A Methodology for Guided Behavioral-Level Optimization

Lisa Guerra
Advanced VLSI Architecture Group
Rockwell Semiconductor Systems
Newport Beach, CA

Miodrag Potkonjak
Computer Science Department
University of California
Los Angeles, CA

Jan Rabaey
EECS Department
University of California
Berkeley, CA

Abstract — Optimization at the early stages of design are crucial. However, due to an overwhelming number of design and optimization options, design exploration is often conducted in a qualitative, ad-hoc manner. This paper presents a methodology and interactive environment for guiding the exploration process. A prototype targeting behavioral-level optimization for datapath-intensive ASIC implementations has been developed. The key to the approach is encapsulated knowledge about the various optimizations and a set of techniques to automatically extract the "essence" of a design description. At each stage in the exploration process, the system suggests and ranks potential optimizations, both in terms of immediate and longer-term impact. It also provides evaluations of the design and of the likely affects each optimization will have on metrics like power and performance. In the new approach, the designer is responsible for making the actual optimization selections. However, using the provided guidance, designers can make decisions in a more informed manner, and therefore can explore the design solution space more effectively. The effectiveness of the approach is demonstrated on a number of designs.

1.0 Introduction

Traditionally, concentration has been placed in the development of point tools and methodologies which address a single task within the optimization process. For example, in behavioral-level optimization, in addition to the approaches proposed for partitioning, template matching, and clock selection, there exist more than a hundred transformations [Bac94, Par95]. Some optimizations are restricted to optimization of a specific class of computation (e.g. only linear computations). Some focus on power optimization while others target area or throughput. Furthermore, some concentrate on reduction of just a specific sub-metric; for example, different power optimization techniques have been developed for reducing capacitance, for reducing activity, and for enabling voltage scaling.

Effective *global* optimization, or optimization of a global objective function on current-day complex and heterogeneous applications, clearly requires the coordinated application of a *collection* of techniques. While many of the problems being addressed by the individual techniques are in themselves computationally intractable, an added layer of complexity clearly exists in deriving an entire optimization trajectory. This added complexity is due to the strong interdependency that exists between techniques. Exploring different trajectories is crucial, as a large variability in attainable improvements exists. Exploring all possible sequences of actions inevitably results in combinatorial explosion, however, and thus is not a feasible option. Furthermore, static scripts are not sufficient since their effectiveness is strongly dependent on the particular design.

This paper proposes a methodology for enabling systematic and effective global optimization. The approach involves interactively and quantitatively guiding the designer's exploration process. At each stage in the exploration process, the system suggests and ranks potential optimizations, taking into consideration not only immediate benefits, but benefits which may be gained by successive optimizations. It provides evaluations of the design and of the likely affects each optimization will have on metrics like power, cost, and performance. Using the guidance environment, designers maintain a more global view of the exploration space, can make decisions in a more informed manner, and thus can more easily and quickly discover effective trajectories of optimizations.

2.0 Preliminaries

Based on the proposed methodology, an environment targeting behavioral-level design optimization for semi-custom ASIC implementations has been developed. This section presents the domain that we have focused on.

Applications are described in either the Silage high-level language or using restricted C++, then automatically parsed into a flowgraph representation. The flowgraph consists of nodes representing data operators or sub-graphs, and edges representing the data, control, and timing precedences. The flowgraph computation model is homogeneous synchronous data flow [Lee87]. Hyper behavioral synthesis [Rab91] tools are used to map the flowgraph to an architectural netlist implementation. The ASIC architecture model consists of an unlimited number of parallel, time-multiplexed execution units. Registers are clustered into register files, which are tied to the input ports of the execution units.

Before mapping, optimizations are applied to improve the design's area, power, or throughput. They can be transformations on the flowgraph itself, manual re-writes of the input specification, or library and parameter changes. Optimization examples include time and for-loop unfolding, algebraic optimizations, pipelining and retiming, voltage scaling, clock selection, operator chaining, and hardware module selection. A good overview of the behavioral synthesis area can be found in [McF90].

