

Coordinated Resource Optimization in Behavioral Synthesis

Jason Cong Bin Liu Junjuan Xu
Computer Science Department, University of California, Los Angeles
Email: {cong, bliu, irene.xu}@cs.ucla.edu

Abstract—Reducing resource usage is one of the most important optimization objectives in behavioral synthesis due to its direct impact on power, performance and cost. The datapath in a typical design is composed of different kinds of components, including functional units, registers and multiplexers. To optimize the overall resource usage, a behavioral synthesis tool should consider all kinds of components at the same time. However, most previous work on behavioral synthesis has the limitations of (i) not being able to consider all kinds of resources globally, and/or (ii) separating the synthesis process into a sequence of optimization steps without a consistent optimization objective. In this paper we present a behavioral synthesis flow in which all types of components in the datapath are modeled and optimized consistently. The key idea is to feed to the scheduler the intentions for sharing functional units and registers in favor of the global optimization goal (such as total area), so that the scheduler could generate a schedule that makes the sharing intentions feasible. Experiments show that compared to the solution of minimizing functional unit requirements in scheduling and using the least number of functional units and registers in binding, our solution achieves a 24% reduction in total area; compared to the online tool provided by `c-to-verilog.com`, our solution achieves a 30% reduction on average.

I. INTRODUCTION

Behavioral synthesis (also referred to as high-level synthesis) is regarded as a promising solution for managing the increasing design complexity and improving productivity. In the 1980s and the 1990s, despite numerous efforts on research and development of behavioral synthesis tools in both academia and industry, the success of these tools was limited in practice for multiple reasons, among which inferior quality of result (QoR) in terms of performance/area/power compared to manual designs is notable.

Many recent studies on behavioral synthesis have shifted their focus to embrace a broader category of considerations such as reliability, temperature and yield. Yet, our experience with several behavioral synthesis tools indicates that the efficient use of resources under a performance constraint is still a fundamental challenge. This is confirmed in other recent comparative studies of commercial behavioral synthesis tools, such as [1]. Improvement in resource usage usually implies a more efficient RTL structure, less global interconnects, lower power dissipation and lower temperature. In this paper we revisit the topic of resource optimization in behavioral synthesis.

Achieving good QoR in behavioral synthesis is hard considering the essential difficulty of the synthesis problem and the huge design space. To manage the complexity, some seminal work decomposes behavioral synthesis into several subtasks, namely resource allocation, scheduling and binding [2]–[4].

Although problems in these subtasks are still mostly NP-hard, the decomposition makes it possible to adopt existing approaches in relatively mature fields (including operations research and code generation) to design of heuristics for behavioral synthesis. Thus, the decomposition has been followed by numerous research efforts, and the resulting sequential flow has become a de facto standard in many behavioral synthesis tools. Resource-constrained scheduling and time-constrained scheduling are basic formulations of the scheduling problem, and they focus on the tradeoff between performance and resource usage. These formulations, after extensive study over the past twenty years, have reasonably good solutions (e.g., [4]–[7]). Approaches to resource binding usually either use the minimum possible number of functional units (FUs) and registers [8]–[10], or assume a given resource allocation. In either case, the resource binding step often focuses on minimizing multiplexers (MUXes) and connections [11]–[16].

Although each of these subtasks seems to be handled well, according to results reported in literature. Many techniques introduced for these subtasks do help in reducing the final area/cost, they may lead to inferior designs because all resources are not considered consistently and coordination between different steps is lacking. For example, scheduling may do a good job in minimizing the lower bound of the number of required FUs and registers, but it cannot predict the impact on interconnection without a complete binding solution. Binding has better models for resource estimation, but existing binding algorithms are often based on a given scheduling solution that is constructed without consideration of the binding techniques being used.

Considering the interdependency between the subtasks, it is natural to solve the problem altogether in a single formulation. Integer-linear programming (ILP) formulations for simultaneous allocation, scheduling and/or binding have been proposed [17], [18], but such formulations are not scalable to designs of a practical size in general. Iterative approaches between scheduling and binding have also been proposed to solve the problem more globally: interleaved scheduling/binding [19], incremental scheduling/binding [20], and Liapunov’s stability theorem [21]. These techniques generally either still have inconsistent optimization models in different steps or suffer from scalability problems.

