Toward a Practical Methodology for Completely Characterizing the Optimal Design Space [*]

Stephen A. Blythe[†] — *Rensselaer Polytechnic Institute*, *Troy, NY 12180*

Robert A. Walker [‡] — *Kent State University*, *Kent, OH 44242*

Abstract

One of the most compelling reasons for developing high-level synthesis systems has been the desire to quickly explore the design space. Since this problem is very difficult to solve optimally, most systems compute either lower bounds or estimates on the optimal tradeoff curve. The methodology described here goes beyond most previous work in several ways: (1) it computes all optimal tradeoff points so as to completely characterize the design space, (2) it solves not only the scheduling problem, but the clock determination and module selection problems as well, and (3) it carefully prunes the search space at each level of the design cycle.

1 Introduction

For many years, one of the most compelling reasons for developing high-level synthesis systems [1, 2] 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.*

Design space exploration is generally considered too difficult to solve optimally in a reasonable amount of time, so the problem is usually limited to computing either lower bounds [3] or estimates [4] on the optimal tradeoff curve for some set of time or resource constraints. Moreover, the design space is usually determined by solving only the scheduling and functional unit allocation subproblems.

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. The resulting methodology can also be extended to include additional subproblems.

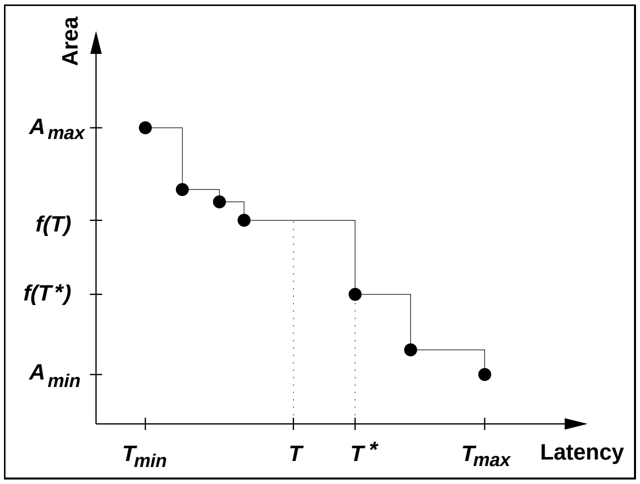

Figure 1: Example Design Space Showing Pareto Points

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 [5].

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* [2, 6], and can be

[*]This material is based upon work supported by the National Science Foundation under Grant No. MIP-9423953.

[†]Department of Computer Science.

[‡]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.

```
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 */
```

Figure 2: Voyager's main design space exploration loop

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.

2 The Base Methodology

To find all of the Pareto points, either the TCS problem could be solved repeatedly for various time constraints, or the RCS problem could be solved repeatedly for various resource constraints. Our methodology repeatedly solves the TCS problem[1], which leads 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 [7], using library A from Table 1 (Timmer's "trivial" library 1 from [3]) and a clock length of 100. The minimum time

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

Table 1: Library A – Timmer's "trivial" library 1

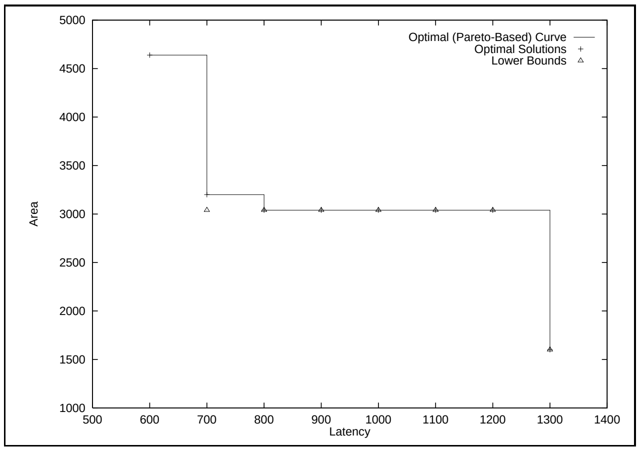

Figure 3: Results from DIFFEQ using library A

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 one 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 [8] efficiently characterizes 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 heuristic 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 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 constraint 700 corresponds to a Pareto point). If this method also fails, then it solves a carefully-developed ILP formulation [9] to determine the optimal solution, using the bounds determined earlier to reduce the search space for that solution.

