VLSI DESIGN
1997, Vol. 5, No. 2, pp. 211–221
Reprints available directly from the publisher
Photocopying permitted by license only

Layout Modeling and Design Space Exploration in PSS1 System

FUR-SHING TSAI and YU-CHIN HSU*

Department of Computer Science, University of California, Riverside, CA 92521

This paper presents the design methodology used in PSS1, a high level synthesis system designed for computation dominated applications. It includes a behavior synthesizer and an area optimizer. Based on a pre-defined architecture, the behavior synthesizer translates a description into a number of designs with different delays and hardware costs. Based on a two-level layout model, the area optimizer fine-tunes the physical design using the information feedback from the layout tools. All the tools are linked by an X-window interface in which users can traverse among different tools and interactively change the design parameters. The output is linked to Lager system [7], a silicon assembler. The layout model allows a designer to interactively merge/split modules, change the shape of modules, and define the pin positions of modules. Experiments show that a considerable area improvement has been achieved using this methodology.

Keywords: High level synthesis, layout model, physical design

1. INTRODUCTION

A high level synthesis system [1–6] takes a behavior description as input and produces a structure description in register transfer level as its output. It can be used to provide a designer with a set of designs with different area and time requirements in a very short time. Based on the information provided by the synthesis system, a designer can choose the best path for his design.

The synthesis process is very complicated and is usually comprised of a series of transformations and optimizations. It is usually divided into a set of subtasks including scheduling, allocation, and control synthesis. Since the overall objective is to minimize the silicon area, a design methodology should also take the layout area into consideration.

The specific attributes which affect the silicon area in the physical design level include the availability of modules from a module library, the functionality, area and delay of each module, the shape and I/O pin position of each module. The physical design tools, including tools for floorplanning, placement and routing, also affect the quality of a design. Some of these factors are very difficult to be taken into account during the synthesis process. How to effectively utilize the layout information and at the same time prohibit a dramatical increment of the problem complexity is of particular interest to us. In our approach, we have

*Corresponding author. Tel.: 909-787-4406. Fax: 909-787-4643.

divided the optimization task into two phases: behavioral synthesis and area optimization. During the behavioral synthesis, the cost of a design is evaluated based on the summation of area of the modules and the number of interconnections. While during the area optimization phase, the shape and I/O pin positions of each module and the characteristics of the layout tools are taken into account.

In this paper, we present the methodology used in PSS1. The paper is organized as follows. We will introduce the system structure in Section 2. The tools used in the behavioral synthesizer are described in Section 3. In Section 4, we discuss the area optimizer. Experiments are presented in Section 5, and finally concluding remarks are made in Section 6.

2. SYSTEM STRUCTURE

PSS1 is a high level synthesis system which is tailored toward the designs for computation dominated applications. Fig. 1 shows the structure of the system. The design information (e.g., behavior description, cost/performance constraints) and designer experiences (e.g., size of memory module, interconnection between modules) are described in two different input files. Through the X-window interface, designers can manually modify the design specification based on the information feedback from the synthesizer and the layout tools. The design process can be iterated many times and at many stages until a high quality design is obtained. The synthesis tool box consists of

