Behavioral Synthesis via Engineering Change

Milenko Drinić
Computer Science Department
University of California
Los Angeles, CA 90095
milenko@cs.ucla.edu

Darko Kirovski
Microsoft Research
One Microsoft Way
Redmond, WA 98052
darkok@microsoft.com

ABSTRACT

Engineering change (EC) is a technique that enables a designer to rapidly perform minor specification alternations while minimally resynthesizing only small portions of the specification throughout several levels of design abstraction. In this paper, we introduce the first EC-based synthesis technique for coordinated design optimization in multiple steps. The technique has four phases: optimization region identification, feedback formulation, resynthesis in first step, and finally resynthesis in the second design step. To demonstrate the technique, we focus on behavioral synthesis and transformation, scheduling, and register assignment steps. We developed a generic EC-based approach for design optimization during multiple consecutive synthesis steps. Next, we show how one can use EC to enhance coordinated application of transformations and scheduling, and scheduling and register assignment.

Categories and Subject Descriptors

B.5.2 [**Register-Transfer-Level Implementation**]: Design AidsOptimization

General Terms

Design

Keywords

Engineering change, transformations, scheduling, register assignment

1. INTRODUCTION

The modern design process is long and complex with numerous subtasks that interact in convoluted ways. The conceptual complexity of applications and quantitative exponential growth of silicon platforms will make the process additionally cumbersome. Even if we restrict our attention to behavioral synthesis only, we see a number of complex tasks (transformation, partitioning, resource allocation, scheduling, assignment, register and interconnect allocation and assignment, module and clock selection, and memory allocation) interacting.

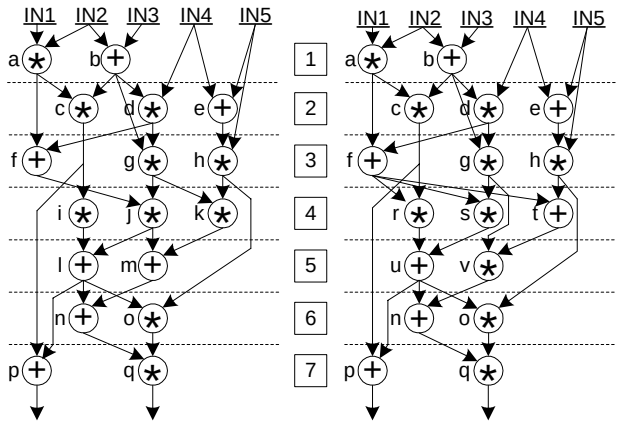
While each of these steps is associated with solving NP-hard problems, the real difficulty is in the layered nature of subtasks and their mutual impact. For example, the standard behavioral synthesis flow is transformations - scheduling - register assignment. During scheduling the primary goal is to minimize the number of execution units and interconnect. However, decisions made during scheduling greatly impact the register requirements. Since the area of a register is almost comparable to the area of an adder or a barrel-shifter and often registers dominate power consumption, it would be advantageous to consider the impact on all components on the final cost during each of steps as well as their interaction. However, the fundamental principle of separation of concerns with implication and emphasis on modularity, precludes this direction.

Nevertheless, there is an elegant and powerful need to address this issue without violating the principle of separation of concerns. Interestingly, the basis for solving this problem has unexpected roots: EC. EC is a technique that facilitates minor specification changes while inducing the need for minimal resynthesis of only small parts of the specification throughout other levels of the design abstraction. Until now, EC has been widely and solely used for enabling localized functional and timing changes, most often during debugging or after initial specification alternations. We introduce the first EC-based synthesis technique for coordinated design optimization in multiple steps. The technique has four phases: optimization region identification, feedback formulation, resynthesis in the first step, and finally resynthesis in the second design step. We focus our effort on the use of EC as a tool for coordinated optimization between multiple tasks in the design flow in the behavioral synthesis steps: transformation, scheduling, and register assignment steps. The starting point is a generic EC-based approach for design optimization during multiple consecutive synthesis steps. We demonstrate how a designer can use EC to enhance the coordinated application of transformations and scheduling, scheduling and register assignment.

