Incremental Exploration of the Combined Physical and Behavioral Design Space [*]

Zhenyu (Peter) Gu, Jia Wang, Robert P. Dick, Hai Zhou
Northwestern University
2145 Sheridan Road
Evanston, IL, USA
{zgu646, jwa112, dickrp, haizhou}@ece.northwestern.edu

ABSTRACT

Achieving design closure is one of the biggest headaches for modern VLSI designers. This problem is exacerbated by high-level design automation tools that ignore increasingly important factors such as the impact of interconnect on the area and power consumption of integrated circuits. Bringing physical information up into the logic level or even behavioral-level stages of system design is essential to solve this problem. In this paper, we present an incremental floorplanning high-level synthesis system. This system integrates high-level and physical design algorithms to concurrently improve a system's schedule, resource binding, and floorplan, thereby allowing the incremental exploration of the combined behavioral-level and physical-level design space. Compared with previous approaches that repeatedly call loosely coupled floorplanners for physical estimation, this approach has the benefit of efficiency, stability, and better quality of results. For designs containing functional units with non-unity aspect ratios, the average CPU time improved by 369 %, the area improved by 14.24 %, and power improved by 4 %.

Categories and Subject Descriptors

J.6 [**Computer-Aided Engineering**]: Computer-Aided Design; B.8.2 [**Hardware**]: Performance and Reliability—*Performance Analysis and Design Aids*

General Terms: Design, Algorithms

Keywords: High-level Synthesis, Incremental, Floorplan

1. INTRODUCTION

Process scaling has enabled the production of integrated circuits (ICs) with millions of transistors. This has enabled increasingly full-featured and high-performance ICs. However, these capabilities have come at a cost. In order to deal with increased design complexity and size, it has been necessary to automate the higher levels of the design process. Unfortunately, considering interconnect dramatically complicates the synthesis problem. Although an abstract high-level description of the system is sufficient to predict logic delay and power consumption [1–4], detailed physical information, i.e., a floorplan, is necessary to make similar predictions about interconnect.

A number of researchers have considered the impact of physical details, e.g., floorplanning information, on high-level synthesis [5–8]. Interconnect and interconnect buffers are now first-order timing and power considerations in VLSI design [9]. This change has complicated both design and synthesis. It is no longer possible to accurately predict the power consumption and performance of a design without first knowing enough about its floorplan to predict the structure of its interconnect. For this reason, a number of researchers have worked on interconnect-aware high-level synthesis algorithms [10–13]. These approaches typically use a loosely coupled independent floorplanner for physical estimation. There are two drawbacks to this approach. First, the independent floorplanner may not be stable, i.e., a small change in the input netlist may result in a totally different floorplan. This results in a high-level synthesis algorithm that bases its moves on cost functions without continuity. Second, even if the floorplanner is stable, creating a floorplan from scratch for each high-level synthesis move is not efficient, given the fact that the new floorplan has only small difference from the previous one. New techniques for tightly coupling behavioral and physical synthesis that dramatically improve their combined performance and quality are now necessary.

Incremental automated design allows a tighter relationship between high-level synthesis and physical design, improving the quality of each [14]. A number of high-level synthesis algorithms are based on incremental optimization and are, therefore, amenable to integration with incremental physical design algorithms. This has the potential of improving both quality and performance. Incremental methods improve quality of results by maintaining important physical-level properties across consecutive physical estimations during synthesis. Moreover, they shorten CPU time by reusing and building upon high-quality, previous physical design solutions that required a huge amount of time and effort to produce.

