Unified System Construction (USC)

BY

Alice C. Parker, Kayhan Kucukcakar,
Shiv Prakash and Jen-Pin Weng

CEng Technical Report 91-01

January 29, 1991

Electrical Engineering Systems

University of Southern California

Los Angeles, CA. 90089-0781

Unified System Construction (USC)

Alice C. Parker Kayhan Küçükçakar Shiv Prakash
Jen-Pin Weng

Electrical Engineering – Systems
University of Southern California
Los Angeles, CA 90089-0781

January 29, 1991

To appear in *Trends in High-Level Synthesis*, edited by
R. Camposano and W. Wolf, Kluwer Academic Publishers, 1991.

This work was supported in part by the Department of Air Force, the Department of Army and the Department of Navy, Contract No. N00039-87-C-0194, in part by the Defense Advanced Research Projects Agency and monitored by the Federal Bureau of Investigation under contract No. JFBI90092, and in part by Semiconductor Research Corporation under contract No. 89-DJ-075. The views and conclusions considered in this document are those of authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of Defense Advanced Research Projects Agency or the U.S. Government.

1 Introduction

When high-level synthesis research began, there were no VLSI chips. Design was assumed to be done with a fixed set of available modules [1]. In at least one case, these modules were assumed to be TTL chips [16]. Such chips and modules had a fixed cost, and wiring delays between chips were minimal compared to the processing delays on chip. Power consumption could easily be computed as the sum of the power consumptions of individual chips, and hot chips could be cooled with a heatsink. Partitioning onto multiple boards was performed manually, or at least handled separately from the high-level synthesis process. General-purpose multiprocessors were constructed only in research laboratories and special-purpose multiprocessors were expensive and constructed for only a few applications. Now, however, we are faced with a situation where high-level synthesis programs must design datapaths and controllers to fit on one or more VLSI chips and boards. There may be multiple processing elements, each with its own controller. For VLSI chips, a large portion of the chip area is consumed by wiring and wire delays can be significant. Partitioning must be performed to map problems onto multiple chips. Many problems are best implemented with multiple control environments, which implies multiprocessor architectures. Given this situation, high-level synthesis programs must take a number of factors into account that were by and large ignored in the early days of synthesis research.

This chapter describes four of the ongoing research tasks at the University of Southern California which address the situation described above. The four tasks we discuss are chip partitioning, performance-area prediction, system-level synthesis and combined scheduling/floorplanning. First, we show the relationships between these four tasks, and then we summarize related efforts. Finally we describe these four active projects which take into account system-level design considerations and physical layout impacts.

Figure 1 illustrates the overall structure of the USC system-level synthesis research described in this chapter. A fifth task, control state specification, is also shown, and is the subject of a recent dissertation [20].

2 Related Research

2.1 Prediction Research

Although most high-level synthesis programs do not use predictions for the design characteristics yet to be determined, we cite BUD [36], ELF [11] and