3.0 Related Work

Before proceeding, this section outlines related efforts in design guidance, design planning, and in performing optimization globally across the boundaries of individual techniques. A design guidance system, the Clio Design Advice System, was proposed by Lopez, Jacome, and Director in 1992 [Lop92]. This system provides design advice consisting of *static* pieces of *qualitative* information about different design options. The advice database operates as the collective memory of the entire community of

designers utilizing the framework. There have also been several related works in the area of "design planning." The ADAM Design Planning System [Kna91], for example, is a knowledge-based planner that constructs a sequence of design synthesis tasks in response to a given set of specifications. The ADAM system concentrates on the design flow of tools (e.g. module selection, allocation) to synthesize an RTL design. Other related work on includes the work of [Goo94] on global design methodologies for heterogeneous ICs and work on flow management [Kle94].

In the area of optimization ordering, there have been a number of approaches in the CAD and compilers areas. These, however, only address fixed sets of techniques. The techniques include peephole optimization [McK65], pre-defined script-based optimization [Bra84], theory-based ordering [Wol91], generic probabilistic techniques [Pot94, Cha95], bottleneck identification and elimination approaches, and approaches using enabling and disabling transformations, [Whi90, Pot92, Hua93].

The key novelties of this work are 1) a combination of *quantitative* and qualitative design advice is given, 2) design advice that is specific to the design at hand is given, 3) the designer remains an integral part of the design process, 4) the approach is extendible to include new optimizations. While the work of [Gue94] presents the use of properties for guidance through estimation, in this work, we present a complete environment for their use in guided optimization.

4.0 Proposed Methodology and System Overview

4.1 Methodology Overview
Fig. 1 depicts an overall view of the interactive guided optimization process. The designer enters a design specification, constraints, and goals; for example, a C++ description of a DCT algorithm, the required sample rate, and the goal of power optimization could be specified. The design guidance system analyzes the specified design and generates design feedback. The user interprets this information and directs the exploration by selecting a specific most promising direction from among the suggestions or by proposing another alternative. Exploration continues, with control alternating between the system and user. Note that the system keeps a global view of the design space, adapting rankings as new information is gained and thus suggesting backtracking to previous design versions as soon as the current path loses promise.

During the exploration process, the user is presented with a visual trace of the exploration (e.g. Fig. 4). This display is essentially a rooted tree representation of all the optimization sequences that have been applied so far. The root of the tree is the original design, the edges represent specific optimizations, and the nodes are the intermediate design versions. The display also shows all the optimizations suggested for further exploration and their ranking, and has links to detailed information for each.

Key advantages of the methodology are that designer expertise can be integrated into the optimization process, and the designer can learn from the system. The 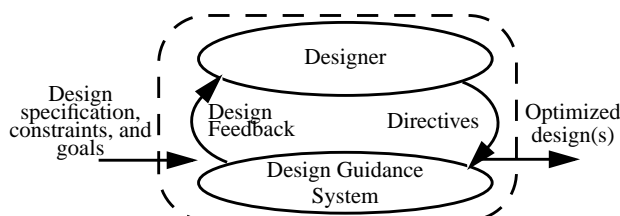experienced designer is given the freedom to synergistically combine the information provided by the system with his/her intuition, specific design ideas, and common sense. Instead of following suggestions, (s)he may modify parameters with which an optimization is invoked, may select poorly ranked suggestions, or may provide an altogether new alternative. Alternatively, (s)he can choose to either blindly follow suggestions or use the system as resource of encapsulated optimization knowledge.

Furthermore, the approach allows the use of not only the available automated techniques, but also non-automated ones. The system can propose a technique which the user can apply manually through code re-write. Using the accompanying predictions, the designer evaluates the time investment to potential reward ratio. The use of non-automated techniques greatly increases optimization power since it expands the library of techniques the designer is likely to apply from just the available ones to include all techniques that have been characterized.

Fig. 2 shows the main components of the design guidance system. The input to the system is a directive specifying a particular optimization and design version. The output is the updated design trace. The system has automated optimizations that it can apply for the designer. It also has estimators [Rab91, Meh94, Gue94] for determining the throughput, area, and power of any of the design versions. Most importantly, it has components for generating design guidance. The components contributing to this are shown in Fig. 2 in bold, and will be the primary topic of Section 4.2. These are the keys to creating guidance: a mechanism for characterizing or extracting the "essence" of a design, a pre-characterized library of optimizations, and a mechanism for evaluating and ranking options.