In this paper, an incremental high-level synthesis system is proposed to reduce synthesis time dramatically while producing ICs with better area and power consumption. The benefits of this approach increase with increasing problem size and complexity. Our work is based on the interconnect-aware high-level synthesis tool, ISCALP [12], which was based on the low-power data path synthesis tool, SCALP [1]. We reuse the power models and iterative-improvement high-level synthesis framework from ISCALP. However, this work differs from previous work because it uses a truly in-

[*]This work is supported by the NSF in part under award CCR-0347941 and in part under award CCR-0238484.

cremental floorplanner to estimate interconnect structure [15]. Previous work relied on a fast constructive algorithm. Moreover, the high-level synthesis algorithm itself is made incremental. This results in substantial improvements in CPU time and solution quality for large problem instances. As shown in Section 4, quality of results improves by an average of 14.24 % for area on non-trivial behavioral specification, i.e., specifications with more than 30 operations. Moreover, CPU time improves by an average of 369 % when compared with similar non-incremental techniques. Even when compared with high-level synthesis algorithms that use carefully tuned fast constructive floorplanners as in ISCALP, we still observe an average improvement of 12.5 % in area for non-trivial specifications and better CPU time for large benchmarks with 169 operations.

This paper is organized as follows. Section 2 describes the design flow for the proposed high-level synthesis system. Section 3 contains a detailed description of our incremental floorplanner. Experimental results are presented in Section 4 and we present conclusions and future work in Section 5.

2. INCREMENTAL HIGH-LEVEL SYNTHESIS

This section presents definitions useful in the discussion of high-level synthesis and describe our incremental floorplanning high-level synthesis algorithm, called IFP-HLS.

2.1 Definition

The input to IFP-HLS is a control data-flow graph (CDFG), G, an input sampling period, T_s, and a library, L, of components to use for implementing the data path. IFP-HLS produces an RTL circuit in which power consumption and estimated area are optimized. IFP-HLS algorithm has two loops.

Given the supply voltage, in the outer loop the number of control steps, *csteps*, changes from its maximum value to its minimum value, where *csteps* is defined as

$$csteps = T_s \times f \tag{1}$$

or alternatively,

$$T_{clock} = T_s / csteps \tag{2}$$

In above equations, sample period T_s is the constraint on the input data rate. The system must be able to process an input sample before the next one arrives. For the given design, the sample period is a constant number. Hence, *csteps* indicates the number of clock cycles required to process an input sample. The variable f is the system clock frequency. T_{clock} is the system clock period.

Given *csteps*, which allows the clock period to be determined, the inner loop first uses the fastest available functional unit from the library to implement each operation. An as-soon-as-possible (ASAP) schedule is then generated for the initial solution to determine whether it meets the timing requirements. The initial solution must be further optimized. Having obtained an initial solution that meets the sample period constraint for the current value of *csteps*, the iterative improvement phase attempts to improve the architecture by reducing the switched capacitance while satisfying the sample period constraints. Additional details can be found in the literature [1], [12] .

2.2 Incremental High-level Synthesis Framework

In this section, we describe our incremental high-level synthesis tool, IFP-HLS. IFP-HLS is built upon ISCALP [12]. However, incorporating incremental floorplanning required substantial changes to that algorithm, resulting in a new low-power, incremental floorplanning high-level synthesis algorithm. IFP-HLS considers both data path and interconnect power consumption. It uses a new, incremental, method of improving functional unit binding during high-level synthesis.