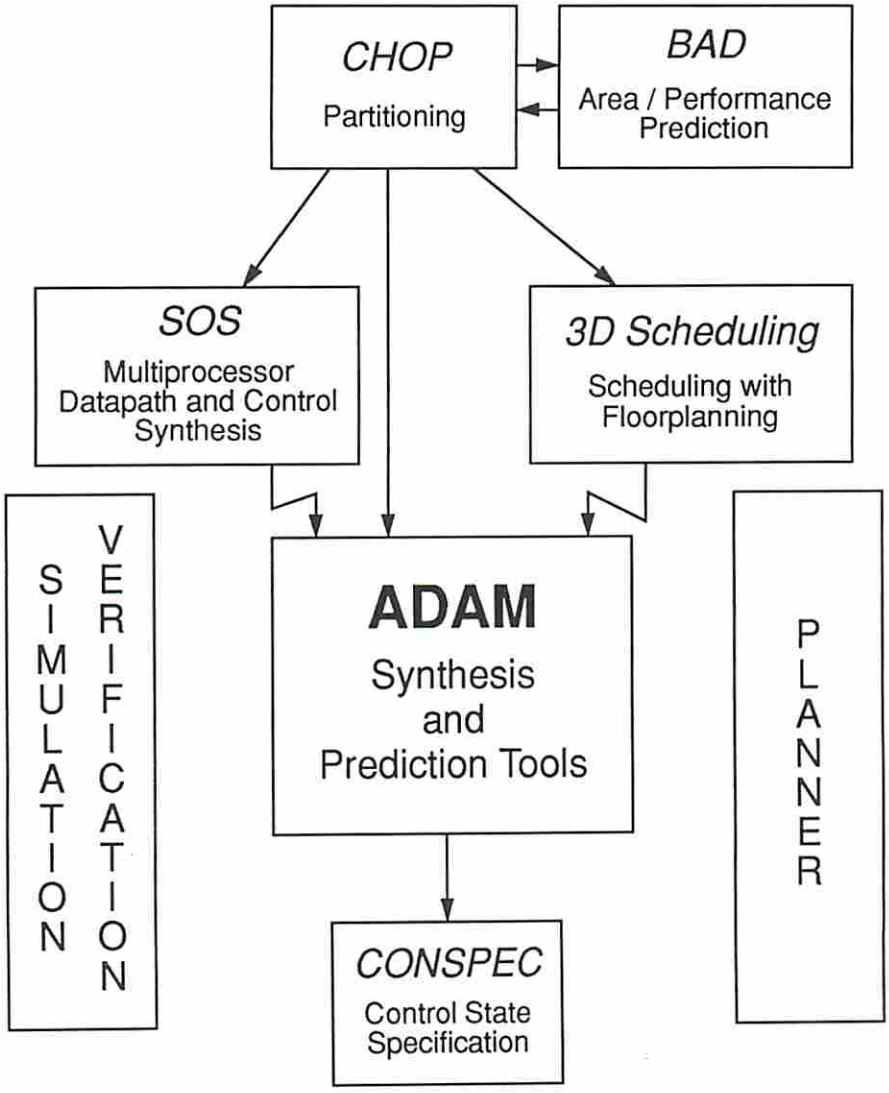

Figure 1: The Unified System Construction (USC) Project

CHIPPE [4] as exceptions. Also, the ADAM System [25] uses predictors to guide the design search.

The early work on the functional allocation predictions was on the prediction of lower-bound functional area with varying time characteristics for pipelined and non-pipelined design styles [27, 26]. The register estimation in the work described here is mainly based on Mlinar's model [38]. The wiring space model proposed by Kurdahi estimates the wiring space requirement of an RTL design assuming that a standard cell layout style will be used [32]. There exist some methods proposed by Mlinar to predict the controller area of RTL designs for the PLA style [38]. PLA delay and wiring delay estimations also exist [13].

2.2 Partitioning

Partitioning has been used at many stages of the digital design process. A study on partitioning strategies for multi-chip VLSI microprocessors was performed in [47]. Partitioning of logic circuits in order to meet gate and pin count constraints of chips was described by Payne and vanCleemput in [43]. SPARTA [46] is a tool which evaluates an RTL design by checking if any area, power and pin count constraints are violated.

A partitioning algorithm at the behavioral level using a multi-level clustering approach has been proposed to reduce the control overhead and routing area [7] but this method does not consider design constraints. Another effort on partitioning at the behavioral level is described in [14].

A heuristic for partitioning graphs with costs on the edges into subgraphs with specified sizes while trying to minimize the total cost of edges cut was proposed by Kernighan and Lin [28]. Although this heuristic can be applied successfully to partitioning of logic circuits or RTL designs, the model used in this approach is not directly applicable for partitioning of behavioral specifications due the fact that behavioral synthesis introduces significant and generally irregular sequential behavior into the design. This causes most final design characteristics to be a function of the sequential behavior introduced and hence a function of the structure as well as of the original behavior. Without having the results of behavioral synthesis or accurate predictions of these results, it is questionable if one can directly correlate "sum of costs of values cut" to the pin count requirement or "sum of sizes of operations in a partition" to the area of chips. In addition, the areas of chips are consumed by not only functional units but also by registers, steering logic, controllers and wiring.

2.3 Scheduling

Very little synthesis research has taken into account physical design effects. An early study of the effects of layout on the design curve was performed by Granacki and Parker [12]. BUD is a program which floorplans prior to synthesis [36]. Fasolt [29] floorplans and analyzes area impact during high-level design. ELF is an early system which estimates interconnection effects during synthesis [11]. Chippe is a constraint-driven expert system which allows users to specify area, time and power constraints for CMOS gate array design [4]. Chippe predicts wire delay from the structural RT-level design.