Thus our base methodology quickly determines whether or not each time constraint corresponds to a

[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.

Pareto point by carefully pruning the search space. It first computes a small set of time constraints to explore. Increasingly tighter heuristics are then used to try to determine if each time constraint corresponds to a Pareto point. Only if those heuristics fail is a more computationally-intensive ILP formulation used.

Unfortunately, assuming the module set and clock length is specified *a priori* is unrealistic with complex module libraries. Accordingly, Section 3 describes how this base methodology can be extended to include clock length determination and Section 4 describes the incorporation of module selection. Finally, Section 5 presents some results using other benchmarks and describes how other problems could be incorporated into the methodology.

3 Adding Clock Determination

As described earlier, our base methodology explores a set of time constraints, determining whether or not the solution to each TCS problem is a Pareto point. The problem was 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 [4, 10, 11, 12, 13, 8]), 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 [8], 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 [11], 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 [12], Corazao *et al.* gave one method for reducing that set. A tighter method was introduced in [4], and later proven correct in [13] and [8] – 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 these 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.

[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 2: 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 3: Slack values found in library B

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*

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

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: Since $s_k(c^*) \leq s_k(c)$ for all modules k, the same quality will hold for operations in a schedule using these modules. Thus all operations in the schedule using c could be scheduled at least as soon, if not sooner, in a schedule using c^* because all operations will be capable of executing faster (or equally as fast) in the schedule using c^*. Thus, changing the clock length to c^* can only improve the schedule.□

To demonstrate the use of this theorem, consider library B, shown in Table 2 (the VDP100 library from [11], 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 3.

Consider the clock length of 33ns, found as $\lceil 163/5 \rceil = 33$. When a multiplier is scheduled using this clock length, there will be a slack of 2ns. There are several clock lengths whose slack for the multiplier is smaller, but the slack corresponding to the alu1 is always larger. However, a clock length of 55ns has the same slack as the multiplier, and less slack for the alu1. Therefore, Theorem 1 says that any schedule that uses a clock length of 33ns can be shortened by using a clock length of 55ns (without increasing the number of functional units).

When Theorem 1 is applied to the full 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

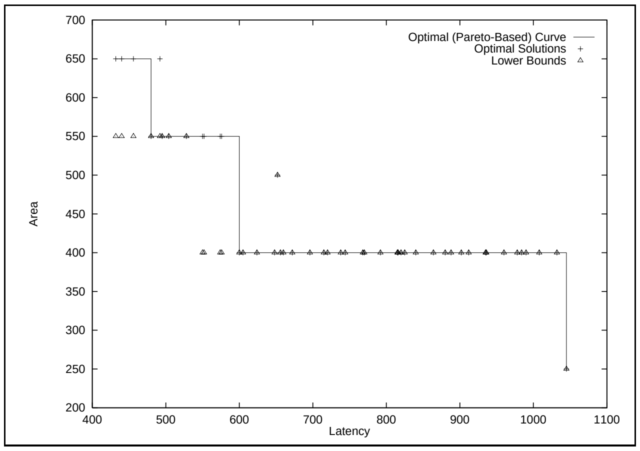

Figure 4: Results from DIFFEQ using library B

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 4: Statistics from solving DIFFEQ using library B

since it would tend to result in a larger controller.

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 5.

Using library B and the DIFFEQ example, this methodology generates the design space shown in Figure 4. 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 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 4 shows (the headings of the last six columns correspond 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 4 also demonstrates the importance of systematically examining all relevant clock lengths

```
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)
```

Figure 5: Voyager's design space exploration loop with clock determination

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 clock length, it is not a Pareto point, and will be 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 any combination of a time constraint, clock length, and module set 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 [14], generates a number of module sets, and then selects the best one. Another method, presented in [3], 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 some of 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 [4] 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.

4.2 Exploring Different Module Sets

Fortunately, such an exhaustive search is not necessary. Many of the possible module sets can be eliminated since they are incapable of implementing all the operation types found in the data flow graph. For example, in the case of the DIFFEQ, the module set must be

[4]This method also incorporates the type mapping problem into the MILP formulation – something our methodology does not yet handle. See Section 5.