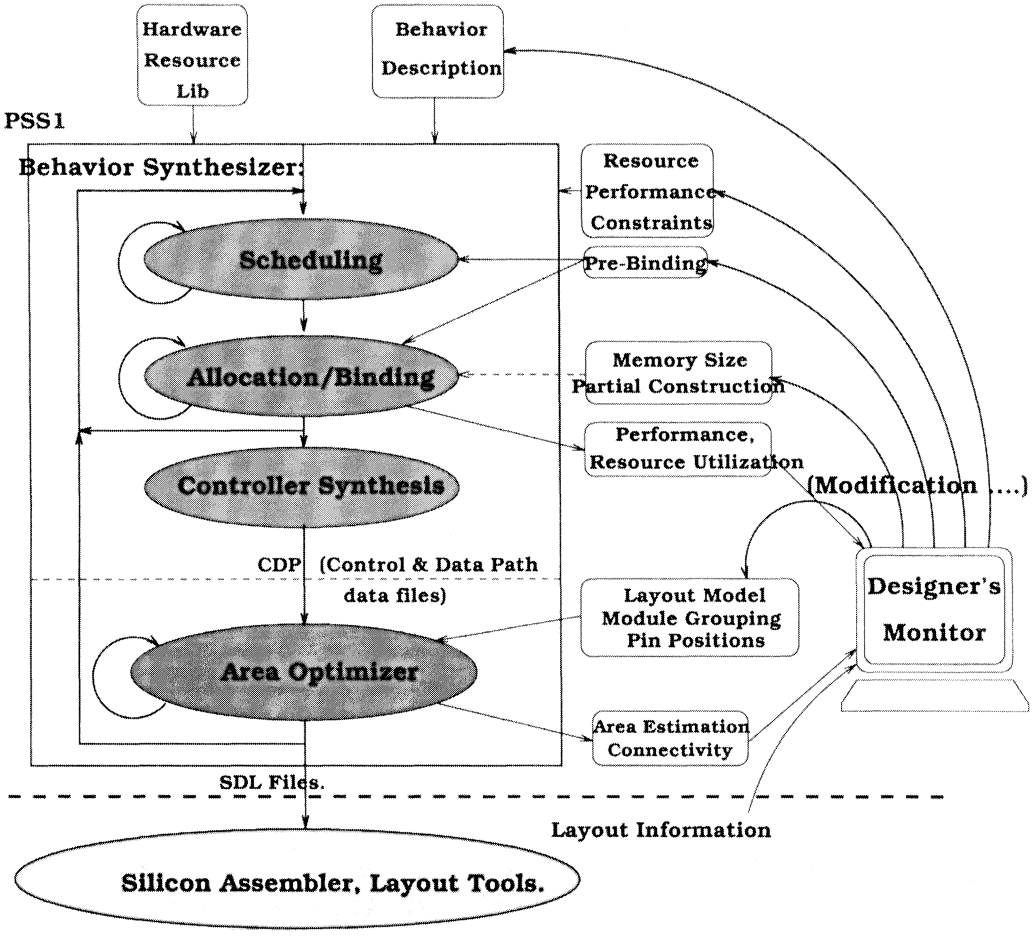

FIGURE 1 System structure of PSS1 high level synthesis system.

a behavior synthesizer and an area optimizer. The behavioral synthesizer takes a behavioral description as its input and produces a structural design as its output. The area optimizer finds a layout for the design chosen by the designer.

Target Architecture: To efficiently synthesize a design from a behavior description, the architecture on which the design will be based should be predefined. Although predetermining a target architecture may reduce the flexibility of a design, it, however, greatly reduces the search space for the optimization algorithms. In our architecture, the data path consists of a set of global buses, a set of function units and memory elements. Both storage elements and function units are hooked on the busses. In the storage part, registers for storing variables are grouped into register files and constants into ROM's. Latches (input and output) are inserted between function units and buses. Every register file, ROM and function unit can be connected to every bus as determined by the data path allocator.

The controller consists of a control memory and a sequencer. The control memory emits the control signals to supervise the execution of the elements in data path. It also provides some control bits for the sequencer to determine the next address. A two-phase non-overlapping clock is used. Detailed definitions of the target architecture has been described in [8][9][10].

Design Specification: The inputs to the system include a *behavioral description*, *a hardware-resource library*, *cost/performance constraints*, and *partial/complete structure specification*. We will briefly summarize each in the following.

1. The behavior of a digital system is described in VHDL (subset). Operations can be data manipulation operations, like arithmetic/logic operations, or control flow operations such as *if*, *case* and *loop* statements. References can be either signals which do not need to be stored or variables which need to be stored in registers, memories, or ports. Since VHDL was designed for simulation, we include some additional features, such as commands for specifying control dependencies or methods of defining a new operation (e.g., shift-and-add), to direct the synthesis process.
2. The *hardware resource library* describes the hardware resources available for the design. It specifies the cost, the silicon area, the functions, as well as the timing characteristics (such as the delay time and the stage time of pipelining) of each primitive module. The information will be used by the behavioral synthesizer to evaluate the quality of a design during synthesis process.
3. The *cost/performance constraints* include the resource constraints and performance constraints of the final design. The resource constraints such as the number of global buses, the number of function units of each type, and the number of memory elements (e.g., register-files or ROMs) may be modified interactively to determine the area/time trade-off at the scheduling stage. If no constraint is given, a minimum number of function units of each operation type will be used. The performance constraint is used by the optimization algorithm to guide the design towards an optimal one.
4. The *partial structure specification*, an optional information provided by designers, describes a partially bound data path. It may contain information about the final design such as the size of memory modules (e.g., register-files or ROMs), partial binding of objects (e.g., variables, data transfers, or operations) to hardware resource (such as register-files, global busses, or function units), and the partial interconnections between modules. With the aids of *partial structure specification*, not only the synthesizer can narrow down the search space of the design, but also the designer gains the control over the synthesis process. With this feature, designers can improve the design during the synthesis process. The layout information feedback from the layout tools will be used to guide the synthesizer to produce a better design.