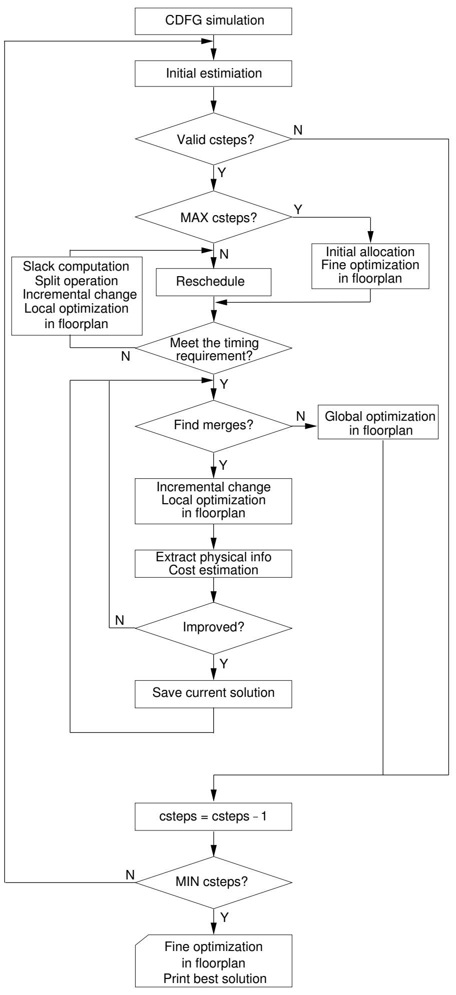


Figure 1: Incremental High-Level Synthesis Algorithm

The flowchart of the incremental high-level synthesis is shown in Fig. 1. IFP-HLS differs from ISCALP in a number of ways. Instead of generating an initial solution for each value of *csteps*, IFP-HLS only generates one solution at the maximum value of *csteps* and incrementally changes the solution as *csteps* decreases. Thus, in addition to using incremental floorplanning, IFP-HLS also eliminates redundant operations by taking advantages of incremental steps in high-level synthesis. Initially, we still use an ASAP schedule and fully parallel allocation to estimate whether valid solution exists for the current value of *csteps*. If not, further operations are not necessary for the current value of *csteps* because a binding that further reduces the finish time of an ASAP schedule is not possible. However, if an ASAP schedule meeting the timing requirements is possible, we start from the final solution for the previous value of *csteps* and reschedule it based on the current value, i.e., the previous *csteps* minus 1. If, after rescheduling, the solution meets its timing requirement, rebinding is not necessary. Otherwise, it is necessary to rebind some of the tasks and use parallel execution to improve performance. A *split operation* is used to eliminate resource contention by splitting a pair of tasks that were initially assigned to the same functional unit onto separate functional units. A detailed description of the split operation may be found in Section 2.3.

For a given value of *csteps*, when a move is chosen, we incrementally change the floorplan to see whether the change improves solution quality. If so, the change is accepted. Otherwise, the change is rejected and other moves are attempted. This technique differs from that in the original ISCALP. In that algorithm, floorplanning was only done at the end of each *csteps* iteration; i.e., the algorithm did not take advantage of solution correlation to save effort across *csteps* values.

A high-quality incremental floorplan was developed and incorporated into IFP-HLS. Each time the high-level synthesis algorithm needs physical information to guide its moves, it extracts that information from the current, incrementally generated, floorplan. Moreover, costs derived from the floorplan are also used to guide high-level synthesis moves. By using incremental floorplanning, closer interaction between high-level synthesis physical design is possible, i.e., the high-level synthesis algorithm may determine the impact of potential changes to binding upon physical attributes such as interconnect power consumption and area.

In summary, IFP-HLS performs scheduling/allocation/binding by iteratively changing *csteps*, determining whether operations need to be rescheduled or re-bound (split) in order to meet timing constraints. At each step the floorplanner is updated.

2.3 Extended Operation

In this subsection, we explain the split operation in detail. In addition, we describe rescheduling and a new graph technique to determine split locations.

First, we make some observations on the iterative improvement algorithm. Since each time *csteps* decreases by one, each individual operation takes, at most, the same number of control steps as it did for the previous value of *csteps*. Given that *csteps* is no less than the previous *csteps* minus one, we can conclude that the optimal schedule for the previous value of *csteps* violates the deadline for the current value of *csteps* by, at most, one clock cycle. We will use slack, S, to represent this information, which is defined as follows:

$$S = LST - EST \tag{3}$$

Here, *EST* is the earliest start time and *LST* is the latest start time. These values were computed by the ASAP and ALAP accordingly. Notice that the calculations consider the sharing of resources.