In this paper we present a behavioral synthesis flow for resource optimization under latency and frequency constraints, in which all resources in the datapath are consistently optimized throughout the whole synthesis flow. This is achieved by the cooperation of *range-based binding*, *soft-constraint-based*

scheduling, and binding legalization. First, the range-based binding generates sharing intentions in favor of minimizing resource usage with the input of an unscheduled control-data flow graph (CDFG). The sharing intentions are transformed into soft constraints during scheduling, and the optimization engine in the scheduler tries to generate a scheduling solution that satisfies as many sharing intentions as possible. In this way, scheduling directly optimizes total resource usage guided by the accurate resource model that is available to binding. Finally, binding legalization is performed based on the valid scheduling solution. Because the sharing intentions are aggressively optimistic and may not be fully satisfied after scheduling, we introduce an iterative flow so that the sharing intentions are gradually imposed and generated over the entire optimization process.

II. MOTIVATION

The input to the behavioral synthesis tool is a CDFG, which can be generated from behavioral descriptions in high-level languages such as C/C++. In a CDFG, nodes represent operations, and edges represent dependencies (including data dependency, control dependency, and pseudo dependency). Data propagated along edges are called variables. In this paper we consider synchronous circuits with clocks, and assume that variables produced in one control step and used in another control step need to be stored in registers.

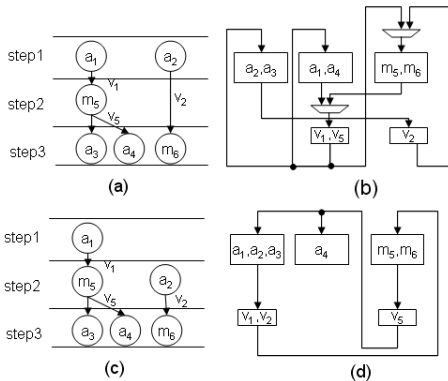


Fig. 1. (a) Scheduling solution 1, (b) Binding solution of (a), (c) Scheduling solution 2, (d) Binding solution of (c).

Fig. 1 shows two scheduling and binding solutions for the same CDFG under the same performance constraints. Operations labeled a_i are additions, and the ones labeled m_i are multiplications. Three variables, v_1 , v_2 , and v_5 , need to be stored in registers. Based on different schedules, both binding solutions use the minimum numbers of FUs and registers and introduce the minimum size of MUXes. The results show that both solutions need two adders, one multiplier and two registers, but the binding solution of (a), shown in Fig. 1(b), requires two additional 2-input MUXes. In terms of FUs and registers, scheduling solutions (a) and (c) are equally good. But considering the final resource usage after binding, schedule (c) clearly has better quality. This example shows that considering

only part of the components in a single step may restrict the solution space unwisely and lead to inferior designs. Thus, a better synthesis flow is needed.

III. OVERVIEW

The behavioral synthesis problem to be solved in this paper is described as follows.

Problem 1: Given: (1) A CDFG $G(O, E)$, where O is the set of operations, and E is the set of dependencies; (2) latency constraint; (3) clock frequency; (4) the hardware platform.

Objective: Perform scheduling, resource allocation and resource binding for the CDFG to minimize overall resource usage including FUs, registers and MUXes.

To minimize the overall resource usage (measured as total area in this paper), all types of components need to be considered altogether and optimized consistently throughout the whole synthesis flow. The resource-driven synthesis (RDS) flow proposed in this paper achieves this goal through the cooperation of range-based binding, soft-constraint-based scheduling, and final binding legalization. The RDS flow is shown in Fig. 2. First, the feasible scheduling range for each operation o in its basic block, $[ASAP(o), ALAP(o)]$, is calculated based on dependencies, target frequency and the latency constraint. In Step 2, the range-based binding (RBB) is performed to obtain an optimistic binding solution, which has a minimized total resource usage (including FUs, registers, and MUXes) but may not be valid — there may not be a valid scheduling solution satisfying all the sharing intentions. This is because RBB share operations/variables based on range compatibilities (described later) instead of traditional compatibilities decided by a valid scheduling solution.