1.1 Motivational Example

Consider an example of a control-data flow graph (CDFG) with ten multiplications and seven additions shown in Figure 1(a). The CDFG has been scheduled in seven control steps. For the given schedule, the computation requires at least three multipliers and two adders. Note that because of dependencies among operations, the operations scheduling alone cannot further reduce the hardware cost.

DAC 2002, June 10-14, 2002, New Orleans, Louisiana, USA.



(a) An example of a scheduled CDFG

(b) The resulting CDFG after applied full cycle of the transformations-scheduling EC optimization procedure

Figure 1: An example of application of the transformations-scheduling EC iterative optimization procedure.

The scheduling bottleneck occurs in control step 4 where three multiplications need to be executed concurrently. Instead of proceeding with the solution as is into subsequent design steps, the scheduling algorithm is modified such that it indicates its bottleneck to the previous design step. In the CDFG from Figure 1(a), operations i, j, and k are selected for the transformations procedure with the highest priority. In order to enable flexibility in transformations, operations l and m (as operations whose inputs are outputs of bottleneck operations i, j, and k), and operations c, f, g, and h (as operations whose outputs are inputs to bottleneck operations) are also marked for changes with the transformations procedure. These operations are ranked by two different criteria. First, operations scheduled closer to the bottleneck control step have a higher priority in the operation selection process for transformations (g and h) then the operations further from it (c). Second, operations of the same type (g and h) as the bottleneck operations are assigned a higher rank than the selected operations of a different type (f).

Transformations are selected from the library of transformations, and applied to the operations indicated by the scheduling. After applying replication on operation j, inverse distributivity on i, j, and l, and j' (replicated node), k, and m scheduler identifies another bottleneck, again in control step 4. This time the bottleneck and the highest priority operations are additions. The resulting CDFG and its schedule after applied distributivity on one addition from control step 4, and one multiplication from control step 5 nodes r is shown in Figure 1(b). This CDFG and its scheduling are balanced so the global procedure exits the iterative loop and continues synthesis subsequent lower design levels. More details for transformations-scheduling EC iterative optimization procedure is presented in Subsection 5.1.

2. RELATED WORK

Engineering change recently attracted a great deal of attention. However, EC efforts have longer history. In 1986, Shinsha et al. [9] proposed an incremental technique for supporting changes during physical design. Kirovski et al [4] defined the engineering change problem as constraint manipulation where the instance of the problem is restructured in such a way that it supports EC.

An inclusive survey of behavioral synthesis can be found in [2]. Scheduling and Register assignment (regularly abstracted as graph coloring) are standard tasks in both compilers [7] and behavioral synthesis [2]. The difficulty and suitability of optimization of the scheduling problem is function of both computation complexity and underlying computational model [5], targeted architecture and optimized metrics. We focus on the synchronous data-flow model [5] of computation that provides an exceptional trade-off in terms of the number and importance of application domains, and suitability for optimization.

Kernighan and Lin introduced the first iterative improvement heuristic and applied it to graph partitioning [3]. Numerous improvements on the basic strategies have been proposed over the years [1]. Iterative improvement has been also applied to many other optimization problems [6].

3. PRELIMINARIES

We have chosen as a computational model the synchronous data flow (SDF) model [5]. We restrict our attention to homogeneous SDF (HSDF), where each atomic operational unit consumes and produces exactly one sample per each input and output in every execution cycle. The syntax of a targeted computation is defined as a control-data flow graph (CDFG) [8]. The CDFG represents the computation as a flow graph, with nodes, data edges, and control edges. The semantics underlying the syntax of the CDFG format is that of the SDF model.

Behavioral synthesis transforms a given behavioral specifications into a RTL description from the input behavioral descriptions (or algorithms). This high-level abstraction provides a more powerful and complete way to explore the design space.

Scheduling is the process of partitioning a CDFG into groups of operations such that the operations from the same group are executed in a single control step. The objective of the scheduling procedure is the minimization of the total number of control steps and the total hardware cost. These objectives are, in general, in contradiction, therefore one must consider possible trade-offs when the objective function of the scheduling algorithm is constructed.

Register assignment is a process of binding each variable to a storage unit. The goal of register assignment is minimization of the total number of used registers. Many register allocation algorithms for CDFGs without loops focus either on unconditional register sharing, or conditional register sharing. Register assignment can be performed by coloring the interval graph.