Many current approaches consider the scheduling and floorplanning problems separately, which optimizes designs locally. Furthermore, many current floorplan packages [50] [33] [2] minimize the total interconnection cost of a net list as their objective. They pay no special attention to reducing the interconnection length between the operators along the critical path.

Timing Driven Layout Design incorporates timing information to influence the placement and wiring processes [10] [9] [5] [39]. This approach uses the path analysis data produced by a static timing analysis program to generate weights for critical nets of clocks and data paths. These weights are then used to bias automatic placement (and/or routing) in the layout program. However, this step is performed after scheduling. Iteration must be performed to take advantage of timing-driven layout.

2.4 System-Level Synthesis

Hafer and Parker [17] used a mixed-integer linear programming approach to automatically synthesize register-transfer level datapaths, given a data flow/control flow graph description of the hardware. The approach involves developing various timing relationships to be satisfied, but does not include interconnection styles or delays, and does not consider the detailed timing of multiple outputs. Hwang et. al. [22] have described an integer linear programming model for the scheduling problem in data path synthesis under resource constraints, and they present a heuristic technique called *Zone Scheduling* for solving large size problems.

The configuration of existing components using predesigned interconnection strategies was the subject of the R1 expert system, a successful package used by DEC to configure the systems it markets. Sara is a well-known system-level tool package which supports the designer in making design decisions, but which makes no design decisions of its own[8]. ADAS is a commercial system-level package which supports the designer by repre-

sentation and simulation. The assignment of tasks to a fixed multiprocessor system was inspired by Stone's work on the two-processor problem [48]. More recently, Chu et. al. [6] have described an integer 0-1 programming approach to the problem of task allocation in distributed data processing. The problem they have considered involves the allocation of a set of m subtasks to a set of p (fixed) processors already interconnected in some fashion. They begin with the task already partitioned into subtasks. They assume the number of processors and the interconnection scheme is known. Their model does not consider the data precedence relations among the subtasks. Houstis [21] describes task allocation for real-time applications with concurrent selection of the optimal number of processing elements. She does consider data precedence, but assumes identical processing elements and does not consider timing constraints. Indurkhya et.al.[24] use random models for distributed programs, and confirm intuitive results for the 2-processor and n-processor cases.

The CATHEDRAL-II system [45] synthesizes a multiprocessor architecture, however, the architecture is almost completely fixed. Haroun and Elmasry [19] mention multiprocessor architecture synthesis for DSP applications, however, this paper primarily concentrates on the design of an individual processor within the system. There have been many research efforts on static scheduling of DSP algorithms on already designed synchronous general purpose multiprocessors, (e.g., [23]). Haddad[15] described a load allocation problem solved with continuous partition sizes to minimize total execution time. Talukdar and Mehrotra [37] describe a problem which is a simplified and similar version of the multiprocessor synthesis problem considered here. They do not consider interprocessor communication delays explicitly. They also model the problem using mathematical programming, though they use heuristics to solve it. An early study mapping algorithms such as the Fast Fourier Transform onto the CM* multiprocessor in order to meet real time constraints was undertaken by Brantley [3].

3 BAD: Behavioral Area-Delay Predictor

Even though automated synthesis can speed up the design process by orders of magnitude, the growing size of designs together with the multitude of tradeoffs possible during synthesis make it impossible to completely search the design space, even with synthesis tools which span behavior to layout. In order to improve the quality of generated designs, the designer has to iterate several times by modifying the functional specification (high-level transformations), design constraints and design decisions. This iterative process can be quite time consuming. Fast prediction tools to estimate

the impact of tentative design decisions while taking into account physical design effects can be extremely beneficial in reducing the design iteration time.

BAD (Behavioral Area-Delay Predictor) is a fast, comprehensive and integrated area-delay prediction tool which can be used to support behavioral synthesis tools [30]. BAD combines former theoretical and heuristic prediction research done at USC along with new additions around a unified statistical representation model. PERT [34] is selected as the statistical model for initial experimentation.

