency cluster, and using the area-axis methodology to explore the high-latency / low-area cluster. When exploring the latency axis in the direction of decreasing latency, the most obvious method of pivoting is to simply switch from latency-axis exploration to area-axis exploration after exploring a certain percentage of the latency axis. Note that after making this switch, the area-axis methodology must still explore the area axis in the direction of increasing area (so that information from previous schedules can be used to prune the search space as described in Section 2), but

	% time constraints explored						
	100	50	33	20	10	5	0
DIFF	14:07 94	3:58 52	2:24 42	2:01 45	1:38 44	1:49 44	2:46 72
AR	48:08 198	11:04 102	7:45 75	5:57 48	4:41 37	9:24 57	8:50 49
EWF	55:45 230	7:01 116	3:01 79	1:29 48	221:46 26	209:51 24	223:22 42

Table VIII. Results from simple pivoting

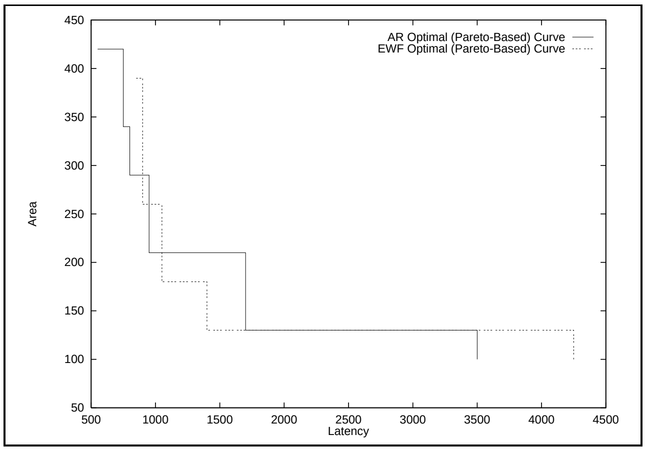

Fig. 7. The EWF and AR optimal Pareto-base curves when using the library from Table V

now it can stop when it reaches the last Pareto point found by the latency-axis methodology.

The results of performing this pivoting process for various percentages of the latency axis are presented in Table VIII. Note that the 0% column corresponds to an immediate pivot to area-axis exploration, and the 100% column corresponds to using solely latency-axis exploration. Not surprisingly, for the tradeoff curves depicted in Figure 7, the percentages that result in the fastest execution times are fairly low (10%-20%), since most of the low-latency cluster of Pareto points are within the first 20% of the latency axis. Unfortunately, however, there is no consistent percentage that will always correspond to the best pivot point for every tradeoff curve, regardless of whether the execution time [6] or the number of points being explored is the quantity being minimized.

Since the best pivot point cannot be determined *a priori*, it should be determined dynamically during the exploration process. As the tradeoff curve often exhibits two clusters of Pareto points as described earlier, one approach would be to determine when a cluster is being left, and pivot while exploring the next few points that are

---

[6] Once again, the EWF example contains several points that are computationally more expensive when solved as RCS problems – thus the dramatic execution time increase between 20% and 10% despite the decrease in points explored.

	% time constraints in window					
	0(a)	5	10	15	20	100(t)
DIFF	2:36	1:48	2:23	4:40	5:20	14:07
	72	44	49	55	59	94
AR	8:50	10:08	6:20	7:34	8:21	48:08
	49	55	53	63	73	198
EWF	223:22	256:05	256:15	2:50	3:50	55:45
	42	27	39	74	86	230

Table IX. Results from dynamic window-based pivoting

not members of either cluster. When exploring the latency axis, this pivot would occur when the curve begins to "flatten out" into a roughly horizontal line.

One simple method of implementing this dynamic pivot is a constant sized window of design points. This window contains the last n design points explored, many of which were pruned as non-Pareto points. If the area of the first point in the window is not significantly larger than the area of the last point (which corresponds to the current time constraint, the current point is selected as the pivot point. In other words, if a Pareto point has not been found recently, the curve is flattening out and the pivot from latency-axis exploration to area-axis exploration is made.

The results of applying this dynamic window-based pivoting are given in Table IX. To determine when a "significant" change in area was reached, the size of the current window was compared to the change in area over that window. If the percentage change in area was smaller than the size of the window as a percentage of the total number of time constraints, we pivoted to using the area axis; otherwise we continued using the latency axis. In general, a window size of 10%-15% of the time constraints seemed to give good results, as shown in Table IX. The first example shown (DIFFEQ) is small enough that 5% of the time constraints is statistically insignificant, leading to results that are dominated by the area-axis exploration. However, the EWF results give a strong argument for using dynamic pivoting – here a bad *a priori* choice of using only latency-axis exploration or area-axis exploration could lead to a significantly larger execution time.