3. BEHAVIOR SYNTHESIZER

Scheduling: Scheduler assigns operations to control steps. GSSP [13] is the scheduler used in the system. The special feature of the scheduler is that it allows

code movement across branches and nested-ifs. The input is a behavior description and resource constraints, and the output is a scheduled control/data flow graph (SCDFG). The behavioral description is first compiled into a set of basic blocks and control flows. Each basic block contains a sequence of consecutive statements in which flow of control enters at the beginning and leaves at the end without halting or possible branching. Program transformations (such as loop unrolling and variable renaming) are applied to increase the parallelism before scheduling. Operations are moved between blocks or different loop bodies during the scheduling to fully utilize the hardware resources and shorten the execution delay. Resource constraints can be modified interactively to fine tune the cost/performance ratio of the scheduled result during scheduling.

Scheduling is basically resource-constrained. The user interactively specifies the number of function units of each type and the scheduler reports the scheduling result, including the number of clock cycles needed and an analysis of the resource utilization. Resource utilization information gives the designer the resource utilization information. User can try a number of different resource allocations and decide which one to accept. Since scheduling process takes only few second, a designer can try out several different combinations of resources and pock up the best for data path allocation.

Data Path Binding: STAR [9] is a data path binder which takes a SCDFG as input and produces a net list of the data path as its output. It performs the data path synthesis in two phases: *data path construction* and *data path refinement*. In the first phase, an initial data path is constructed using a set of heuristic rules. The sizes of memory elements and the interconnections between data path modules are taken into account as part of cost function during the construction of data paths. In the second phase, the initial data path is refined by an iterative and global approach. In each iteration, it evaluates the binding quality of each object (data transfers, operations, and variables), then probabilistically selects a cluster of heavy correlated objects, rips up the objects in the cluster and rebinds them by a branch-and-bound search technique. The refinement process is terminated after no more cost improvement for a number of iterations.

The execution time of the branch-and-bound search algorithm depends on the 'cluster-size' which is user given parameter. Our experience shows a cluster-size within 16 and 25 gives the best tradeoff between the run time and the solution quality. In general, the larger the cluster-size is, the higher potential the improvement will be. However, larger cluster size means longer run time. Normally, the whole process takes minutes to half an hour depending on the design size and cluster size. Users can decide whether to proceed to the next step or change the cluster-size and re-synthesize the data path.

Controller Synthesis: The controller is designed based on a micro-code architecture. The synthesizer, MACS [11], is capable of synthesizing a multi-branch controller for application specific multi-function-unit processors. The controller is designed to pipeline the control word fetching and operation executions. A schedule in operation level is converted into micro-operation level. Control words are re-arranged by MACS to fit the control scheme. Then it optimizes the delay slots and assigns an address for each word such that the size of control ROM is minimal.

4. AREA OPTIMIZER

The Area Optimizer takes the output from the behavior synthesizer, and optimizes the layout of the final design according to a two-level macro-cell layout model. The output is described in SDL [7]. Whether to group or split modules and how the pin positions will be can be controlled by designers. The final layout is generated by the tool set of the Lager system.

Based on our bus-based architecture, we have developed a two-level, processor and module level, layout model for the final layout. On the top (processor) level, a processor is constructed by a connection of number of functional modules, memory modules, global busses, and a micro-code controller. Fig. 2(a) shows the processor level block diagram. We can fur-

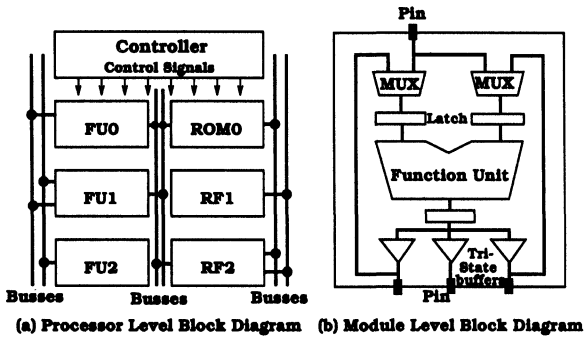

FIGURE 2 Two level layout model.

ther decompose a functional module into several submodules such as multiplexers, latches, tristate-buffers, and a module core (as shown in Fig. 2(b)). A module core can be either a function unit (e.g., adder, multipler) or a memory unit (e.g., register-file, ROM).

Designers can determine whether to merge or split modules based on the information generated from the area optimizer. The area optimizer reports an estimation of the size and aspect ratio of each module, the pin positions of each module, and the connectivity between modules. Based on this information and the visual display aids from floorplanning tool, designers can iteratively merge or split modules, or re-define the pin positions of the module. The process continues until a best layout is obtained. The main features of this layout model are:

- **Locality and Modularity**
  Each basic module collects the most strongly connected primitive cells into a single block, which forms a natural function cluster. The layout has the features of modularities and locality. Because the basic components to perform a function are grouped into one single module, functional simulation and timing verification can be easily done. Further, the modularity increases the testability of the final circuit.
- **Controllability of the number, the size, and the aspect ratio of modules**
  The capability of merging or splitting modules gives designers controllability over the size and aspect ratio of each module and the total number of modules in the top level. Based on the connectivity information and the size of each module, designer can either merge some small modules into a larger one, or split a big module into several smaller modules. Module merging which reduces the number of modules in the top level of a hierarchy simplifies the task of floorplanning and global routing. Moreover, it puts some global busses into local interconnections inside modules, and thus reduces the routing complexity among the modules. The total silicon area can be reduced considerably by proper merging of modules.
- **Pre-definability of module shape and pin positions:**
  The designer can specify the number of rows and the I/O pin positions for the standard cell modules during net list generation. Shape matching can significantly reduce the dead space of a macro cell layout, while good assignment of pin positions contributes to shorter wiring paths and less routing area.