Nodes with slack values greater than 0 do not result in timing violations. However, nodes with slack values of -1 cause timing violations, i.e., they must be executed one cycle earlier. These timing violations can be removed by splitting merged operations which, although useful for previous values of *csteps*, now harm performance. Based on this observation, the split operation is used to eliminate timing violations. Therefore, the whole high-level synthesis algorithm is implemented in an incremental way from maximum to minimum values of *csteps* without rebinding from scratch at each value of *csteps*. Few changes to binding and scheduling are required as a result of single-unit change to *csteps*. However, in order to meet timing requirements, it is sometimes necessary to split operators mapped to the same functional unit. The split operation makes it possible quickly apply these isolated changes. Previous high-level synthesis systems, e.g., SCALP and ISCALP, started from a fully parallel implementation for each value of *csteps* and repeatedly merged operators to reduce area. Although both techniques are reasonable in the absence of an integrated floorplanner, the incremental approach used in IFP-HLS speeds optimization (without degrading solution quality) by requiring far fewer changes to the floorplan. Generally, the split and merge operations, combined, allow complete exploration of the solution space. However, the primary goal, when changing the number of control steps, is meeting timing constraints. We therefore focus our exploration of the solution space on the most promising region by iteratively splitting functional units on the critical timing path. In summary, completeness is easy to achieve but inefficient in practice compared with the proposed critical path based algorithm.

We will give an example to further describe the split operations. Consider the data flow graph shown in Fig. 2(a), in which arrows represent the data dependencies. Scheduling and allocation yield the CDFG in Fig. 2(b). Here, we can see that three functional units (FUs) are used. Tasks A, B, and C share FU1, tasks D and E share FU2, and task F uses FU3. When *csteps* is reduced from 3 to 2, instead of initializing binding and scheduling from scratch, the algorithm reschedules it based on current binding. For this example, after rescheduling, there is still a timing violation for tasks A, B, and C because they were all bound to FU1. Therefore, the split operation is necessary in order to allow all tasks to meet their timing requirements. The following steps are used to do the split operation.

Based on the result of slack computation, we produce a graph including all the tasks with negative slack. Each task is represented by a node. In addition, there are three kind of edges, defined as follows:

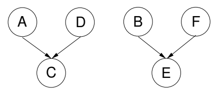
1. Data dependency edges, indicating that the destination node takes the source node's data as input;
2. Merge edges, indicating that the two nodes are bound to the same functional unit or the same storage unit; and
3. Pseudo edges, used to restructure the graph for application of the min-cut algorithm. A pseudo source node and pseudo sink node are introduced to the graph. All input nodes are connected to the pseudo source node and all output nodes are connected to the pseudo sink node.

After constructing this graph, the min-cut algorithm is executed. First, an infinite weight is assigned to all pseudo edges and data dependency edges. Merge edges are each given weights of one. If two nodes are connected by both a data dependency edge and a merge edge, the merge edge is eliminated because split operations on nodes sharing dependency edges do not improve the timing properties. Using the min-cut algorithm in this manner splits a minimal cardinality subset of nodes, allowing a reduction in the finish time of the ASAP schedule.

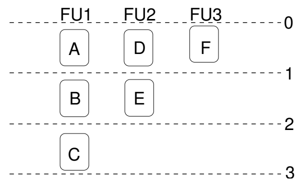
Although decrementing *csteps* may increase delay by at most one clock cycle, there may be some value of *csteps* for which even fully parallel bindings do not allow an ASAP schedule to meet its timing constraints. Therefore, min-cut and rescheduling may not be carried out for some values of *csteps*. After the split operation, the tasks are rescheduled and slack is recomputed to determine whether timing constraints are met.

Fig. 2(c) illustrates a merge graph. The dashed lines represent merge edges and the solid lines represent pseudo-edges. For this example, it is possible to cut through either A and B, or B and C. Here, we cut through A and B, thereby assigning A to a new functional unit, FU4. B and C remain bound to the original functional unit, FU1. As shown in Fig. 2(d), all tasks now meet their timing constraints.

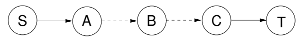
Another case need also be considerated. If no valid solutions exist for the current value of *csteps*, IFP-HLS will skip further optimization and decrement *csteps*. IFP-HLS may reach a valid value of *csteps* after skipping several *csteps* values. In this case,