MODULE	AREA	DELAY (ns)	OPERATIONS
mult	1440	200	{*}
alu1	160	100	$\{+, -, <\}$
sub1	150	100	{-}
add1	150	100	{+}
alu2	90	200	$\{+, -, <\}$
sub2	85	200	{-}
add1	85	200	{+}

Table 5: Library C – Timmer's library 2

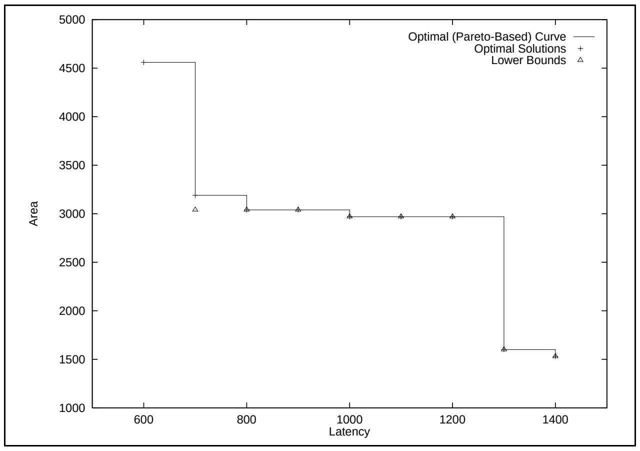

Figure 6: Results from DIFFEQ using library C

capable of performing the operations $\{+, -, *, <\}$; any module sets that do not can be eliminated.

Moreover, 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.

Using library C, shown in Table 5 (Timmer's library 2 from [3]), and the DIFFEQ example, the methodology described above generates the design space shown in Figure 6. There are 32 possible module sets, but only 1 pruned candidate clock length (100ns) and 9 time constraints, resulting in 288 TCS problems to solve. 32 resulted in infeasible schedules (i.e., no solution was possible), and as before, the vast majority of the solutions were determined to be either Pareto or non-Pareto points using the bounding heuristics.

As another example, consider library D, shown in Table 6 (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

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

Table 6: Library D – an artificial complex library

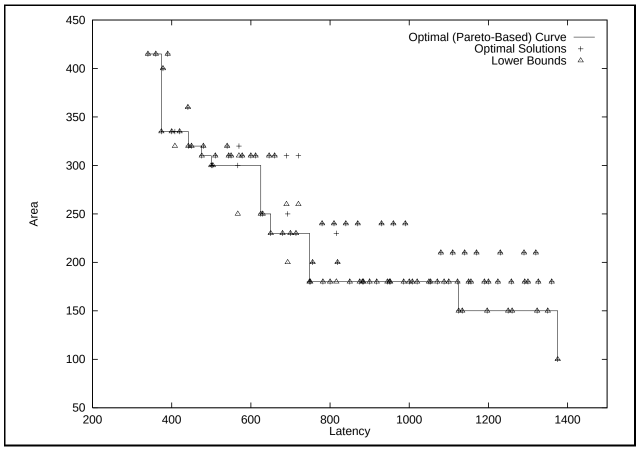

Figure 7: Results from DIFFEQ using library D

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 7: Statistics from solving DIFFEQ using libraries C and D

in Figure 7. 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 6 pruned candidate clock lengths, and 93 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 heuristics. 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.

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 explores the design space. For example, [3] presents a method that begins at time constraint $T_{\max}$ and alternately performs time and area lower-bounding to find a stair-step tradeoff curve. Even if that methodology is enhanced to alternate between optimally *solving* the resource-constrained and time-constrained scheduling problems, it would only find the Pareto-based tradeoff curve in the absence of the combined module selection and clock length determination problem. If this combined problem was included, the enhanced methodology would fail to find the Pareto-based curve if one of the points found by time-constrained scheduling is a suboptimal point that lies above the optimal Pareto-based curve. Such a point (which would have a non-minimal area) would