Before delving into the details of these blocks, we summarize the key features of the environment:

- provides estimates of the design's power, area, and throughput
- suggests relevant optimizations
- provides reasonings for their application and predictions of their effects
- applies automated optimizations
- enables the use of non-automated techniques
- maintains a global view of the exploration space, backtracking when necessary
- allows designers to integrate their expertise in the exploration
- provides optimization knowledge that designers can learn from

4.2 Guidance System
The backbone of the new methodology is guidance generation.

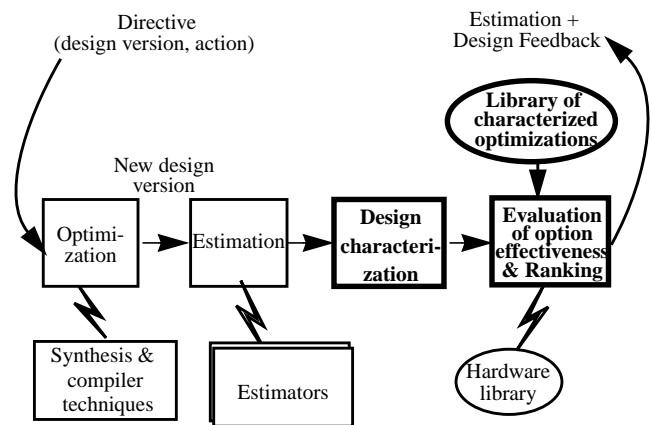Fig. 1. Guided design space exploration and optimization

Fig. 2. Components of the design guidance system

The basis of the system is a library of optimization techniques. These techniques are characterized into models of performance. These models produce cost analysis based on properties that can be extracted from the design. This section will explain the characterization of both designs and optimizations. It will also explain how this information is used to produce ranked suggestions.

4.2.1 Design Characterization

A key requisite for providing useful design feedback is a mechanism for analyzing the design being optimized. Previous research has shown the use of design characteristics, or design properties, for predicting a variety of design metrics of the final implementation [Jam87, Gue94]. Effectiveness of various optimizations can also be linked to a design's properties. The properties *identify* situations in which certain optimization techniques will work well and can be used in predicting their effectiveness. Classes of properties used in this work are outlined below. Examples of their use in optimization characterization are shown in the next section.

- Size measures includes quantities such as the number of functional operations, the bitwidth, the number of I/O and delay operations, and the number of memory accesses.
- Timing measures include the critical path, the ratio of sample rate to the critical path, the number and types of nodes on the ε-critical network, the delays of various operators, and statistics of the scheduling slack.
- Concurrency measures estimate the number of operation and interconnect accesses that can be executed concurrently.
- Uniformity metrics capture the degree to which resource accesses are evenly distributed in time.
- Temporal properties capture information about the lifetimes of variables in the computation.
- Regularity metrics quantify the degree to which common patterns appear in the control data flow graph.
- Cyclic properties include the iteration bound, the number of strongly connected components and the critical paths of each, and the percentage of operations in cycles.
- Other metrics include characterizing whether the computation is linear or non-linear and recursive or non-recursive.

When not already available, algorithms were developed for quantifying the various design properties. In the developed environment, properties are automatically extracted and are available for the interested designer to view (Fig. 3a). This raw feedback (without the guidance layer) can in itself be a great design aid that relieves the designer of mundane time-consuming calculations (e.g. of the critical path, operation counts, concurrency, etc.) that (s)he would have otherwise computed by hand. The precise definition of the assumed properties are beyond the scope of this paper, but can be found in [Gue94, Gue96].

