*Efficiently Searching the Optimal Design Space**

Stephen A. Blythe
Saint Louis University
Computer Science Department
St. Louis, MO 63156

Robert A. Walker
Kent State University
Mathematics and Computer Science Dept.
Kent, OH 44242

Abstract

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.*

1 Introduction

One of the primary advantages of a high-level synthesis system is its ability to explore the design space in an attempt to find the optimal tradeoff curve between area (ideally total area, but often only functional unit area) and time (the schedule length, or latency). This process of design space exploration can be viewed as solving either the time-constrained scheduling (TCS) problem (minimizing the functional unit area) for some range of time constraints, or the resource-constrained scheduling (RCS) problem (minimizing the latency) for some range of resource constraints.

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.

Alternatively, one could compute either lower bounds [1] or estimates [2] on the optimal tradeoff curve for some set of time or resource constraints. However, it would be preferable to compute the *optimal* tradeoff curve, provided that such a derivation can be made in an efficient manner.

Fortunately, the optimal tradeoff curve can be completely characterized by the set $\{(T^*, f(T^*))\}$ of *Pareto points* [3, 4] (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. More intuitively, these points are represented by all the knees found in the tradeoff curve.

*Portions of this work were supported by the National Science Foundation under Grants MIP-9423953 and MIP-9796085.

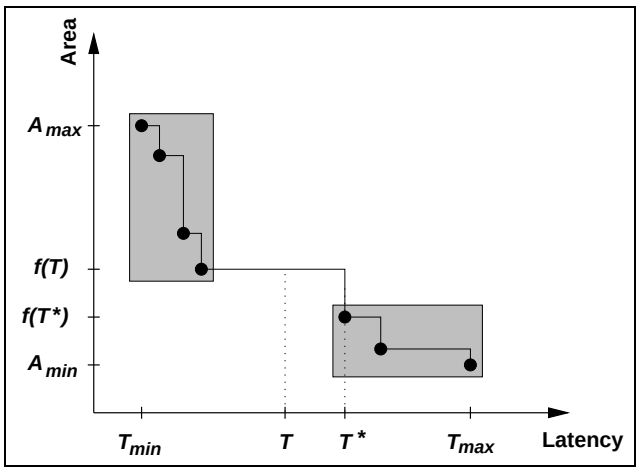

Figure 1: Example design space showing Pareto points. The shaded regions show the two distinct clusters of Pareto points that many tradeoff curves exhibit.

A truly effective design space exploration methodology must incorporate as many design parameters as possible so as to reflect a realistic tradeoff curve. It must support complex module libraries, and must consider controller effects, register and interconnect area, etc. We have concentrated on *extending our prior work to incorporate module selection as well as clock length determination*, and on *efficiently finding the optimal tradeoff curve for this combined problem.* Note that this is far more complex than simply solving a single scheduling problem – it requires finding the optimal solutions to thousands of permutations of this problem.

2 Axis-Based Exploration

To find the Pareto points for this combined problem of scheduling, clock length determination, and module selection, the most obvious approach is to simply scan either the latency or area axis; that is, either solve the TCS problem repeatedly for various time (latency) constraints or solve the RCS problem repeatedly for various resource (functional unit area) constraints. However, since this may require solving thousands of scheduling problems, it is important both (1) to use the structure of the problem to reduce the number of constraints to explore, as well as (2) to efficiently explore the design space at each constraint. In this section, we will concentrate on the former point; for a discussion of using polynomial time bounding and exact techniques in conjunction for efficient exploration, see [5].

1999 Great Lakes Symposium on VLSI, pp. 192-195

2.1 Latency-Axis Exploration

In [5], we described a latency-axis exploration methodology for finding all the Pareto points by determining a *small* set of time constraints to explore, and solving the TCS problem at each time constraint. This set is explored in increasing order, with each time constraint being analyzed to determine if it corresponds to a Pareto point. This analysis is performed in several stages. First, a lower bound on the area is computed using efficient bounding techniques [6]. If the lower bound is above the current curve, it can not be a Pareto point. If it is below the current curve, it may be a Pareto point, so an upper bound on the area is calculated. If this upper bound is equal to the lower bound, those bounds correspond to the optimal schedule, and that schedule is indeed a Pareto point. If not, a tighter lower bound method is computed using an LP-relaxation [7, 8], and this process is repeated. If the results are still not conclusive, then an exact ILP-based scheduler is used [9].

2.2 Area-Axis Exploration

Viewing the previous approach as a latency-axis exploration methodology, an alternative approach is based on the area axis: the Pareto points can be found by determining a set of area constraints to explore, and then optimally solving the RCS problem at each area constraint. It is also necessary to reduce the number of constraints to explore, as searching the entire integral range along the area axis would be prohibitively expensive.