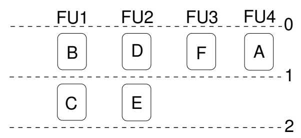
(a) CDFG



(b) Scheduling and Allocation



(c) Slack Computation



(d) After Split Operation

Figure 2: Incremental Changes on HLS

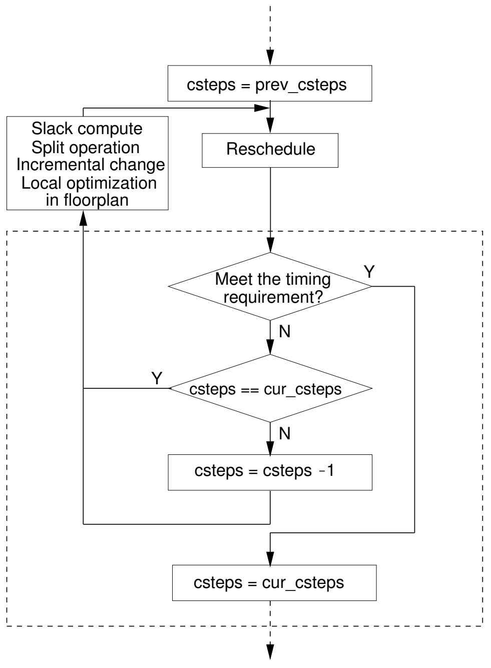


Figure 3: Iterative Split Operation for Slack Smaller than -1

the slack values for some tasks may less than -1. We use the following algorithm to cope with this case, as illustrated in Fig. 3. The value of *csteps* is decremented and the split operation, followed by rescheduling, are repeated until valid solution is produced.

3. INCREMENTAL FLOORPLANNING

As discussed in previous sections, in order to introduce incremental combined behavioral and physical optimization into high-level synthesis, a high-quality incremental floorplanner is necessary. We have tested this idea by building an incremental simulated annealing floorplanner into the IFP-HLS algorithm. In this section, we describe this incremental floorplanner.

This floorplanner handles blocks with different aspect ratios and produces non-slicing floorplans. However, unlike the network partitioning approach used in ISCALP, it was designed primarily for quality, not speed. Although the impact on synthesis time would prevent incorporation of a conventional high-quality floorplanner in the inner loop of a high-level synthesis system, incremental floorplanning enables both high quality and low synthesis time. High-level synthesis moves typically remove a single module or split a module into two. Therefore, many changes are small and their effects on the floorplan are mostly local. We reuse the previous floorplan as a starting point for each new floorplan. The previous floorplan was already high-quality. Therefore, re-optimization of the current floorplan to incorporate local changes is fast. In practice, we have found that this technique leads to quality-of-results and performance improvements over constructive floorplanning, even when compared with a very fast constructive floorplanner.

3.1 Floorplan Representation

The Adjacent Constraint Graph (ACG) floorplan representation is used within IFP-HLS's incremental floorplanner. This representation is described in detail in another publication [15], but will be summarized here. An ACG is a constraint graph with exactly one geometric relationship between every pair of modules. ACGs have invariant structural properties that allow the number of edges in the graph to be bounded. Operations on ACGs have straightforward meanings in physical space and change graph topology locally; they require few, if any, global changes. The operations of removing and splitting modules are designed to reflect high-level binding decisions. To obtain the physical position of each module, packing based on longest path computation is employed.

Although initial floorplan optimization is done with simulated annealing, re-optimization requires fewer global changes and less hill climbing. Moreover, perturbations resulting from high temperatures may disrupt high-quality floorplan structures. Therefore, it is reasonable to use lower temperatures for re-optimization. In practice, we have found that using a temperature of zero results in good quality and performance when re-optimizing; in other words, although simulated annealing is necessary during initial global floorplan optimization, using a greedy iterative improvement algorithm during re-optimization produced a good trade-off between quality and CPU time.