4. GENERIC DESIGN FLOW

Consider a pair of synthesis/optimization tasks A and B, where task B uses the output of task A as its input. The quality of the solution provided by task B is highly dependent on the quality of the solution provided by task A. The solution of task A can be of high quality based on criteria ensuing from the level of abstraction of task A. At the same time, task B could be prevented, upfront, from obtaining a good quality solution because its input is such that there is little or no room for task B to perform good synthesis measured on effectiveness by the criteria of task B.

We propose a new EC-based synthesis technique that in-

Run task A
Run task B
While the solution of B unbalanced
  Identify bottleneck region $\mathcal{BR}$ of B
  Identify ε-neighborhood of $\mathcal{BR}$
  Rank operations/nodes
  Run task A on $\mathcal{BR}$ and ε-neighborood
  Run task B on modified $\mathcal{BR}$ and ε-neighborood
end while

Figure 2: Pseudo-code of iterative segment of the generic design flow for two consecutive synthesis/optimization tasks A and B.

tegrates two subsequent abstraction level tools through iterative collaboration. The key idea behind this approach is the passing information from the lower level to the higher level of abstraction, and the resynthesis of only small portions of the design specifications. The pseudo-code of the generic design flow is shown in Figure 2. It depicts the part of the full design flow which includes collaboration of two consecutive tasks.

Initially, both tasks, A and B are run without any changes. When task B finishes synthesis, it is necessary to analyze the solution, and formulate feedback for task A. This analysis includes identification of portions of the design that represent bottlenecks of the design causing the design to be unbalanced. Task B ranks variables, operations, or nodes in such a way that highest ranked items are those entailing the highest cost in the bottleneck. Immediate, ε-neighborhood of the bottleneck region is also selected, and ranked accordingly. The last step of the formulation of the feedback for task A is to make guidelines for its objective function. These guidelines indicate in which the direction optimization should proceed, and they are given in form of parameters included in the objective function of the task A.

After the feedback for the task A has been formulated, the task A performs a resynthesis procedure only on the portion of the design specified by the feedback. The resynthesis procedure is guided with the feedback also through a set of objectives. For example, if task B is an operation scheduling procedure, besides the selection and ranking of operations that need to be transformed, it indicates which type of operations needs to be minimized. Upon completion of task A, task B is run again, only on the bottleneck region. Since both tasks are performed only on small portions of the design, the total running time is not significantly increased.

5. APPLICATIONS OF EC SYNTHESIS

5.1 Transformations - Scheduling Procedure

The conceptual complexity of transformations makes design synthesis difficult to be algorithmically defined through them. A localization of a targeted area for the optimization can open a clear path for a set of transformations toward the specified goal. We have built a library of transformations from a restricted set which includes algebraic laws and redundancy manipulation, and sub-operational level transformations. The latter set has a powerful property of enabling a wide range of transformations from different sets.

The goal of transformations is to minimize the weighted total number of operations in the computation trough transformations. We define the objective function which needs to be minimized during an optimization in the following way:

$$OF = \sum_{\mathcal{T}_i=\mathcal{T}_1}^{\mathcal{T}_n} R_{\mathcal{T}_i} \cdot \alpha_{\mathcal{T}_i} \cdot |op_{\mathcal{T}_i}| - \omega$$

where $R_{\mathcal{T}_i}$ is the sum of ranks, $\alpha_{\mathcal{T}_i}$ represents the hardware cost, $|op_{\mathcal{T}_i}|$ is the cardinality of operations of type $\mathcal{T}_i$, and ω is proportional to the total number of available transformations.

Optimize CDFG with transformations
Schedule CDFG
While $|op_{\mathcal{T}}^{cs_i}| - AVG(|op_{\mathcal{T}}^{cs}|) > C1$
 Select:
  $op_{\mathcal{T}}^{cs_i}$; $op_j | out(op_i) = in(op_{\mathcal{T}}^{cs_i})$; $op_k | in(op_k) = out(op_{\mathcal{T}}^{cs_i})$
 Rank:
  $R(op_{\mathcal{T}}^{cs_i}) = HR$; $R(op^{cs_l}) = LR > 0$,$|cs_l - cs_i| = max$
  $LR < R(op_k), R(op_j) < HR$,$op_k \neq op^{cs_l}$, $op_k \neq op^{cs_l}$
  Transform selected op
  Reschedule selected op
end while

Figure 3: Pseudo-code of computational transformations - scheduling EC iterative optimization procedure.