5. EXPERIMENTS

In our experiments, a design in behavior description is first synthesized by the behavior synthesizer and then the area optimizer and the Lager system are used to complete the final layout. The synthesized design is described textually by a hierarchical and parameterized description using Structural Description Language (SDL). To achieve greater flexibility and better design quality, designer can choose hardware components from different cell libraries (such as TimLager, Stdcell, and dpp) to construct a design. The area optimizer is capable of producing the proper input descriptions for various layout tools (such as bdsyn logic synthesizer, and stdcell/macro cell layout generators). The SDL description for each module contains parent cell parameters, layout generator command, structure processor command, subcells, nets, and terminals. The layout of each module is produced by the layout generator specified in the description. The Flint macro-cell layout generator is used to complete the chip layout at the top level. A designer can either automatically generate the layout based on a slice floorplan or manually place the modules.

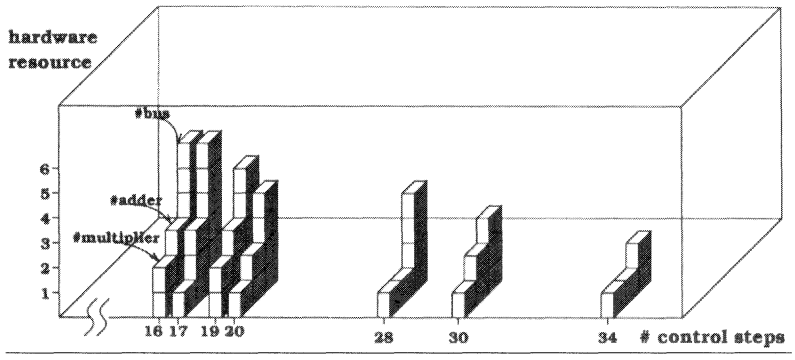

FIGURE 3 Design exploration of Elliptic Filter.

Most data path components chosen from the dpp library are based on a bit-slice structure. The dpp structure processor [7] tiled the components in the vertical direction and placed linearly along the horizontal direction while use of the feed-throughs to minimize the wiring area. As a result, it produces very compact results. However, since components are placed linearly, the shape of a module is usually long and slim when the number of components in a module is large. Our area optimizer will generate the geometric information for each module. Based on the shape and size of each module, a designer can determine whether to merge or split modules.

In the first experiment we use the fifth-order digital elliptical wave filter [14] to show the capability of exploring design space of our system. By modifying the resource constraints, a number of designs with different performance and hardware cost can be obtained by our scheduler. For examples, the behavior synthesizer produces 7 synthesis results (Fig. 3) with performance ranging from 34 control steps to 16 control steps, and different hardware resource requirements. A designer can choose some from these designs for area optimization.

For illustration purpose, suppose we choose the design with 17 control steps, 6 global busses, 3 adders, 1 multiplier, 1 constant ROM, and 4 register-files, for area optimization. By using the proposed two level layout model, the area optimizer first produces a net-list for each module. As shown in Fig. 4(a), it contains 9 dpp modules, 6 global busses, and 24 connections (module pins). Since RF_0 and FU_3 are the only two modules connected by bus_1 in Fig. 4(a), they are selected to be merged. After module mergence, bus_1 becomes an internal wire of the merged module. Similarly, RF_1 and FU_1 are merged and then RF_3 and FU_0 are also merged. Fig. 4(b) shows the result after these merges. The resulting net-list contains 6 dpp modules, 3 global busses, and 15 global connections. We may further merge these modules. A final net-list (Fig. 4(c)) is generated by 3 more module mergences. The final layout contains 3 dpp modules, 3 global busses, and 9 connections.