4.2.2 Optimization Characterization

Another key step in the methodology involves identifying and characterizing relevant optimizations. An optimization is quantified by an encapsulated command-line call for its execution, by a set of trigger conditions, and by information regarding its immediate and longer term optimization potential. This information, or design *knowledge,* is encapsulated in a library of optimization characterizations. To date we have characterized over 25 optimizations, with focus on their impact on throughput, power, and area. Sample optimization characterizations can be found in [Gue96]. Characterized optimization actions can be either an atomic synthesis task (e.g. clock selection, module selection, retiming) or a static script of tasks.
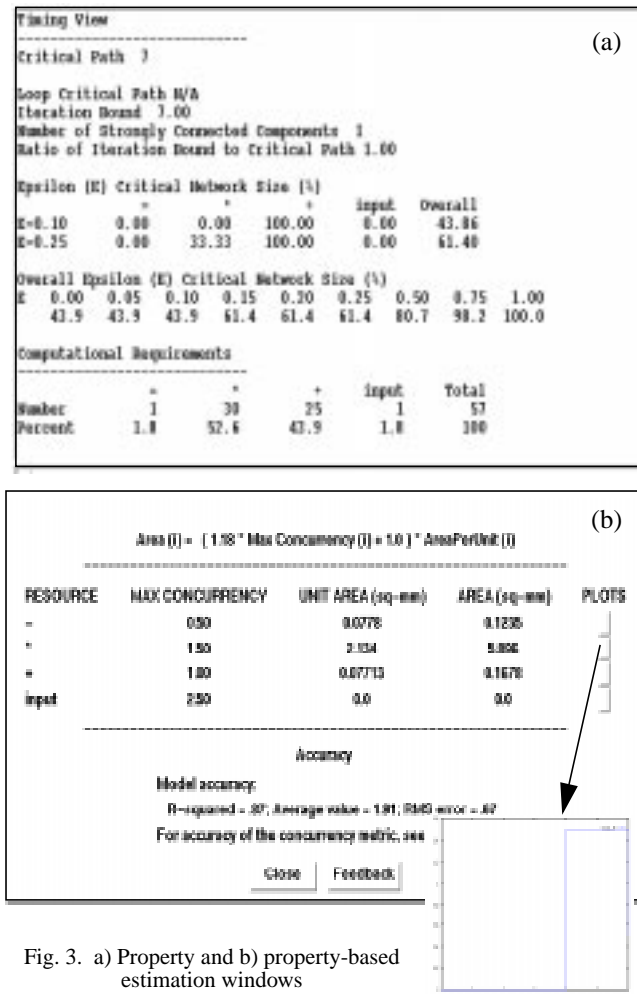

Fig. 3. a) Property and b) property-based estimation windows

The encapsulated command line calls are included so that an optimization can be automatically applied (with the appropriate parameters) when the designer selects it. The trigger conditions aid in quick pruning of options. If an option's triggers are not satisfied, the option is eliminated from consideration, and time need not be wasted generating feedback regarding its effects. Design characteristics play a major role in trigger definition. Triggers can be any boolean function of the design's properties. Example triggers include "Is the computation linear?" (for optimizations that only work on linear computations), "Does the computation have delays?" (for the delay retiming optimization), and "Does the computation have multiplications by constants?" (for constant multiplication expansion into adds and shifts).

The feedback to aid in selecting the next optimization to apply are encapsulated in the form of text and parameterized rules/equations (as a function of the design's properties). There are 3 main types of feedback that are characterized. The first type of feedback involves predicting the immediate effect that the action has on the design metrics. The second type of feedback is information on the enabling effect that the action has to enhance subsequent optimizations. We characterize enabling potential for only those optimizations identified as typical enablers/enhancers of subsequent optimizations. For these enablers, predictions are made regarding their 1 or 2-step look-ahead predictions when applied in combination with a few common transformations. The third type of feedback includes qualitative hints about the order in which to apply the optimizations (e.g. "this optimization is often effective

after applying optimization x"). All information used in the characterization is assumed to be obtained through the literature and through design experience.

The feedback is both presented textually to the designer and used in option ranking. While the ranking uses only the quantitative feedback pertaining to immediate and potential affects on the cost functions, the textual feedback can additionally contain more qualitative information (for example, information about typical optimizations with which an optimization works well).

4.2.3 Ranking

In addition to suggesting and predicting the effect of feasible options, another key component of design guidance involves their ranking. Options are ranked in order of their overall predicted effectiveness in helping to meet the specified constraint and optimization objectives. Ranking is done between all proposed optimization actions along all partially-explored optimization paths. Note that if a quantitative prediction of an optimization's effectiveness is not available, then it is not given a ranking. The designer uses the rankings in conjunction with the textual feedback to aid in selecting an optimization and specific design version on which to apply it.

There are three basic factors used to evaluate an action a_i's effectiveness on design version d_j — the resulting performance as compared to existing solutions; its potential to enhance subsequent optimization; and the resulting feasibility in meeting constraints. As an example, for the goal of power minimization under a throughput constraint T_c, measures of these three factors are defined below:

$ImmediateImprovement_{ij} = (lowest\ power\ so\ far - power_{ij})\ / $
$\qquad\qquad\qquad\qquad\qquad (lowest\ power\ so\ far)$