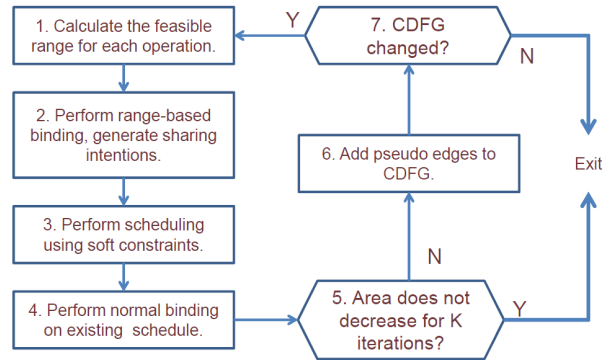


Fig. 2. The resource-driven synthesis flow.

Sharing intentions are derived from the resource-efficient but possibly invalid binding solution. Operations (variables) bound to the same resource are preferred to share a functional unit (register). In Step 3, the soft-constraint-based scheduling is performed to generate a valid scheduling, which not only honors all the data/control dependency in CDFG and performance constraints, but also takes the sharing intentions into consideration — in fact, the scheduler will try to meet as many of them as possible. In this way, the scheduler is guided by the preferred sharing solutions in favor of efficient

resource sharing and low interconnect complexity and the optimization goals in scheduling and binding are consistent. After scheduling, the final binding is performed to derive a valid binding solution.

Note that because the sharing intentions generated in Step 2 are based on range compatibilities, they are optimistic in some sense and may conflict with each other. To remedy this situation, we add in an iterative flow. Taking into consideration both the sharing intentions (Step 2) and the valid scheduling solution (Step 3), part of the satisfied sharing intentions are selected and pseudo edges are added to the CDFG to guarantee that these sharing intentions will always be satisfied in later iterations. These edges will influence the feasible range of every operation in the next iteration. As iterations continue, sharing intentions will have less conflicts and the solution given by range-based binding gradually converges to a feasible solution. When no more pseudo edges can be added at the end of an iteration, or the total cost does not decrease for a few iterations, the whole synthesis flow exits.

IV. RANGE-BASED BINDING

A. Problem Formulation and Definitions

The range-based binding problem is described as follows. **Problem 2:** Given: (1) A CDFG $G(O, E)$; (2) the range compatibility for operations and variables; (3) the hardware platform. **Objective:** Perform resource allocation and resource binding based on range compatibility, such that overall resource usage is minimized.

Range compatibility is a function defined for a pair of operations or a pair of variables in the CDFG. It can take one of the three values: *compatible*, *conflict*, and *undecided*. Two operations (variables) are range-compatible if they are always compatible in any feasible schedule, and they are range-conflict if they conflict (cannot share a component) in any feasible schedule; in all other cases, they are range-undecided.

For two operations o_i and o_j of the same functional type, we denote their range-based compatibility as $RC(o_i, o_j)$. $RC(o_i, o_j) = \textit{compatible}$ if any of the following conditions is satisfied.

- 1) $[ASAP(o_i), ALAP(o_i)] \cap [ASAP(o_j), ALAP(o_j)] = \emptyset$.
- 2) o_i and o_j are connected by a path in the CDFG, and the two operations cannot be chained in the same cycle.
- 3) o_i and o_j has mutually exclusive execution conditions (e.g., in different branches of an *if* or *case* statement).

When $ASAP(o_i)$, $ALAP(o_i)$, $ASAP(o_j)$, and $ALAP(o_j)$ are all equal and o_i and o_j are not under mutually exclusive conditions, $RC(o_i, o_j) = \textit{conflict}$, because in such a case o_i and o_j will always be scheduled to the same c-step. If o_i and o_j are of different types, they will always *conflict*. For all other cases, we define $RC(o_i, o_j)$ as *undecided*. Taking the CDFG

in Fig. 1 as an example, we have

$$\begin{aligned} RC(a_1, a_2) &= \textit{undecided} & RC(a_1, a_3) &= \textit{compatible} \\ RC(a_1, a_4) &= \textit{compatible} & RC(a_2, a_3) &= \textit{compatible} \\ RC(a_2, a_4) &= \textit{compatible} & RC(a_3, a_4) &= \textit{conflict} \\ RC(m_5, m_6) &= \textit{undecided}. \end{aligned}$$