3.2 Incremental Floorplan Operation

The details of our approach follow. First, after generating the first ASAP schedule, we have an initial set of modules and interconnections. Simulated annealing is used to obtain an initial floorplan. Since every interconnect net has exactly one driving module, multi-pin nets are broken into two-pin wires with the driving module as the source. The wire length is calculated as the Manhattan distance between the two modules connected by the wire. At this point, the unit-length switched capacitances of data transfers between two modules are available. We use these as weights for the wire lengths. The weighted total wire length is related to power consumption, i.e., optimizing weighted wire length minimizes interconnect power consumption. A weighted sum of the area and the interconnect power consumption is calculated for use as the floorplanner's cost function, e.g.,

$$A + w \sum_{e \in E} C_e D_e \qquad (4)$$

where A is the area, w is the power consumption weight, E is the set of all wires, e is an interconnect wire, C_e is the unit-length switched capacitance for the data transfer along e, and D_e is the length of e. With this approach, we optimize the floorplan for both the interconnect power consumption and the area. The resulting floorplan will be improved during the consecutive incremental floorplanning high-level synthesis moves. Therefore, the number of simulated annealing iterations is bounded to reduce synthesis time.

After each high-level synthesis move, the previous floorplan is modified by removing or splitting a module. The modules and switched capacitances are updated based upon the impact of these merges and splits. The floorplan is then re-optimized with a greedy iterative improvement algorithm using the same cost function as the simulated annealing algorithm. The greedy improvements are divided into consecutive rounds. In every round we apply the same number of perturbations to the floorplan. If less than 10% of the perturbations result in reduced costs, we conclude that we have approached a local minimum and stop the greedy improvement algorithm. Although it would be easy to use a low simulated annealing temperature to allow some hill climbing during re-optimization, this was not necessary during re-optimization in practice.

When we find the best binding for a given value of *csteps*, we do floorplanning again and compare it with the best floorplan from the previous value of *csteps*. This time, non-zero temperature simulated annealing is used because it increases the accuracy of estimations. These normal simulated annealing runs occur only once per *csteps* value, allowing their time cost to be amortized. After determining the best binding across all the possible values of *csteps*, another simulated annealing floorplanning run is carried out for that binding. This final floorplanning stage occurs only once for every synthesis run. Therefore, it is acceptable to use a slower (but higher-quality) annealing schedule than those in the inner loop of high-level synthesis, thereby improving integrated circuit area and interconnect power consumption.

During the annealing schedule, we use a constant cooling factor, r, that is,

$$T' = r \times T \quad (5)$$

where T is the current temperature and T' is the temperature during the next iteration. The number of the perturbations for the initial floorplanning run, the floorplanning for each value of *csteps* run, and the final floorplanning run can be normalized to 1 : 2 : 20. The number of perturbations per round for the greedy iterative improvement algorithm is the same as that for final floorplanning run.

4. EXPERIMENTAL RESULTS

In this section, we present experimental results for the IFP-HLS incremental floorplanning high-level synthesis algorithm described in Sections 2 and 3. The results generated by ISCALP and IFP-HLS are compared. As explained in Section 2.2, both approaches optimize area and power consumption. The circuits described in this section were synthesized using a register transfer level (RTL) design library based on NEC's $0.25\,\mu$m process. The experiments were conducted on AMD Athlon-based linux workstations with 512 MB–1 GB of random access memory.

4.1 Benchmarks

We evaluated fifteen high-level synthesis benchmarks using a $0.25\,\mu$m technology library. *Chemical* and *IIR77* are infinite impulse response (IIR) filters used in industry. *DCT_IJPEG* is the Independent JPEG Group's implementation of discrete cosine transform (DCT). *DCT_Wang* is a DCT algorithm named after the inventor. Both DCT algorithms work on 8×8 pixel arrays. *Elliptic*, an elliptic wave filter, comes from the NCSU CBL high-level synthesis benchmark suite [16]. *Jacobi* is the Jacobi iterative algorithm for solving a fourth order linear system. *WDF* is a finite impulse response (FIR) wave digital filter. The largest benchmark, Jacobi, has 24 multiplications, 8 divisions, 8 additions, and 16 subtractions. In addition, we generated three CDFGs using a pseudo-random graph generator [17]. Random100 has 20 additions, 15 subtractions, and 19 multiplications. Random200 has 39 additions, 44 subtractions, and 36 multiplications. Random300 has 59 additions, 58 subtractions, and 72 multiplications.