The text display of the net list and the layout of the design for 6 dpp modules, 3 global buses and 15 global connections are shown in Fig. 5. The module width (*mm*), module height (*mm*) and the silicon area

```
Cluster 0(length  50.50) :BUS -1---- (RF0)
Cluster 1(length  43.30) :BUS --2--5 (RF1)
Cluster 2(length  22.90) :BUS --2-4- (RF2)
Cluster 3(length  33.10) :BUS 0-23-- (RF3)
Cluster 4(length  11.70) :BUS 0---4- (RF4)
Cluster 5(length  36.10) :BUS 0-234- (FU0)
Cluster 6(length  41.10) :BUS 0-2-45 (FU1)
Cluster 7(length  36.10) :BUS 0-2-4- (FU2)
Cluster 8(length  26.10) :BUS 01--4- (FU3)
```

(a) initial net-list (9 dpp modules)

```
Cluster 0(length  81.90) :BUS 0-2-45 (RF1,FU1)
Cluster 1(length  22.90) :BUS --2-4- (RF2)
Cluster 2(length  66.70) :BUS 0-234- (RF3,FU0)
Cluster 3(length  11.70) :BUS 0---4- (RF4)
Cluster 4(length  36.10) :BUS 0-2-4- (FU2)
Cluster 5(length  74.10) :BUS 01--4- (RF0,FU3)
```

(b) after module mergences

```
Cluster 0(length  91.10) :BUS 0-2-45 (RF1,FU1,RF4)
Cluster 1(length  94.50) :BUS 012-4- (RF2,RF0,FU3)
Cluster 2(length 100.30) :BUS 0-234- (RF3,FU0,FU2)
```

(c) final net-list (3 dpp modules)

FIGURE 4 Example for module mergence.

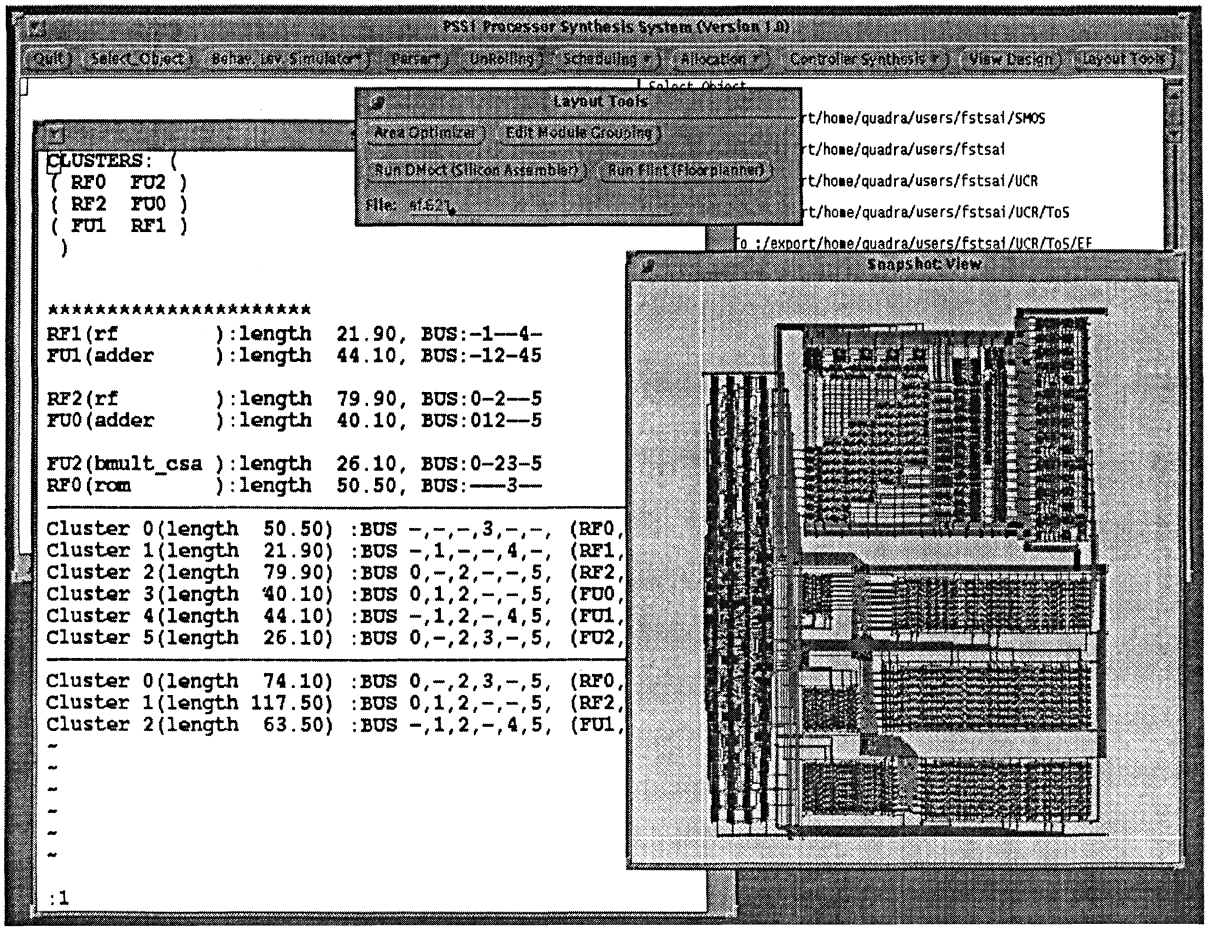

FIGURE 5 Layout of Elliptic Filter.