6. CONCLUSIONS

This paper has examined the process of design space exploration, reducing that process to one of characterizing the optimal latency-area tradeoff curve by finding all the Pareto points on that curve. For the combined problem of scheduling, clock length determination, and module selection, we have presented two exploration methodologies: dedicated latency-axis or area-axis exploration and pivoting between the two axis-based methods. Each of these methodologies takes advantage of the structure found along both the latency and are axes by carefully pruning a large number of sub-optimal solutions at each level of the design cycle, making it possible to use optimal scheduling techniques rather than bounds or estimates. Furthermore, we discussed how the tradeoff curve is dominated by two clusters of Pareto points, and how that structure can be used to more efficiently find the Pareto points.

References

BLYTHE, S. A. AND WALKER, R. A. 1996. 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, Nov. 6-8 1996). IEEE Computer Society Press, 8–13.

BRAYTON, R. K. AND SPENCE, R. 1984. *Sensitivity and Optimization.* Computer-aided design of electronic circuits. Elsevier Science Publishing Co., INC., 52 Vandervilt Avenue, New York, NY 10017, USA.

CHAIYAKUL, V., WU, A. C.-H., AND GAJSKI, D. D. 1992. Timing Models for High Level Synthesis. In *Proc. of the European Design Automation Conference (EuroDAC)* (Hamburg, Germany, Feb. 1992). IEEE Computer Society Press, 60–65.

CHAUDHURI, S. AND WALKER, R. A. 1996. Computing Lower Bounds on Functional Units before Scheduling. *IEEE Transactions on VLSI Systems 4*, 2 (June), 273–279.

CHAUDHURI, S., WALKER, R. A., AND MITCHELL, J. E. 1994. Analyzing and Exploiting the Structure of the Constraints in the ILP Approach to the Scheduling Problem. *IEEE Transactions on VLSI Systems 2*, 4 (Dec.), 456–471.

CHEN, L.-G. AND JENG, L.-G. 1991. Optimal Module Set and Clock Cycle Selection for DSP Synthesis. In *Proc. of 1991 IEEE International Symp. on Circuits and Systems.* (Singapore, June 11-14 1991). IEEE Computer Society Press, 2200–2203.

CORAZAO, M., KHALAF, M., GUERRA, L., POTKONJAK, M., AND RABAEY, J. M. 1993. Instruction Set Mapping for Performance Optimization . In *Proc. of the IEEE/ACM International Conference on Computer-Aided Design* (Santa Clara, California, Nov. 7-11 1993). IEEE Computer Society Press, 518–521.

DE MICHELI, G. 1994. *Synthesis and Optimization of Digital Circuits.* McGraw-Hill series in electrical and computer engineering. McGraw-Hill, New York, NY, USA.

GAJSKI, D. D., VAHID, F., NARAYAN, S., AND GONG, J. 1994. *Specification and Design of Embedded Systems.* P T R Prentice-Hall, Englewood Cliffs, NJ 07632, USA.

JAIN, R., PARKER, A. C., AND PARK, N. 1988. Module Selection for Pipeline Synthesis. In *Proc. of the 25th ACM/IEEE Design Automation Conf.* (Anaheim, California, June 12-15 1988). IEEE Computer Society Press, 542–547.

JHA, P., PARAMESWARAN, S., AND DUTT, N. 1995. Reclocking for High Level Synthesis. In *Proc. of the Asia-South Pacific Conference on Design Automation (ASP-DAC)* (Makuhari Messe, Chiba, Japan, Aug. 29-Sept. 1 1995). IEEE Computer Society Press.

NARAYAN, S. AND GAJSKI, D. D. 1992. System Clock Estimation based on Clock Slack Minimization. In *Proc. of the European Design Automation Conference (EuroDAC)* (Hamburg, Germany, Feb. 1992). IEEE Computer Society Press, 66–71.

OHM, S. Y., KURDAHI, F. J., DUTT, N., AND XU, M. 1995. A comprehensive Estimation Technique for High-Level Synthesis. In *Proc. of the 8th International Symposium on System-Level Synthesis* (Cannes, France, Sept. 13-15 1995). IEEE Computer Society Press.

PAULIN, P. G. AND KNIGHT, J. P. 1989. Force Directed Scheduling for the Behavioral Synthesis of ASICs. *IEEE Transactions on Computer-Aided Design 8*, 6 (June), 661–679.

TIMMER, A. H., HEIJILIGERS, M. J. M., AND JESS, J. A. G. 1993. Fast System-Level Area-Delay Curve Prediction. In *Proc. of 1st APCHDLSA* (1993). 198–207.

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