$if\ (a_i\ is\ not\ an\ enabler)\ \{$

$\qquad EnablingPotential_{ij} = 0$

$\}\ else\ \{$

$\qquad EnablingPotential_{ij} = 1/k,\ where\ k\ is\ a_i\text{'s}\ rank\ among\ actions$
$sorted\ in\ order\ of\ enabling\ potential\ on\ d_j\ \}$

$if\ constraints\ are\ met\ (predicted\ throughput > T_c)\ \{$

$\qquad Infeasibility_{ij} = 0$

$\}\ else\ \{$

$\qquad Infeasibility_{ij} = (predicted\ critical\ path\ /\ available\ time)\ \}$

The immediate improvement function is a measure of how the resulting power after application of a_i to d_j will compare to the lowest power attained so far among all solutions. The one with the lowest predicted power will have the highest immediate improvement function. The enabling potential is 0 if a_i is not an enabler. Otherwise it is a function of the action's predicted enabling potential as compared to other actions. The infeasibility function is 0 if constraints are met, otherwise it is a measure of how far the current achievable throughput is from the minimum throughput constraint.

The basic effectiveness R_{ij} of action a_i on design version d_j can now be formed as a linear combination of these factors. The actions are ranked in decreasing order of R_{ij}:

$$R_{ij} = \alpha_1 t_{ij} ImmedImprovement_{ij} + \frac{1}{t_{ij}} EnablingPotential_{ij} - (\alpha_2 t_{ij})^{\alpha_3 t_{ij}} Infeasibility_{ij}$$

The α_1, α_2, and α_3 are constants empirically derived to be 4, 2, and 5, respectively. The value t_{ij} is an indicator of how far the application of action a_i is into the particular optimization trajectory:

$$t_{ij} = (1 + length(d_j))/\ total\ number\ of\ applicable\ actions \qquad 0 < t_{ij} \le 1$$

It is defined as the ratio of the number of unique actions that have been applied along a given path to the total number of actions. The functions $1/t_{ij}$, $\alpha_1 t_{ij}$, and $(\alpha_2 t_{ij})^{\alpha_3 t_{ij}}$ are used to weight the sub-components of the ranking function as follows. In the beginning of an exploration trajectory, the actions with high enabling potential are favored. As the exploration progresses, the guidance favors more immediate improvements. Towards the end, ensuring feasibility becomes vitally important.

4.2.4 Interface and Implementation

An environment has been developed to demonstrate the interactive guided exploration approach. The environment's graphical user interface is written using Tcl and Tk. Embedded functions which perform the design analysis, optimization analysis, and ranking are implemented in C. The parameters passed to these functions include the design parameters and constraints, and pointers to the design (in flowgraph form), hardware library, and optimization characterization library. All invocations of existing tools are made through the framework. The Tcl-Tk interface handles all the software system glue logic and management of exploration history.

From the main window of the guidance environment, design management, estimation, or guided exploration is invoked. Design management tasks include loading the design, editing the algorithm input description, setting parameters and constraints, or exiting the tool. From the design management menu, the designer can also view the design's property metrics (Fig. 3a). Under estimations, Hyper estimation for area, speed, and power is encapsulated [Rab91, Meh94], as well as property-based area estimation. A sample property-based estimation window is shown in Fig. 3b. The plot sub-window shows a concurrency graph of subtraction operations over time. Guided design space exploration is invoked by depressing the guided exploration button, and selecting a desired performance objective. A sample screen shot for guidance will be shown in the next section.

5.0 Putting It All Together

5.1 Example

To illustrate the key ideas and the effectiveness of the new methodology, this section presents the highlights of a sample "session" of an ASIC design of a General Electric (GE), state-space, 5-state, 32-bit linear controller [Cha93]. The goal is to provide a feel for the system, the types of feedback and guidance that it provides, and the underlying design methodology. Assuming a required sample rate of 1.2 MHz, direct synthesis using the Hyper behavioral level synthesis tools and Berkeley Low-Power library, gives a design with estimated power [Meh94] of 441 mW at 5 volts using a 1.2 μm technology.

Upon commencement of the guided optimization exploration session, the system characterizes the design then prunes a number of non-applicable optimizations. For example, library selection is not triggered since the low-power library has more effective modules with respect to power than the alternate dpp library [Bro92]. Clock selection is not suggested since it has already been set efficiently by the user. The user-specified description does not have common sub-expressions or for-loops so CSE and for-loop unfolding are not applicable. Finally, direct voltage scaling is not triggered since there is no difference between the critical path and the available time.