(mm^2) of each module as well as of the designs are shown in Table I. All designs use the MOSIS 1.0 micron SCMOS technology. The wiring area on the processor level can be reduced drastically after a proper mergence of modules. In the above three cases, the wiring space and the empty space among modules of each design are 10.56 mm^2, 7.40 mm^2, and 5.44 mm^2, respectively. The aspect ratio of the controller can also be changed to best fit the chip layout. In the above designs, the number of rows of the controllers are all set to 4. Notice that, in this experiment, we use registers and multiplexors from the dpp library to construct the register-files. In general, the memory components chosen from TimLager library have a smaller size.

In addition to the Elliptic filter, we also did some experiments on the following three benchmarks: GCD, FIR filter [15], and FDCT [16]. We used the behavior synthesizer to synthesize several designs for each benchmark, the area optimizer to merge/split module and re-define pin position of modules, and Flint floorplanner/layout generator to produce the chip level layout. For each design, a number of floorplans are generated either automatically or manually by Flint. A range of layout area for each synthesis result is obtained (Fig. 6). Our experiences show that, as the number of modules on the top level reduced, the interconnection complexity is reduced and, in general, less wiring area was needed to complete the layout. It is interesting to see that the range of layout area often

TABLE I Layout area comparison for 3 different designs of Elliptic Filter

	Design (a)	Design (b)	Design (c)
RF_1	(1.10, 0.59, 0.65)	(2.32, 0.64, 1.48)	
FU_1	(1.17, 0.61, 0.71)		(2.71, 0.77, 2.09)
RF_4	(0.47, 0.47, 0.22)	(0.47, 0.47, 0.22)	
RF_2	(0.62, 0.51, 0.32)	(0.62, 0.51, 0.32)	
RF_0	(1.39, 0.61, 0.84)	(2.22, 0.64, 1.42)	(3.07, 0.79, 2.42)
FU_3	(0.76, 0.66, 0.51)		
RF_3	(0.83, 0.60, 0.49)	(1.84, 0.68, 1.25)	
FU_0	(0.96, 0.51, 0.49)		(2.91, 0.70, 2.03)
FU_2	(1.10, 0.51, 0.56)	(1.10, 0.51, 0.56)	
multipler	(2.41, 3.21, 7.73)	(2.41, 3.21, 7.73)	(2.41, 3.21, 7.73)
controller	(2.65, 1.29, 3.43)	(4.56, 0.82, 3.73)	(2.41, 3.21, 3.73)
Area Sum of Modules	(-, -, 15.95)	(-, -, 16.71)	(-, -, 18.00)
Chip Area	(4.96, 5.35, 26.51)	(4.67, 5.16, 24.11)	(4.62, 5.07, 23.44)

become wider when the number of modules is reduced. This is because larger module often leaves more empty spaces among the modules. However, the layout area can be reduced if a better floorplan is found; Our experiments show that we can obtain a design with the smallest layout area in most of the cases if the number of merged dpp modules is within the range of 3 to 8. Note that, in the case of GCD, the best layout was obtained by merging all the data path elements into a single module because the design is very