Range compatibility can be defined similarly for variables. For a variable v , let $prd(v)$ be the operation producing v , and $cons(v)$ be the set of operations consuming v . We define $minlife(v) = [ALAP(prd(v)), \max\{ASAP(o) | o \in cons(v)\}]$, and $maxlife(v) = [ASAP(prd(v)), \max\{ALAP(o) | o \in cons(v)\}]$. For two variables v_i and v_j , $RC(v_i, v_j) = \textit{compatible}$ if any of the following conditions is satisfied.

- 1) $maxlife(v_i) \cap maxlife(v_j) = \emptyset$.
- 2) $\forall o \in cons(v_i)$, either $o = prd(v_j)$ or there is a path from o to $prd(v_j)$.
- 3) $\forall o \in cons(v_j)$, either $o = prd(v_i)$ or there is a path from o to $prd(v_i)$.
- 4) $prd(v_i)$ and $prd(v_j)$ are under mutually exclusive conditions.

$RC(v_i, v_j) = \textit{conflict}$ if any of the following conditions is satisfied.

- 1) $minlife(v_i) \cap minlife(v_j) \neq \emptyset$.
- 2) $cons(v_i) \cap cons(v_j) \neq \emptyset$.

The second condition means that if v_i and v_j have a common consuming operation, their lifetime will always overlap since they are both alive during the execution of the common operation. For all other cases, $RC(v_i, v_j) = \textit{undecided}$. For the example in Fig. 1, we have

$$\begin{aligned} RC(v_1, v_5) &= \textit{compatible} & RC(v_1, v_2) &= \textit{undecided} \\ RC(v_2, v_5) &= \textit{undecided} \end{aligned}$$

B. Overview of RBB Algorithm

Resource allocation decides how many FUs and registers are allocated, and resource binding decides how to bind operations to FUs and variables to registers. The range-based binding algorithm solves both tasks at the same time so that the trade-off among different types of resources is made globally and automatically. No predefined resource allocation is needed, and we do not stick to the solution with the minimum number of FUs and registers, which does not necessarily give the minimum overall resource usage.

The basic idea is that we explore the solution space and generate several intermediate binding solutions with decreasing resource allocation. A final resource allocation and binding will then be derived from these intermediate solutions. The generation of the intermediate solutions is adopted from [14], which breaks the interdependency between FU binding and register binding so that it performs better than previous work in terms of MUXes. The basic idea is to interleave FU binding and register binding in such a way that both tasks can access partial information from each other and do not have to sacrifice the accuracy of any task.

The resource allocation of the intermediate binding solutions is decided as follows. Initially the number of FUs of

type f is set as the total number of operations of type f in the CDFG, and the number of registers is set as the total number of variables. Then in each iteration, the number of FUs of each type will be decreased by some ratio $d\%$, where the value d controls the number of binding solutions are to be generated.

After intermediate resource allocation, FU binding and register binding are performed based on the previous binding solution, which has a larger resource allocation and thus less sharing of operations/variables. When no feasible solution can be found, we stop decreasing the resource allocation and a final resource allocation and binding is then derived.

C. Bipartite Binding

A binding algorithm using bipartite matching to reduce MUXes is proposed in [12]. In our work, we use a similar framework in an iterative manner. to solve both FU binding and register binding with range compatibility. Register binding is used to illustrate the algorithm. We first divide all variables into disjoint groups, G_1, G_2, \dots, G_n , such that variables in the same group conflict with each other and thus have to be stored in different registers. The grouping is formulated as a clique partitioning of the conflict graph of variables. These groups are then sorted in decreasing order of group size and bound to registers one group at a time. Assume the number of available registers is M and the current group to be bound is G_i . The bipartite graph is comprised of M nodes representing registers and another $|G_i|$ nodes representing variables. There is an edge between a register node r and a variable node v if and only if v is range-compatible or range-undecided with all variables already stored in r . The cost of edge (r, v) is defined as

$$\text{cost}(r, v) = \alpha \times C_{MUX} + \beta \times C_{feasibility},$$

where α and β are constants, C_{MUX} represents whether a new connection will be added due to the binding of v to r , and $C_{feasibility}$ is introduced to improve feasibility of the binding solution.