The system proposes and ranks a number of actions that may result in immediate or subsequent power reduction. In this case,

suggested actions are pipelining, constant multiplication replacement with additions and shifts, time loop unfolding, and the "maximally fast" script [Pot92]. For each action it provides feedback to guide the designer's selection. Information regarding pipelining, for example, includes a bound on critical path improvement attainable by pipelining; in this case, it can reduce the critical path from 12 clock cycles to the iteration bound of 6 clock cycles. A power prediction is also provided based on an empirical voltage-delay curve [Cha95]; in this case, the critical path reduction has enabled a voltage reduction to 2.8 volts, giving an estimated power reduction of 3.2. Constant multiplication expansion into additions and shifts is another potential optimization. There are many constant multiplications: 36 out of 67 total operations. Furthermore, their bitwidth of 32 results in a large difference between multiplier and adder power dissipation, delay, and area. The maximally fast script is also an attractive alternative since it can reduce the critical path to at least 4. The maximally fast static transformation script combines several algebraic and redundancy manipulation transformations for critical path and operation reduction.

While each of the mentioned tasks will result in immediate power reduction, the system proposes time loop unfolding as the most attractive alternative. This technique reduces the critical path to the iteration bound of the design, but is favored not for it's immediate benefit (which is actually very similar to pipelining's), but rather for the long-term improvement potential that it provides for greater subsequent optimization. In particular, as indicated by the tool's analysis, the computation is linear (formal degree is 1) and therefore unfolding enables effective application of the "maximally fast" technique simultaneously on several iterations of the computation. The specific unfolding factor is selected by the designer. The system provides a table (partial table shown in Table 1) of suggested voltage and number of operations for various unfolding factors (closed form equations exists for computing the estimated critical path and number of operations). The suggested voltage is based on maintaining some slack between the resulting critical path after voltage scaling and the available time (a slack of 10% is used). Based on this table and qualitative feedback that control overhead increases with greater unfolding, the designer selects an unfolding factor of six to balance achieving both speedup for voltage scaling and reduction in capacitance.

Application of time loop unfolding followed then by the suggested maximally fast script reduces both the effective critical path by a factor of 21 and more importantly reduces the effective number of operations (the number of operations per iteration) by a factor of 2. This in turn enables application of the MCM technique [Pot94] for conversion of constant multiplications to series of shifts and

Unfolding factor	New critical path	Speedup	Suggested voltage	Number of additions and multiplications
0	4	3	2.2	66
5	.67	18	1.2	33.5
6	.57	21	1.2	33.43

Table 1: GE controller example — the impact on attainable voltage and operation counts for various unfolding factors.

additions, reducing the effective capacitance per operation. The final steps are suggested by the environment so that immediate improvements are maximized. They involve lowering the voltage from 5 to 1 volts and performing clock selection for efficient utilization of the clock period. The optimized design has a power of 2.9 mW. For this example, not only is power reduced by a factor of 151, but area was also reduced by a factor of 2.

Fig. 4 shows a snapshot of the exploration session. The left-most justified actions are those that were initially suggested (unfolding,
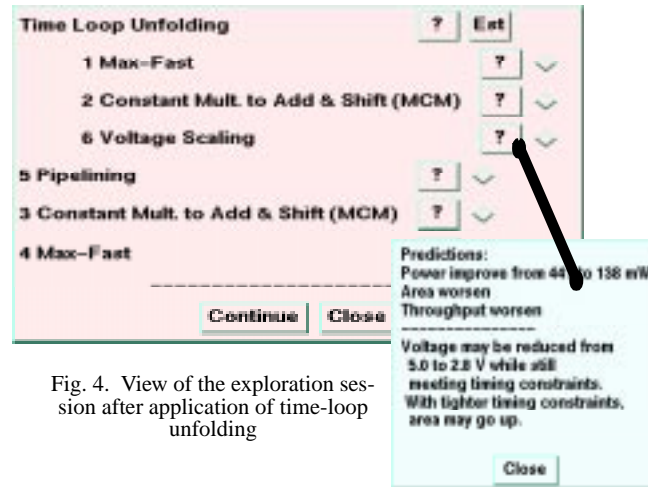

Fig. 4. View of the exploration session after application of time-loop unfolding

pipelining, MCM, and max-fast). Application of time-loop unfolding has already been done through selection of the appropriate radio button and the "Continue" button. Several optimizations are now suggested for application to the new unfolded design version (e.g. max-fast, MCM, voltage scaling). Note that ranking is done on both optimizations suggested for application to the new unfolded design (ranks 1, 2, and 6) as well as to the original design (ranks 3, 4, 5). Specific feedback regarding a suggested action is provided by selecting the corresponding information button marked with a "?" (see sub-window in Fig. 4). The information for voltage scaling is displayed.