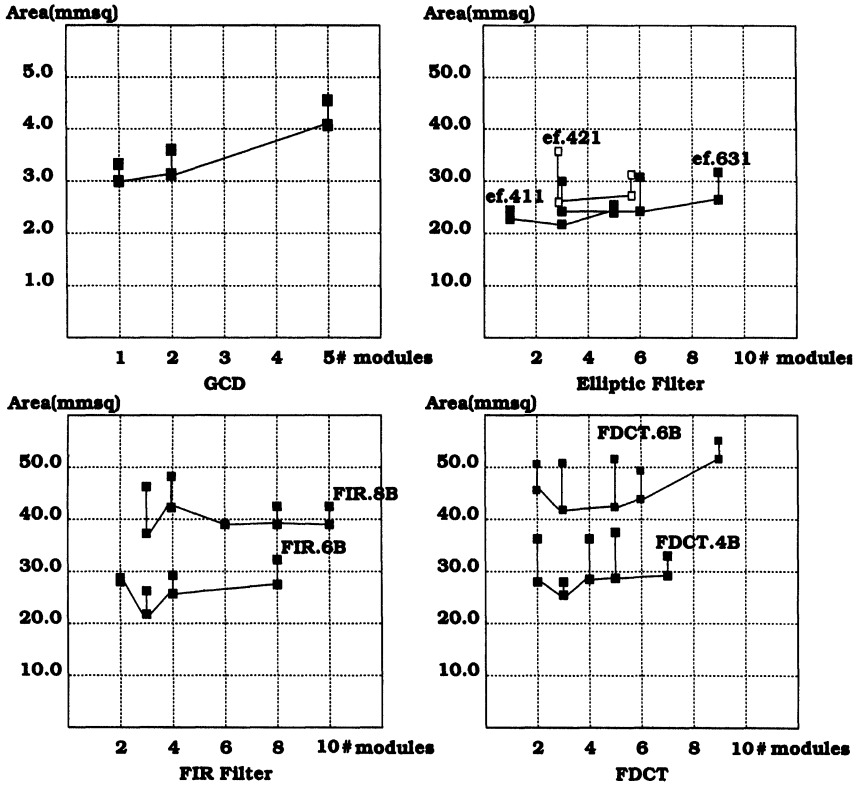

FIGURE 6 Area optimization experiment based on proposed 2-level layout model.

small. Fig. 7 shows the layout of GCD with only one data path module and a control module.

Table II shows the results of benchmarks. The area of FDCT with 15 control steps and the area of FIR with 9 control steps are much larger than those of others because both of them use 2 carry save multipliers. All the cases are for 8-bit designs. The run time of behavior synthesizer and area optimizer for each design is within several minutes (excluding description input time). The scheduling for these benchmarks typically takes a few seconds; the allocation takes a few minutes; the area optimization may take a few minutes; while to generate a layout may take hours on a SUN SPARC station.

6. CONCLUSIONS

We have presented a design methodology for high level synthesis. The methodology is divided into two phases: behavioral synthesis and area optimization. We take the size and delay of the modules into consideration during the behavioral synthesis phase, and the module shape and I/O pin positions during the area optimization phase. Using the connectivity information among modules and the size information of each module, designers can iteratively merge or split modules, determine the size of modules, and define pin positions to optimize the area of the final chip layout. Our experience with the system shows that

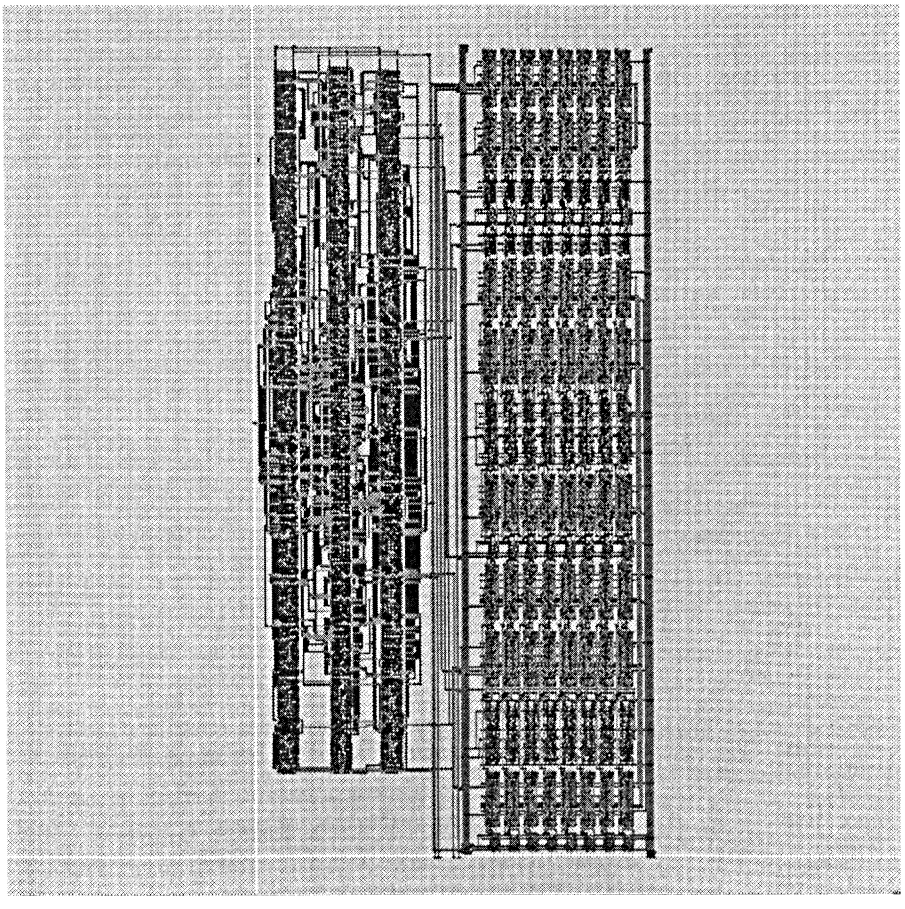

FIGURE 7 Layout of the GCD example.