The pseudo-code of transformation - scheduling EC iterative optimization procedure is depicted in Figure 3. In the initial run of transformations, the rank of each operation is a constant, and equal for all operations. In a scheduled CDFG, the bottleneck region is identified as a set of operations entailing the highest cost. These operations (labeled as $op_{\mathcal{T}}^{cs_i}$) are all of the same type $\mathcal{T}$, belong to a single control step cs_i, and are assigned the highest rank HR. In order to enable a reasonably wide spectrum of transformations, the operations whose inputs are outputs of $op_{\mathcal{T}}^{cs_i}$, and those whose outputs are inputs to of $op_{\mathcal{T}}^{cs_i}$ are also selected and assigned a non-zero rank. Higher rank is assigned to these operations if: (i) they are scheduled closer to the bottleneck control step, and (ii) they are of type $\mathcal{T}$.

The iterations between transformations and scheduling are continued while there is a control step entailing significantly more resources than the remainder of control steps on the average. Once when the peaks of hardware requirements are eliminated, it is not likely that further iterations which are performed locally will bring a substantial benefit. An example of transformation - scheduling EC iterative optimization procedure is introduced as the motivational example in Subsection 1.1.

5.2 Scheduling - Register Assignment Procedure

The pseudo-code of scheduling - register assignment EC iterative optimization procedure is depicted in Figure 4. The initial scheduling of CDFG implies that the complete EC iterative optimization procedure described in Subsection 5.1 has been processed. The cut in the variable lifetime graph over a single control step cs_i, $Cut(cs_i)$ is defined as a number of live variables in that control step. The condition for entering the iterative collaboration of two design stages is that the largest $Cut(cs_i)$ is larger for more than empirical constant $C2$ than the average cut over a single control step, $AVG(Cut(cs))$.

The information where each variable is consumed is lost on the interval graph level. The register assignment (graph coloring) algorithm passes only the information about variables are in $Cut(cs_i)$ and the number of the critical step cs_i. This cut is causing the minimal required number of registers to be also nine. On the abstraction level of CDFG, operations belonging to the bottleneck region are selected by three criteria.

Design name	Number of operations	Active area [mm^2]								Improvement [%]		
		Original		Trans.-Sche.		Sche.-Reg.Asg.		Trans.-Sche.-Reg.Asg.				
		EU	Regs	EU	Regs	EU	Regs	EU	Regs	EU	Regs	Total
volterra	30	0.96	1.17	0.95	1.1	0.96	1.09	0.95	1.06	1.0	9.4	5.6
conv5	45	1.19	2.41	1.09	2.14	1.19	2.05	1.09	1.92	8.4	20.3	16.4
LinearCntrl3	56	3.92	9.11	3.71	8.21	3.92	6.56	3.71	6.14	5.4	32.6	24.4
Differentiator	80	1.16	3.40	1.04	3.16	1.16	2.68	1.04	2.60	10.3	23.5	20.2
Winogradfft11	104	2.56	6.29	2.13	5.41	2.56	4.34	2.13	4.02	16.8	36.1	30.5
DSkais55	137	3.00	10.12	2.52	8.50	3.00	6.78	2.52	6.21	16.0	38.6	33.4
fir100	302	4.18	16.41	3.39	13.33	4.18	10.83	3.39	9.93	18.9	39.5	35.3

Table 1: Impact of EC based behavioral synthesis on active area of the design. Active area for four different design flows are shown: (i) original, (ii) enhanced with only transformation-scheduling iterative procedure, (iii) enhanced with only scheduling-register assignment iterative procedure, (iv) enhanced with both iterative procedures. Active area shown separately for execution units (EU) and registers.

Schedule CDFG
Assign registers
While $Cut(cs_i) - AVG(Cut(cs)) > C2$
Select:
$op_j \in cs_i$; $op_l | v_k \in Cut(cs_i) \wedge v_k \rhd cs_i \wedge$
$LT(v_k) \geq AVG(LT(v)) \wedge in(op_l) = v_k$;
$op_m | LT(v_m) \ll AVG(LT(v)) \wedge out(op_m) = in(op_j)$
$\vee out(op_m) = in(op_l)$
Rank:
$R(op_j) = HR$; $R(op_m) = LR > 0$, $LR < R(op_l) < HR$
Reschedule selected op
Reassign registers for affected variables
end while

Figure 4: Pseudo-code for scheduling-register assignment EC iterative optimization procedure.