5.2 Experimental Results

Table 2 presents the obtained improvements in energy and area. The designs include a 6th order IIR filter, a bandpass filter, cordic, 5 and 7 point convolutions, an 8-point fft, an image convolver, a

Design	Initial cap. (C)	Final C	C reduction	Energy improvement	Area improvement	Optimization steps
6 IIR	9.08	0.38	23.9	260.2	5.67	U4, MF, MCM + CS, AIB, MP, V, CS
bandpass	12.26	1.54	7.96	86.7	2.20	CM + CS, U7, V, CS
conv5	11.20	0.04	280	3049.0	32.57	CM + CS, U5, V, CS
conv7	3.34	0.04	83.5	909.3	26.9	CM + CS, U6, V, CS
cordic	49.96	32.76	1.53	16.6	1.08	U8, MP, V, CS
fft8	5.37	1.02	5.26	57.3	2.49	U5, CM, V, CS
imagec	38.82	36.08	1.08	11.7	0.95	U2,MP, V, CS
modem	38.67	0.80	48.3	526.4	16.19	CM + CS, U4, AIB, P2, V, CS
steam	137.58	85.50	1.61	17.5	0.49	CM + CS, U3, AIB, P2, V, CS
volterra 2	7.42	1.03	7.20	78.5	4.36	U4, CM + CS, AIB+ MP, V, CS
volterra 3	59.90	13.64	4.39	47.8	2.77	U3, CM + CS, AIB+ MP, V, CS
wangd	15.90	2.41	6.60	71.85	2.86	U4, CM, V, CS

Table 2: Experimental Results: minimizing power under a fixed throughput constraint

modem example, a large controller for a steam plant, 2nd and 3rd order non-linear Volterra filters, and the Wang 8-point DCT. Note that most designs have feedback, and several including the Volterra filters, image convolver, and cordic are nonlinear. The latter 2 are also control dominated with a high percentage of conditionals. The last column of Table 2 identified the optimization steps used for each design. The abbreviations used in the table are defined as follows: UN - unfolding by a factor N; CM - substitution of constant multiplications by addition and shifts; MCM - substitution of constant multiplications by addition and shifts using the MCM transformation; MF - maximally fast algorithm; MP - maximal pipelining; PN - pipelining using N stages; R - retiming; AIB - associativity for iteration bound; A - associativity; V - voltage scaling; CS - clock selection. Note that a "+" is used to indicate optimizations that are suggested together as a script (e.g. CM+CS).

For all examples, we assume that clock selection has been initially performed and that a starting voltage of 3.3 Volts is used. For all examples, the final voltage attained was 1 Volt. The maximum, minimum, average and median capacitance improvements were by a factor of 280, 1.08, 7.68, and 33.6 respectively. The maximum, minimum, average and median energy improvements were by factors of 3049, 11.7, 126.8 and 83.6, respectively. Simultaneously for area, the maximum, minimum, average and median improvements were 32.57, 0.49, 7.88 and 2.86 times, respectively. Note that although the resulting sequences of optimization steps are in many cases similar, rarely are they identical. Some of the key issues the design guidance feedback helped in included determining the unfolding factor, determining how much to pipeline (if at all), freeing the designer from many manual calculations, and eliminating optimizations. While in these examples we lowered voltage always to 1 V, in practice the designer will likely target a variety of voltages.

Obtained improvements when targeting exclusively area reduction for the same set of designs are also available, but were omitted here due to space.

6.0 Summary

A novel methodology for guided design space exploration has been presented. It has been applied for behavioral-level optimization of datapath-intensive semi-custom ASIC implementations. The approach encourages systematic and quantitative as opposed to ad-hoc global optimization across point optimizations. The intelligence of both the system and designer can be leveraged upon. The methodology and environment facilitates these goals using a library which characterizes design optimizations, mechanisms for ranking design alternatives, and techniques for analysis of designs.