To estimate C_{MUX} in the bipartite matching proposed in [12], an FU binding solution is needed. This solution can be retrieved either from the current iteration if FU binding is performed prior to register binding, or from the previous iteration if register binding is performed first, except in the first iteration (where we use the algorithm in [15] to do initial binding).

$C_{feasibility}$ is calculated as $N_{undecided}/|V_r|$, where V_r is the set of variables already bound to r , and $N_{undecided}$ is the number of variables in V_r whose range compatibility with v is undecided. The lower $C_{feasibility}$ is, the more feasible the sharing decisions are. For the example in Fig. 1, assume v_1 is bound to r in the current partial binding solution, $C_{feasibility}$ in $\text{cost}(r, v_2)$ is $1/1 = 1$, since $RC(v_1, v_2) = \text{undecided}$. $C_{feasibility}$ in $\text{cost}(r, v_5)$ is $0/1 = 0$, since $RC(v_1, v_5) = \text{compatible}$.

D. Sharing Intentions

The binding solution generated by RBB is supposed to be good in terms of resource usage, because it is not restricted by a given schedule, and there are less conflicts.

Thus, it is preferred that the scheduler generates a solution based on which many sharing decisions (for both FUs and registers) by RBB are feasible. In the example in Fig. 1, if the range-based binding solution is Fig. 1(d), we have the following sharing intentions: $P_{FU}(a_1, a_2, a_3)$, $P_{FU}(m_5, m_6)$, and $P_{REG}(v_1, v_2)$, where P_{FU} and P_{REG} are sharing intentions for FUs and registers, respectively.

V. SCHEDULING WITH SOFT CONSTRAINTS

In this section we describe the scheduler used in the RDS flow. The intentions for resource sharing can be modeled as soft constraints in a natural way. A soft constraint is a constraint that is preferred to be satisfied but not necessarily. In [22], the problem of scheduling using soft constraints is formulated, and an efficient method for solving the problem is developed. The problem solved in our scheduler can be described as follows.

Problem 3: Given: (1) A CDFG $G(O, E)$; (2) latency constraint; (3) clock frequency; (4) the hardware platform; (5) sharing intentions of operations and variables.

Objective: Assign all operations to control steps, so that as many as possible sharing intentions are satisfied.

In our scheduler, we use a mathematical programming formulation to model various constraints and objectives, following [6], [22], [23]. First, for each operation o , we introduce a variable $s(o)$ to represent the starting time of o in the scheduling solution. The central task of scheduling is thus deciding $s(o)$ for every operation.

In [6], the authors introduced integer linear difference constraints, which have the form $s(o_i) - s(o_j) \leq d$, where d is an integer. Such a special form of constraint can be used to model a wide range of commonly encountered constraints in scheduling, including dependency, frequency, latency, relative timing, etc. However, sharing intentions are essentially soft constraints. They indicate favorable situations in the scheduling solution but are not necessarily satisfied. In [22], it is shown that an integer difference soft constraint can also be handled efficiently. An integer difference soft constraint is in the same form as an integer difference constraint, but it is soft and allows violation. In the objective function, each violation of a soft constraint is associated with a penalty. If the penalty is a convex function of the violation, the problem is solvable in polynomial time.

We will use P_{FU} as an example to illustrate how to add constraints for sharing intentions. Given a P_{FU} , the scheduler will try not to schedule any pair of operations in P_{FU} to the same control step, because otherwise a conflict will happen. When a linear order π of the operations is available as $\{o_1, \dots, o_m\}$, we can add soft constraints

$$s(o_i) - s(o_{i+1}) \leq -l, i = 1, \dots, m-1,$$

where l is the initiation interval of the corresponding functional unit. A quadratic cost will be added into the objective function when such a soft constraint is violated. The linear order can have a significant impact on the scheduling result. In our implementation, we use a heuristic to order the operations

according to their ALAP schedule, indicating that operations on the critical path should be performed first. As shown in [22], the above formulation can be solved efficiently. Interested readers may refer to [22] for details about the theoretical results and the implementation details of the solver.