2.3 Latency-Axis vs. Area-Axis Exploration

In practice, neither approach is universally better than the other. Experiments have shown that most of the time, it was faster to use area-axis exploration, but for some examples, 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. Thus it is unclear how to determine, *a priori*, which axis to choose for exploration.

3 A Timmer-Like Exploration

Since neither axis-based method is universally effective, it might be desirable to combine them in some way to further increase the efficiency of the exploration methodology. One approach would be to combine them using a search methodology similar to the one employed by Timmer in [1]. However, our studies have shown that this method is not efficient for the combined problem of scheduling, clock length determination, and module selection. Moreover, our studies have also shown that Timmer's methodology may fail to correctly find the optimal tradeoff curve for this problem. This pitfall is briefly outlined in Figure 2, where Timmer's method has (correctly) found the rightmost Pareto point through an application of RCS. Since TCS is now performed at this new time constraint, the result may miss the candidate clock length and module selection combination that corresponds to the next true

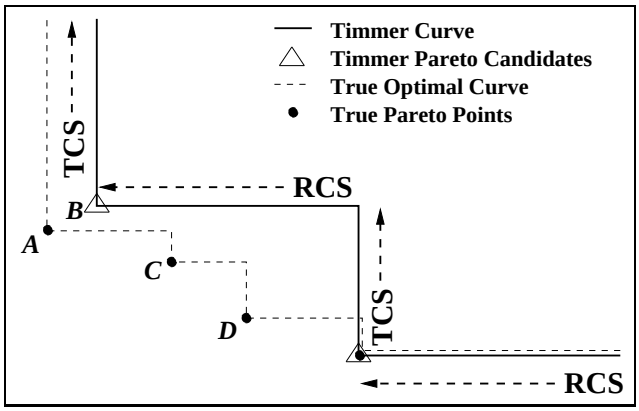

Figure 2: The pitfall of Timmer's method when considering clock length and module set determination

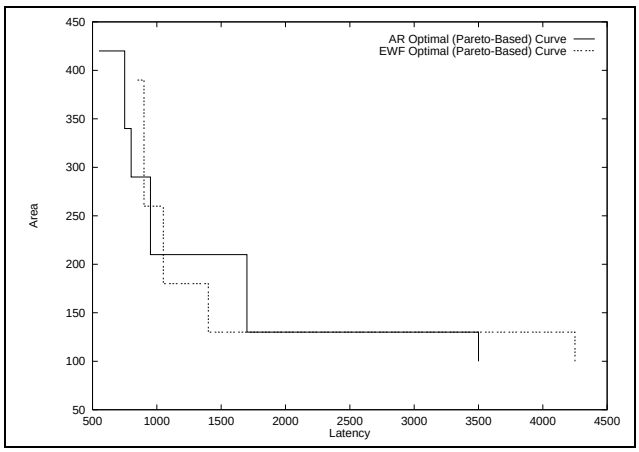

Figure 3: The EWF and AR optimal Pareto-base curves when using the library from Table 1

Pareto point (D) to the left, which our methodology would correctly and *efficiently* find.

4 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 3. 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.

However, this shortcoming can be overcome by *pivoting* between the two axis-based methods – using the latency-axis methodology to explore the high-area /

Module	Area	Delay	Operations
alu	100	125	$\{*, +, -, <, >\}$
mul	80	100	$\{*\}$
add	50	50	$\{+\}$
sub	60	60	{-}
cmp	65	60	$\{<, >\}$

Table 1: An artificial complex library

	% 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 2: Results from simple pivoting

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 [5]), but 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 2. 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 3, 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 [1] or the number of points being explored is the quantity being minimized. Thus, a better method for deciding where to pivot must be found.

5 *Dynamic Pivoting*

Since the best pivot point cannot be determined *a priori*, it must be determined dynamically during the exploration process. Since 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 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 would be to consider a window of constant size that contains the last n design points explored (although many of these would not be Pareto points). Then, if the area corresponding to the first element in the window (i.e., the shortest time constraint) is not significantly larger than the area of 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, pivot from latency-axis exploration to area-axis exploration.

The results of applying this dynamic window-based pivoting are given in Table 3. To determine when a "insignificant" 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. Unfortunately, there was no consistent window size that yielded the best result in all cases. However, a window size of 10%-15% of the time constraints generally seemed to give good results.

Looking at Table 3, 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 (as shown in Table 2) could lead to a significantly larger execution time than 15% dynamic pivoting.

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

Table 3: Results from dynamic window-based pivoting

6 *Further Results*

In all of our results so far, we have used the library presented in Table 1. That library has a number of different delays, which complicates any design space exploration methodology that considers clock length determination. However, it has only two alternatives for each operation type, leading to a fairly small number of module selection candidates.

Now consider Table 4, which is the opposite: it has fewer unique functional unit delays, and several of those delays are multiples of each other. Both of these factors result in fewer resulting candidate clock lengths (for example, several functional units have 50 as a candidate clock length). However, this library has a much larger number of module selection candidates.