TABLE II Layout experiment for benchmarks

Benchmarks	performance	# buses	# multipliers	# of alus,	# regs	chip
GCD	4	2	0	1	3	3.03
EF	28	4	1	1	12	21.60
	20	4	1	2	9	24.08
	17	6	1	3	10	26.51
FDCT	22	4	1	2	11	25.77
	15	6	2	2	10	46.08
FIR	15	4	1	1	21	20.33
	13	6	1	2	23	22.41
	9	8	2	2	18	37.60

the capability and the characteristics of the layout tools greatly affect the solution. As a result, the area optimizing approach are highly co-related to the cell library and the layout tools used.

Acknowledgments

This work has been supported in part by the California Micro Program.

References

[1] M. C. McFarland, A. C. Parker, and R. Camposano, "Tutorial on High-Level Synthesis," *Proc. of 25th Design Automation Conference*, pp. 330–336, June 1988.

[2] D. Gajski, *Silicon Compilation*, Addison-Wesley, New York, 1988.

[3] C. Tseng and D. P. Siewiorek, "Automated Synthesis of Data Paths in Digital Systems," *IEEE Trans. on CAD*, pp. 379–395, July 1986.

[4] B. S. Haroun and M. I. Elmasry, "Architectural Synthesis for DSP Silicon Compiler," *IEEE Trans. on CAD*, pp. 431–447, April 1989.

[5] H. DeMan, J. Rabaey, P. Six and L. Claesen, "Cathedral-II: A Silicon Compiler for Digital Signal Processing," *IEEE Design and Test*, pp. 13–25, Dec. 1986.

[6] P. Pfahler, "Automated Data Path Synthesis: A Compilation Approach," *Microprocessing and Microprogramming*, North-Holland, pp. 577–584, 1987.

[7] C. B. Shung, *et al.*, "An Integrated CAD System for Algorithm-Specific IC Design," *IEEE Trans. on CAD*, pp. 447–463, April 1991.

[8] F. S. Tsai and Y. C. Hsu, "Data Path Construction and Refinement", *Proceeding of ICCAD-90*, pp. 308–311, November 1990.

[9] F. S. Tsai and Y. C. Hsu, "STAR: An Automatic Data Path Allocator", *IEEE Trans. on CAD*, (to be appear).

[10] S. Z. Lin, C. T. Hwang and Y. C. Hsu, "Efficient Microcode Arrangement and Controller Synthesis for Application Specific Integrated Circuits," *Proceeding of ICCAD-91*, pp. 38–41, November 1991.

[11] C. T. Hwang, Y. C. Hsu and Y. L. Lin, "Scheduling for Functional Pipelining and Loop Folding," *Proc. of the 28th Design Automation Conference*, June 1991.

[12] C. T. Hwang, Y. C. Hsu and Y. L. Lin, "Optimum and Heuristic Data Path Scheduling under Resource Constraints," *Proc. of the 27th Design Automation Conference*, June 1990.

[13] S. H. Huang, C. T. Hwang, Y. C. Hsu, and Y. J. Oyang, "A New Approach to schedule Operations Across Nested-ifs and Nested-loops" *25th International Synposium on Microarchitecture*, Dec. 1992.

[14] D. E. Thomas and R. L. Blackburn, "The System Architect's Workbench," *Proc. of the 25th Design Automation Conference*, pp. 337–343, June 1988.

[15] N. Park and A. C. Parker, "Sehwa: A Software Package for Synthesis of Pipelines from Behavioral Specifications," *IEEE Trans. on CAD*, Vol. 7, No. 3, pp. 365–370, March 1988.

[16] D. J. Mallon and P. B. Denyer, "A New Approach To Pipeline Optimization," *The Proceedings of The European Design Automation Conference*, March 1990.

Authors' Biographies

Fur-Shing Tsai received the B.S. degree in chemical engineering from Tung-Hai University, Taiwan, in 1984, and Ph.D. degree in computer science from Tsing Hua University, Taiwan, in 1991. From 1991 to 1992, he was a Post-Doctoral researcher in the computer science department, University of California, Riverside. He is now with S-MOS Systems in San Jose. His research interests include physical design automation and high level synthesis. He co-received an Outstanding Young Author Award from IEEE Circuits and Systems Society in 1990.

Yu-Chin Hsu received the B.S. degree in computer science and information engineering from National Taiwan University, Taiwan, in 1981, and the M.S. and Ph.D. degrees in computer science from the Univer-

sity of Illinois at Urbana-Champaign in 1986 and 1987, respectively. He was an Associate Professor of Computer Science at Tsing Hua University, Hsin-Chu, Taiwan, from 1987 to 1991. He is currently Associate Professor of Computer Science at University of California, Riverside. His research interests include most aspects of computer-aided design for VLSI, computer architecture design and synthesis. He co-received an Outstanding Young Author Award from IEEE Circuits and Systems Society in 1990.