VI. BINDING LEGALIZATION

Step 4 of Fig. 2 solves a traditional binding problem and generates a valid binding solution. It takes a valid scheduling as input and the compatibilities of operations/variables are derived from this scheduling. The algorithm of Step 4 is similar to RBB except that RBB uses range compatibilities to decide if two operations/variables are shareable, while Step 4 uses traditional compatibility based on a given schedule.

VII. DETERMINATION OF SHARING

In the scheduling solution, some sharing intentions are satisfied while others may not be. This section discusses Step 6 of Fig. 2, which selects part of the sharing intentions generated by RBB and makes them finalized.

Take variable sharing as example, assume r is one of the registers in B_{range} and the group of variables stored in r is $V_r = \{v_1, v_2, \dots, v_m\}$. We then build a conflict graph containing nodes for each variable in V_r . If v_i and v_j conflict in S , there will be an edge between the two nodes representing v_i and v_j . The set of variables in V_r that are compatible with each other forms an independent set of the conflict graph. When the CDFG is in the SSA form, the conflict graph is a chordal graph, whose maximum independent set can be computed in polynomial time [24]. We then sort the variables in the maximum independent set in the order of the time step in which they are produced. For each pair of adjacent variables, (v_i, v_j) , we change the CDFG by adding in directed pseudo edges from each operation in $cons(v_i)$ to $prod(v_j)$. These pseudo edges ensure that v_i and v_j are range-compatible in the following iterations. And they are shared in successive binding solutions before any other sharing is performed. Operation sharing are handled in a similar way.

VIII. EXPERIMENTAL RESULTS

The RDS algorithm is implemented in xPilot, a behavioral synthesis tool developed at UCLA [25]. In our experiment flow, a CDFG is generated from an optimized behavioral description in C. RDS is applied to the CDFG for area optimization under performance constraints. Our system outputs RTL model in VHDL, and is passed to the Xilinx ISE v9.1 for implementation on the Virtex4 FPGA platform. To make the total area of datapath consistent and easier to compare, we implement multipliers with LUTs and use the slice count to measure the resource usage.

A. Analysis of RDS

We first examine the iterative flow of RDS. Fig. 3 shows the total resource usage, including FUs, registers, and MUXes, by RBB and the binding legalization (BL) for the test case *lee*. We see that the gap between RBB and BL decreases along

the iterations, which means RBB is more accurate and closer to feasible solutions with the addition of pseudo edges. Fig. 4 lists the number of sharing intentions generated by RBB and the satisfied sharing intentions by the scheduler in different iterations for test case *lee*. Sharing intentions for FUs and registers are calculated separately. We consistently observe that more sharing intentions are feasible in later iterations.

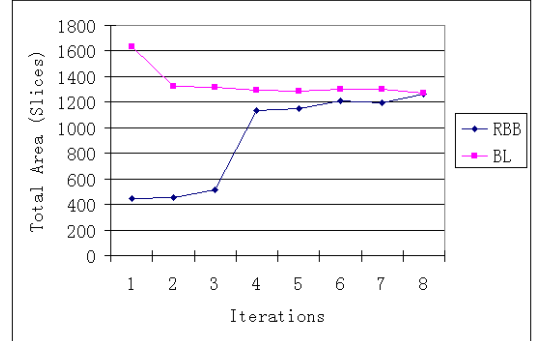


Fig. 3. Area variation of the solutions by RBB and BL.

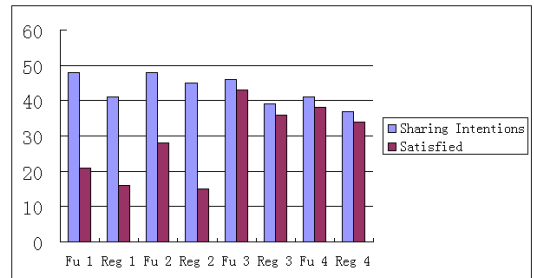


Fig. 4. Number of satisfied sharing intentions.

B. Comparison of QoR

To validate our proposed approach, we set up four experimental synthesis flows as follows.