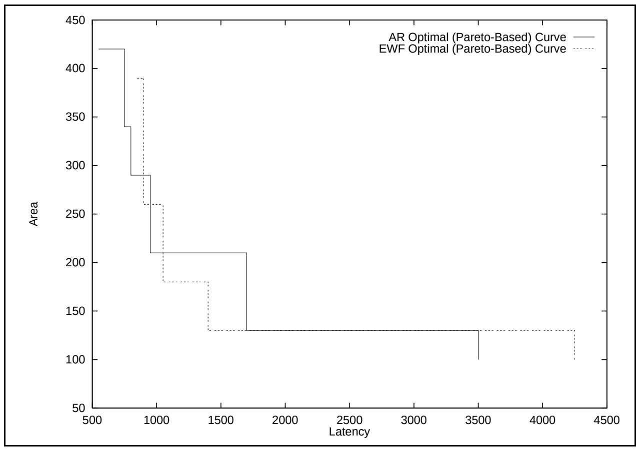

Figure 8: Results for AR-lattice Filter and EWF using library D

then be used by resource-constrained scheduling to find the minimal latency with this (non-minimal) area, thus compounding the problem and giving an erroneous design curve that actually lies above the optimal area curve based on the Pareto points.

5 Conclusions and Future Work

This paper has presented a methodology to compute *all* optimal tradeoff points (Pareto points) in order to completely characterize the design space for the module selection, clock length determination, and scheduling problems. Moreover, the methodology finds those Pareto points efficiently 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.

Tests using the DIFFEQ benchmark and a variety of module libraries have shown the importance of considering the clock length determination and module selection problems in conjunction with (rather than prior to) the scheduling problem. Without considering those additional problems, we have shown that the results are over-simplistic and do not accurately reflect the optimal tradeoff curve; in fact, they may entirely miss many globally optimal points.

Furthermore, we have shown that handling realistic module delays makes the problem much more difficult than simply increasing the number of modules of each type (compare Figures 4 and 6, or Figures 6 and 7). This goes against the "classical wisdom", which says that the module selection problem is crucial to handling modern libraries, yet the clock length determination problem should be ignored because we do not have enough information at this point in the design cycle.

It is also interesting to note that not only is the module library a significant factor in the complexity of the design space, but the structure of the data flow graph is important as well. Consider Figure 8, which shows the results for the EWF and AR-lattice benchmarks with library D. Both data flow graphs contain only additions and multiplications, but even though the EWF is more than 20% larger, the AR filter's Pareto-based tradeoff curve is more complex.

Although we have presented a methodology that solves the TCS problem optimally to completely characterize the design space, the ideas presented here can also be used to generate a preliminary characterization using only bounding techniques.[5] For example, for the DCT benchmark and library D, there are about 24,000 combinations of module sets, clock lengths, and time constraints, yet only about 2500 remain after clock length pruning. Of those, about 2400 would be identified as non-Pareto points using the heuristics, leaving only about 100 to be characterized later.

Finally, the methodology presented here could be extended to consider other problems as well. First, although this methodology solves the module selection problem, it currently ignores the type mapping problem – that is, if two different adders are available, it will choose only one, and schedule all additions onto adders of that type. Second, the methodology currently considers only functional unit area, whereas it should also consider register area, interconnect area, controller area, etc.

References

[1] D. D. Gajski, F. Vahid, S. Narayan, and J. Gong, *Specification and Design of Embedded Systems.* Englewood Cliffs, NJ 07632, USA: P T R Prentice-Hall, 1994.

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

[3] 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.

[4] 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.

[5] M. C. McFarland, "Reevaluating the Design Space for Register Transfer Hardware Synthesis," in *Proc. of the IEEE/ACM International Conference on Computer-Aided Design*, (Santa Clara, California), pp. 262–265, IEEE Computer Society Press, Nov. 9-12 1987.

[6] 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.

[7] P. G. Paulin and J. P. Knight, "Force Directed Scheduling for the Behavioral Synthesis of ASICs," *IEEE Transactions on Computer-Aided Design*, vol. 8, pp. 661–679, June 1989.

[8] S. Chaudhuri, S. A. Blythe, and R. A. Walker, "A Solution Methodology for Exact Design Space Exploration in a 3D Design Space." to appear in IEEE Trans. on VLSI.

[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] V. Chaiyakul, A. C.-H. Wu, and D. D. Gajski, "Timing Models for High Level Synthesis," in [15], pp. 60–65.

[11] S. Narayan and D. D. Gajski, "System Clock Estimation based on Clock Slack Minimization," in [15], pp. 66–71.

[12] 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.

[13] 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.

[14] 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.

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

[5]Note that the bounding methodology described here would more fully characterize the design space than the one described in [3], for the reasons explained in Section 4.2.