First, operations op_j in the critical step cs_i are selected and assigned the highest priority. Second, consider a variable v_k with a long lifetime LT which is consumed (indicated by $\rhd$ in the pseudo-code in Figure 4) both in cs_i by an operation already selected into the bottleneck region, and by one or more operations outside of cs_i. This variable is highly likely causing the bottleneck in register assignment, so operations op_l consuming v_k are selected for rescheduling. The third criterion has a purpose to make the bottleneck region more amenable to rescheduling. By this criterion, operations op_m, whose outputs are inputs of already selected operations by the first two criteria, are also chosen to be added in the bottleneck region. The restriction is established by the lifetime of the produced variables by op_m which needs to be short compared to average lifetime of variables.

As a scheduling and register assignment algorithm we have used the standard HYPER synthesis script [8]. The objective functions of both optimization procedures were augmented to accommodate the ranking of operations, variables, or nodes, and limited scope determined by the bottleneck region and its ε-neighborhood.

6. EXPERIMENTAL RESULTS

The selected benchmarks for evaluating the effectiveness of our approach include designs of the volterra filter, a convolution, a linear controller, Winogradfft11 transformation, a differentiator, and FIR filters (DSkais55, and fir100).

The experimental results are presented in Table 1. We have conducted four different types of experiments, and mapped solutions on .25μ technology. The initial results (shown in the third and the forth column) have been obtained using the standard HYPER synthesis script [8]. In the next six columns, active area is shown for execution units and register of the following design flow enhancements: only transformation-scheduling EC iterative optimization, only scheduling-register assignment, and these two optimizations combined. The area improvement for execution units, registers and the overall improvement are shown in the last three columns of Table 1. During our experiments, we have encountered only minimal run time overheads not exceeding 5% of the original run time for each benchmark.

7. CONCLUSION

In this work, we have introduced the first behavioral synthesis technique based on EC for coordinate design optimization. The collaboration is established between of two subsequent design stages where the latter one formulates and presents feedback information based on its bottleneck regions, to the first design stage. The developed technique assumes multiple resynthesis steps of two design stages which are performed in alternating order. The generic EC-based design approach has been introduced with examples of application of the technique to transformations and scheduling, and scheduling and register assignment. We have demonstrated the effectiveness of the new technique on a number of benchmarks while entailing minimal run-time overhead.

8. REFERENCES

[1] C.J. Alpert and A.B. Kahng. Recent Directions in Netlist Partitioning: A Survey. Integration: The VLSI Journal, Vol.19, (no.1-2), pp.1–81, 1995.

[2] G. De Micheli. Synthesis and optimization of digital circuits. McGraw-Hill, 1994.

[3] B.W. Kernighan and S. Lin. An Efficient Heuristic Procedure for Partitioning Graphs. Bell System Technical Journal, Vol.49, (no.2), pp.291–308, 1970.

[4] D. Kirovski and M. Potkonjak. Engineering change: methodology and applications to behavioral and system synthesis. Proc. of Design Automation Conference, pp.604–609, 1999.

[5] E.A. Lee and D.G. Messerschmitt. Synchronous dataflow. Proceedings of the IEEE, Vol.75, (no.9), pp.1235–45, 1987.

[6] S. Lin and B.W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. Operations Research, Vol.21, (no.2), pp.498–516, 1973.

[7] S. S. Muchnik. Advanced Compiler Design & Implementation. Morgan Kaufmann Publishers, Inc., 1997.

[8] J. Rabaey, C. Chu, P. Hoang, and M. Potkonjak. Fast prototyping of data path intensive architectures. Design & Test of Computers, Vol.8, (no.2), pp.40–51, 1991.

[9] T. Shinsha, T. Kubo, Y. Sakataya, J. Koshishita, and K. Ishihara. Incremental logic synthesis through gate logic structure identification. Proc. of Design Automation Conference, pp.391–397, 1986.