- 1) *FB*: scheduling is done using force-directed scheduling with the goal of minimizing the number of FUs [4], and binding is done using bipartite matching [12].
- 2) *ILP*: scheduling is done using integer-linear programming with the goal of minimizing the number of FUs, and the binding algorithm is the same as the one in FB.
- 3) *C2V*: the behavioral synthesis tool for FPGA that is available on line [26].
- 4) *RDS*: the approach proposed in this paper.

Table I shows the resource distribution of FB and RDS, counted by the resource estimator implemented in xPilot. For each flow we show the number of multiplier, adders, registers, as well as multiplexers. Fig. 5 compares the total area in number of slices after place-and-route. RDS achieves an average (geometric mean) area reduction by 25%, 24% and 30% compared to FB, ILP and C2V, respectively. This shows the potential of consistent optimization in different steps.

Due to the iterative flow, RDS runs longer than FB and C2V (which usually take a few seconds), but it still runs in reasonable time. For the set of designs in the experiments, RDS finishes in five minutes. ILP is slow because of its enumerative nature, and it does not give a solution in 10 hours for two test cases in our experiment.

TABLE I
COMPONENT USAGE COMPARISON.

	FB				ILP				RDS			
	#a	#m	#r	#x	#a	#m	#r	#x	#a	#m	#r	#x
dir	5	5	35	126	4	3	36	127	4	4	26	116
honda	2	1	18	60	2	1	20	70	2	1	11	56
lee	6	3	30	72	3	2	22	72	5	2	15	70
mcm	7	4	43	109	4	2	37	117	8	2	30	118
matmul	3	5	40	48				timeout	2	4	20	44
cftmdl	9	7	73	111				timeout	11	6	21	95

#a, #m, #r, #x are the number of adders, multipliers, registers and multiplexers, respectively.

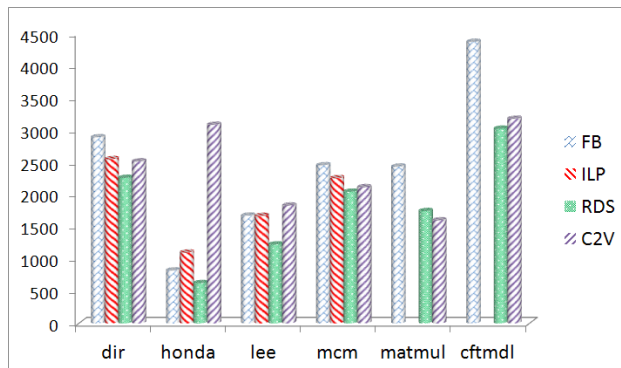


Fig. 5. Area comparison.

IX. CONCLUSION

In this paper we present a behavioral synthesis flow, in which all resources in the datapath are consistently optimized throughout the whole synthesis flow. This is achieved by guiding the scheduler using sharing intentions in favor of efficient resource usage. Experimental results show significant reduction compared to previous approaches in total area for a set of benchmarks. Our idea can also be used for optimizations of other objectives (e.g., power), by consistent consideration throughout different steps in the behavioral synthesis flow.

ACKNOWLEDGMENT

This work was supported by the Semiconductor Research Corporation under Contract 2009-TJ-1879.

REFERENCES

- [1] A. Agarwal, "Comparison of high level design methodologies for algorithmic IPs: Bluespec and C-based synthesis," Master's thesis, Massachusetts Institute of Technology, February 2009.
- [2] D. D. Gajski, N. D. Dutt, A. C.-H. Wu, and S. Y.-L. Lin, *High-level synthesis: introduction to chip and system design*. Kluwer Academic Publishers, 1992.
- [3] G. D. Micheli, *Synthesis and Optimization of Digital Circuits*. McGraw-Hill Higher Education, 1994.