Currently, we don't consider wire delays because we use $0.25\,\mu$m technology for which buffered and sized wire delay [9] is only 0.45 ns for 1 cm wires. Functional units delay ranges from 13.2 ns to 47.8 ns. We also checked a $0.18\,\mu$m process, for the wire delay is only 0.39 ns for 1 cm wires. This is also very small compared with the delay of functional units. Hence, the wire length is only used with unit-length switched capacitances to obtain interconnect power consumption (however, please see Section 5). Here, we use the wire capacitances from Cong's paper [9] along with an RTL design library from NEC to estimate the power consumption for the benchmarks.

4.2 Results

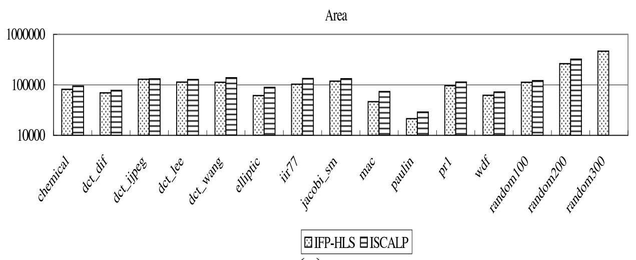
The results of running ISCALP and IFP-HLS on non-unity aspect ratio functional units are shown in Fig. 4. As shown in the table, IFP-HLS enable an average of 14 % improvement in area, 4 % improvement in power consumption, 172 % reduction in the number of merge operations, and 369 % improvement in CPU time.

We later used a unity aspect ratio for the functional units in both ISCALP and IFP-HLS. The IFP-HLS algorithm enable an average of 12 % improvement in area, 7 % improvement in power consumption, and 100 % reduction in the number of merge operation. Although IFP-HLS has a higher CPU time for small benchmarks, synthesis time was smaller than ISCALP for large benchmarks (Jacobi and random300). For random300, IFP-HLS required 60 minutes and 11 seconds to finish. ISCALP was not able to finish within 6 hours.

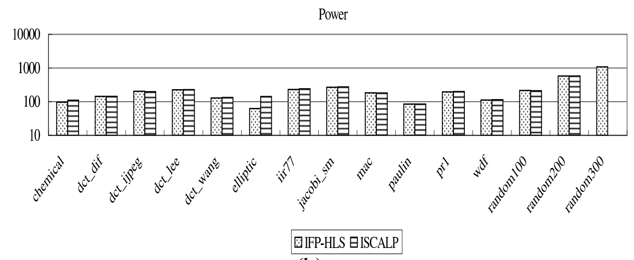
The reported power is calculated by summing the power consumptions of the functional units, multiplexors, registers (from the library), and interconnect (from the floorplan). We compared the power consumptions of our designs with those produced by an existing high-level synthesis algorithm (ISCALP) coupled with a constructive floorplanner. For the set of benchmarks with unity aspect ratio functional unit, the interconnect power decreased by 28.7 % and functional unit power is decreased by 4 %; i.e., it is possible to significantly decrease interconnect power consumption without increasing functional unit power consumption. Moreover, since both IFP-HLS and ISCALP were already designed to minimize power consumption, it is notable that we were still able to reduce power consumption while, at the same time, dramatically reducing CPU time for large problem instances.

5. CONCLUSIONS AND FUTURE WORK

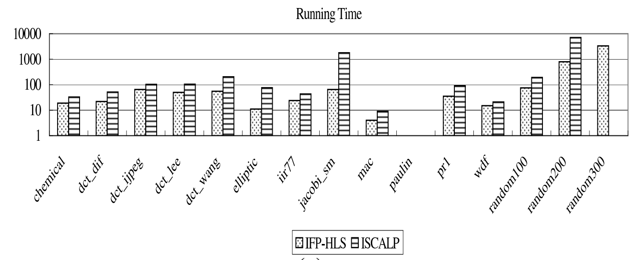
In this paper, we presented an incremental floorplanning, high-level synthesis system that integrates high-level and physical design algorithms to concurrently improve a system's schedule, resource binding, and floorplan. Compared with previous approaches that repeatedly call loosely coupled floorplanners for physical estimation, this work makes the following contributions: 1) it extends this system to support tight integration with physical design algorithms; and 2) it builds a novel incremental high-level synthesis algorithm into the resulting framework. This approach has the benefit of efficiency, stability, and better quality results. As shown in Section 4,