[1] 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.

Module	Area	Delay	Operations
mul1	500	200	{*}
mul2	600	150	{*}
mul3	800	100	{*}
sub1	100	160	{-,<}
sub2	200	110	{-,<}
sub3	400	60	{-,<}
add1	90	150	{+}
add2	185	100	{+}
add2	380	50	{+}

Table 4: A module selection intensive library

	Latency	Area	Timmer	Pivot (15%)
DIFF	44:22 89	4:37 359	32:58 158	6:15 176
AR	369:32 59	213:17 224	313:27 69	192:17 60
EWF	149:59 63	232:01 259	298:46 52	43:13 31

Table 5: Results when using a library with many module selections

Results using this library are presented in Table 5. Compared to results using the previous library when the latency-axis methodology is used, there is significantly more time being spent exploring the latency axis, since the module selection problem is now much more difficult and latency-axis exploration gets most of its time savings due to the structure of the clock length determination problem. However, for area-axis exploration the results are now generally faster, reflecting the savings due to considering the structure of the module selection problem. Again, as with the first library, EWF gives several RCS problems that are time consuming to solve optimally (while the corresponding TCS problems are not as time consuming), thus dramatically increasing overall run time for the area axis. Note that in all cases the number of area constraints to solve is much higher – thus the savings in execution time must result from the fact that there is more structure to each constraint along the area axis for this library.

As with the prior library, Timmer-like exploration (even our neighborhood-based Timmer-like exploration) once again fails to produce faster run times for this library, although in this case the primary contributing factor is not only clock length determination but the number of possible module selection candidates at each of the generated time constraints. For AR and EWF, the pivoting method once again gave the best execution times[2], but this time it also explored fewer points than the Timmer-like method!

Finally, note that the resulting design spaces for these two benchmarks are also *much* more complex for this library, as can be seen in Figure 4. The added complexity of these plots is directly attributable to the complexity of the module selection problem – many more area constraints exist, leading to more Pareto points being derived from the corresponding resource constraints.

[2]The DIFFEQ benchmark once again gives skewed results as its size does not allow a statistically significant number of Pareto points to be incorporated within the 15% design window, thus not allowing the pivoting method to take full advantage of the structure in the resulting design space.

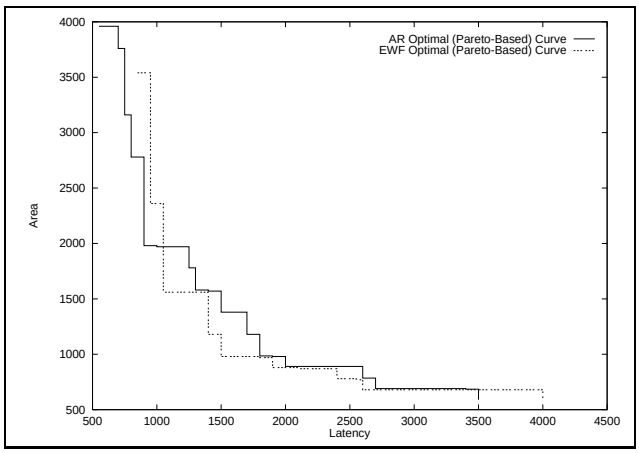

Figure 4: The EWF and AR optimal Pareto-base curves when using the library from Table 5

7 Conclusions and Future Work

We have 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 several exploration methodologies: dedicated latency-axis or area-axis exploration, a Timmer-like exploration method, and two methods (one static, one dynamic) for pivoting between the two axis-based methods. We have discussed how the tradeoff curve is dominated by two clusters of Pareto points, and how that structure, along with the structure of the combined problem, can be used to more efficiently find the Pareto points.

References

[1] A. H. Timmer, M. J. M. Heijiligers, and J. A. G. Jess, "Fast System-Level Area-Delay Curve Prediction," in *Proc. of 1st APCHDLSA*, pp. 198–207, 1993.

[2] 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.

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

[4] R. K. Brayton and R. Spence, *Sensitivity and Optimization*. Computer-aided design of electronic circuits, 52 Vandervilt Avenue, New York, NY 10017, USA: Elsevier Science Publishing Co., INC., 1984.

[5] 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), pp. 8–13, IEEE Computer Society Press, Nov. 6-8 1996.

[6] S. Chaudhuri and R. A. Walker, "Bounding Algorithms for Design Space Exploration," in *Proc. of the 9th Great Lakes Symposium on VLSI*, (Ann Arbor, Michigan), pp. ??–??, Mar. 4-6 1999.

[7] S. Chaudhuri, S. A. Blythe, and R. A. Walker, "A Solution Methodology for Exact Design Space Exploration in a Three Dimensional Design Space," *IEEE Transactions on VLSI Systems*, vol. 5, pp. 69–81, Mar. 1997.

[8] 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.

[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.