One of the key unanswered issues concerns the portability of the presented ideas to new application areas (e.g. control-intensive applications) and architecture models. A change in either would require a new design characterization where some metrics are unchanged, some are modified, and some are altogether new. It would also require establishing the relations between these metrics and performance. The hope is that with each new domain addressed, the number of new metrics needed will decrease, and at some point, a core superset of metrics will be established. Other areas of future work include continued growth and improvement of the optimization "knowledge database," continued research on ranking and partitioning, and the use of automated searches to complement the user-driven exploration.

7.0 References

[Bac94] Bacon, D.F., Graham, S.L., Sharp, O.J. Compiler transformations for high performance computing. *ACM Computing Surveys*, Vol. 26, No. 4, 345-420.

[Bra84] Brayton, R.K., et al. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer, Boston, MA, 1984.

[Cha93] Chatterjee, A., Roy, R.K., d'Abreu, M.A. Greedy hardware optimization for linear digital circuits using number splitting and refactorization. *IEEE Trans. on VLSI Systems*, Vol.1, No.4, 423-431, 1993.

[Cha95] Chandrakasan, A., et al. Optimizing power using transformations. *IEEE Trans. on CAD*, Vol. 14, No. 1, 1995.

[Goo94] Goossens, G., Bolsens, I., Lin, B., Catthoor, F. Design of heterogeneous ICs for mobile and personal communication systems. *ICCAD*, 524-531, 1994.

[Gue94] Guerra, L., Potkonjak, M., Rabaey, J. System-level design guidance using algorithm properties. *VLSI Signal Processing Workshop*, 73-62, 1994.

[Gue96] Guerra, L. Behavioral-level design guidance for ASIC implementations. Ph.D. dissertation, UC Berkeley Dept. of EECS, 1996.

[Hua93] Huang, S., Rabaey, J. Maximizing the throughput of high performance DSP applications using behavioral transformations. *European Design & Test Conf.*, 25-30, 1994.

[Jam87] Jamieson, L. Characterizing parallel algorithms, in *The Characteristics of parallel algorithms*, L. Jamieson, D. Gannon, R. Douglass (eds.), MIT Press, Cambridge, Mass., 1987.

[Kle94] Kleinfeldt, S., et al. Design methodology management. *Proc. of the IEEE*, Vol.82, No.2, 231-50, 1994.

[Kna91] Knapp, D. W., Parker, A. C. The ADAM design planning engine. *IEEE Trans. on CAD*, Vol.10, No.7, 829-46, 1991.

[Lee87] Lee, E.A. and Messerschmitt, D.G. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Trans. on Computers*, Vol. 36, No. 1, 24-35, 1987.

[Lei83] Leiserson, C. and Saxe, J. Optimizing synchronous systems. *Journal of VLSI and Computer Systems*, Vol.1, No.1, 1983.

[McF90] McFarland, M.C., Parker, A.C., Camposano, R. The high-level synthesis of digital systems. *Proc. of the IEEE,* Vol. 78, No. 2, 301-317, 1990.

[McK65] McKeeman, W.M. Peephole optimization. *Comm. of the ACM*, Vol. 8, No. 7, 443-444, 1965.

[Meh94] Mehra, R. and Rabaey, J. Behavioral level power estimation and exploration. *Int'l Workshop on Low-Power Design*, 197-202, 1994.

[Par95] Parhi, K.K. High-level algorithm and architecture transformations for DSP synthesis. *Journal of VLSI Signal Processing*, Vol. 9, No. 1-2, 121-143, 1995.

[Pot92] Potkonjak, M. and Rabaey, J. Maximally fast and arbitrarily fast implementation of linear computations. *ICCAD*, 304-308, 1992.

[Pot94] Potkonjak, M., Srivastava, M., and Chandrakasan, A. Efficient substitution of multiple constant multiplications by shifts and additions using iterative pairwise matching. *Design Automation Conference*, 189-194, 1994.

[Rab91] Rabaey, J., et al. Fast prototyping of data path intensive architectures. *IEEE Design and Test*, Vol. 8, No. 2, 40-51, 1991.

[Whi90] Whitfield, D. and Soffa, M.L. An approach to ordering optimizing transformations. *ACM Symposium on Principles and Practice of Parallel Programming*, 137-147, 1990.

[Wol91] Wolf, M. and Lam, M. A loop transformation theory and an algorithm to maximize parallelism. *IEEE Trans. on Parallel and Distributed Systems,* Vol. 2, No. 4, 452-471, 1991.