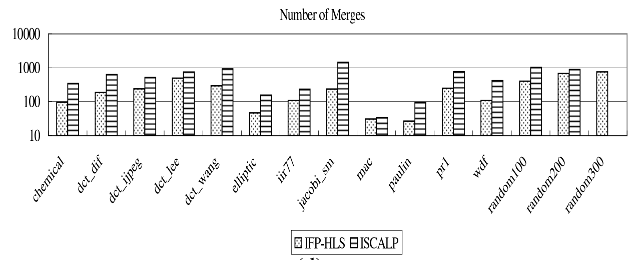
(a)



(b)



(c)



(d)

Figure 4: Experimental Result

the proposed incremental high-level synthesis algorithm allows improvement in area and power consumption while dramatically decreasing CPU time especially for big benchmarks.

Interconnect delay is becoming increasingly important. Therefore, our future work will focus on delay-driven incremental high-level synthesis for 0.13 μm and finer processes.

6. ACKNOWLEDGMENTS

We would like to thank Prof. Niraj Jha's research group at Princeton University for access to the ISCALP and NEC for access to their 0.25 μm technology library. We also would like to thank Yongpan Liu, Dr. Anand Raghunathan, and Dr. Lin Zhong for their helpful suggestions.

7. REFERENCES

[1] A. Raghunathan and N. K. Jha, "SCALP: An iterative-improvement-based low-power data path synthesis system," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, no. 11, pp. 1260–1277, Nov. 1997.

[2] A. P. Chandrakasan, *et al.*, "Optimizing power using transformations," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 1, pp. 12–51, Nov. 1997.

[3] R. S. Martin and J. P. Knight, "Power profiler: Optimizing ASICs power consumption at the behavioral level," in *Proc. Design Automation Conf.*, June 1995.

[4] K. S. Khouri, G. Lakshminarayana, and N. K. Jha, "High-level synthesis of low power control-flow intensive circuits," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 12, pp. 1715–1729, Dec. 1999.

[5] D. W. Knapp, "Fasolt: A program for feedback-driven data-path optimization," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 11, no. 6, pp. 677–695, June 1992.

[6] J. P. Weng and A. C. Parker, "3D scheduling: High-level synthesis with floorplanning," in *Proc. Design Automation Conf.*, June 1992.

[7] Y. M. Fang and D. F. Wong, "Simultaneous functional-unit binding and floorplanning," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1994.

[8] W. E. Dougherty and D. E. Thomas, "Unifying behavioral synthesis and physical design," in *Proc. Design Automation Conf.*, June 2000.

[9] J. Cong and Z. Pan, "Interconnect performance estimation models for design planning," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, pp. 739–752, June 2001.

[10] R. Mehra, L. M. Guerra, and J. M. Rabaey, "Low power architecture synthesis and impact of exploiting locality," *J. VLSI Signal Processing*, vol. 13, no. 8, pp. 877–888, Aug. 1996.

[11] P. Prabhakaran and P. Banerjee, "Simultaneous scheduling, binding and floorplanning high-level synthesis," in *Proc. Int. Conf. VLSI Design*, Jan. 1998.

[12] L. Zhong and N. K. Jha, "Interconnect-aware high-level synthesis for low power," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 2002.

[13] A. Stammermann, *et al.*, "Binding, allocation and floorplanning in low power high-level synthesis," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 2003.

[14] O. Coudert, *et al.*, "Incremental CAD," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 2000, pp. 236–244.

[15] H. Zhou and J. Wang, "ACG–Adjacent Constraint Graph for General Floorplans," in *Proc. Int. Conf. Computer Design*, Oct. 2004.

[16] "NCSU CBL," www.cbl.ncsu.edu/benchmarks/.

[17] R. P. Dick, D. L. Rhodes, and W. Wolf, "TGFF: Task graphs for free," in *Proc. Int. Wkshp. Hardware/Software Co-Design*, Mar. 1998, pp. 97–101.