- [4] P. G. Paulin and J. P. Knight, "Force-directed scheduling for the behavioral synthesis of ASICs," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 8, no. 6, pp. 661–679, 1989.
- [5] K. Kuchcinski, "Constraints-driven scheduling and resource assignment," *ACM Trans. on Design Automation of Electronics Systems*, vol. 8, no. 3, pp. 355–383, 2003.
- [6] J. Cong and Z. Zhang, "An efficient and versatile scheduling algorithm based on SDC formulation," in *Proc. Design Automation Conf.*, 2006, pp. 433–438.
- [7] S. Haynal and F. Brewer, "Automata-based symbolic scheduling for looping dfgs," *IEEE Trans. on Computers*, vol. 50, no. 3, pp. 250–267, 2001.
- [8] P. G. Paulin, J. P. Knight, and E. F. Girczyc, "HAL: a multi-paradigm approach to automatic data path synthesis," in *Proc. Design Automation Conf.*, 1986, pp. 263–270.
- [9] D. Springer and D. E. Thomas, "New methods for coloring and clique partitioning in data path allocation," *Integr. VLSI J.*, vol. 12, no. 3, pp. 267–292, 1991.
- [10] P. Brisk, F. Dabiri, R. Jafari, and M. Sarrafzadeh, "Optimal register sharing for high-level synthesis of SSA form programs," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 5, pp. 772–779, May 2006.
- [11] B. M. Pangre, "Splicer: a heuristic approach to connectivity binding," in *Proc. Design Automation Conf.*, 1988, pp. 536–541.
- [12] C.-Y. Huang, Y.-S. Chen, Y.-L. Lin, and Y.-C. Hsu, "Data path allocation based on bipartite weighted matching," in *Proc. Design Automation Conf.*, 1990, pp. 499–504.
- [13] D. Chen and J. Cong, "Register binding and port assignment for multiplexer optimization," in *Proc. Asia and South Pacific Design Automation Conf.*, 2004, pp. 68–73.
- [14] J. Cong and J. Xu, "Simultaneous FU and register binding based on network flow method," in *Proc. Design, Automation and Test in Europe*, 2008, pp. 1057–1062.
- [15] H. W. Zhu and C. C. Jong, "Interconnection optimization in data path allocation using minimal cost maximal flow algorithm," *Microelectronics Journal*, vol. 33, no. 9, pp. 749–759, 2002.
- [16] J. Cong, Y. Fan, and W. Jiang, "Platform-based resource binding using a distributed register-file microarchitecture," in *Proc. Int. Conf. on Computer Aided Design*, 2006, pp. 709–715.
- [17] C. H. Gebotys and M. Elmasry, "Global optimization approach for architectural synthesis," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, no. 9, pp. 1266–1278, Sep. 1993.
- [18] S. Chaudhuri, S. Blthye, and R. Walker, "A solution methodology for exact design space exploration in a three-dimensional design space," *IEEE Trans. on Very Large Scale Integration Systems*, vol. 5, no. 1, pp. 69–81, Mar. 1997.
- [19] M. Balakrishnan and P. Marwedel, "Integrated scheduling and binding: A synthesis approach for design space exploration," in *Proc. Design Automation Conf.*, Jun. 1989, pp. 68–74.
- [20] J. Cong, Y. Fan, G. Han, X. Yang, and Z. Zhang, "Architecture and synthesis for on-chip multicycle communication," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 4, pp. 550–564, April 2004.
- [21] M. Nourani and C. Papachristou, "Move frame scheduling and mixed scheduling-allocation for the automated synthesis of digital systems," in *Proc. Design Automation Conf.*, 1992, pp. 99–105.
- [22] J. Cong, B. Liu, and Z. Zhang, "Scheduling with soft constraints," in *Proc. Int. Conf. on Computer Aided Design*, 2009, pp. 47–54.
- [23] W. Jiang, Z. Zhang, M. Potkonjak, and J. Cong, "Scheduling with integer delay budgeting for low-power optimization," in *Proc. Asia and South Pacific Design Automation Conf.*, Jan. 2008, pp. 22–27.
- [24] S. Hack and G. Goos, "Optimal register allocation for ssa-form programs in polynomial time," *Information Processing Letters*, vol. 98, no. 4, pp. 150–155, May 2006.
- [25] J. Cong, Y. Fan, G. Han, W. Jiang, and Z. Zhang, "Platform-based behavior-level and system-level synthesis," in *Proc. IEEE Int. SOC Conf.*, Sep. 2006, pp. 199–202.
- [26] <http://www.c-to-verilog.com>.