The prediction model includes functional, register, multiplexing, wiring and control area and delay predictions. The predictions are based solely on the behavior of the target design, the specific architecture style and area-performance constraints.

3.1 Overview of BAD

Predictions are performed by following the actual behavioral synthesis guidelines without dealing with the actual synthesis details. The overall area-delay predictions are obtained by performing several prediction subtasks (corresponding to actual synthesis tasks) in a specific order. The initial data pool (before performing any predictions) consists of the input data given to BAD (clock cycle, data flow graph, design library, constraints and goals). Each prediction subtask contributes some predicted characteristics of each potential design to the data pool. So, each prediction subtask uses the original input data as well as prediction results existing prior to its execution.

BAD generates predicted designs for all possible (and meaningful) implementations of the design. It enumerates configurations of all module sets (each module set contains one module per operation type). For each module set configuration, pipelined and nonpipelined style predictions are generated for each possible value of initiation interval and number of stages (in terms of clock cycles). The selection of the best feasible predicted design (satisfying design constraints) from the pool of predicted designs also solves the problems of design style selection and module selection.

The prediction subtasks include pipelined and non-pipelined functional allocation, pipeline length estimation, register area estimation, multiplexer area and delay estimation, estimation of number of 2-point nets, PLA area and delay estimation, wiring space area and delay estimation.

This prediction tool and the methods presented can be effectively used in a number of design tasks including but not limited to design space exploration prior to high-level synthesis, evaluation of behavioral transformations and guidance for system-level decisions. BAD is now being used in

the behavioral partitioning package described later which uses prediction methods to guide its search. The run-time for BAD averages about 0.5 msec of CPU time per predicted design on a Sun Sparc 4/460 (180msec for 346 predicted designs).

3.2 Results

To illustrate how well the prediction methods work, the AR lattice filter from [25] will be used to compare predicted and synthesized design points for the pipelined design style. The synthesized designs were produced by the ADAM synthesis tools [25]. Both the predictor and the synthesis tools are given the same library and technology (3 micron).

Figure 2 shows the combined operator, register, multiplexer and wiring areas of predicted (dotted line) and synthesized (points) designs for varying initiation intervals. The controller area is not included in the area figures. The wiring area for each point is obtained by using PLEST [32] for both curves due to difficulty in actually laying out these designs[1].

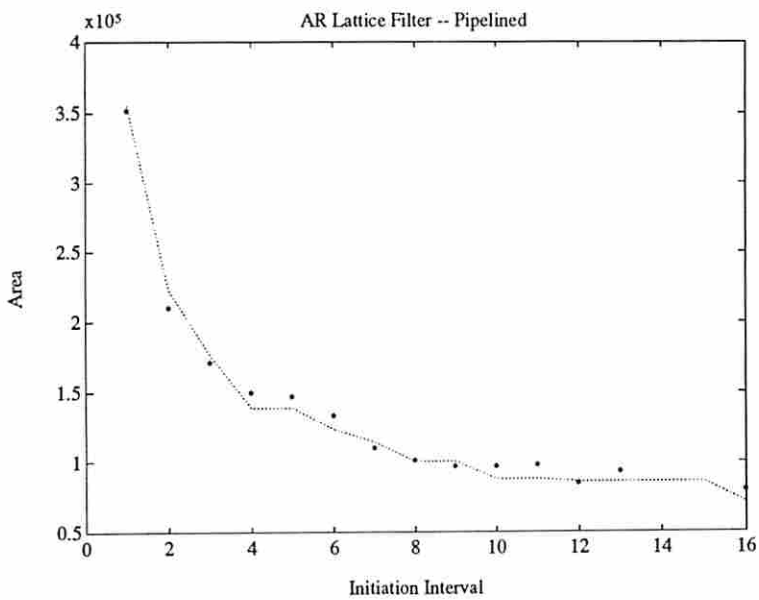

Figure 2: Overall Area - Delay characteristics

The estimated (lines) and actual (points) number of stages are shown

[1] Subsequent layouts of the AR filter have shown this to be a good assumption.

in Figure 3. The solid line is the expected number of stages and the dotted lines show the 3-sigma range for the estimated number of stages. Probability theory tells that more than 99% of actual characteristics will be within the 3-sigma range of the solid line if the distribution is normal.

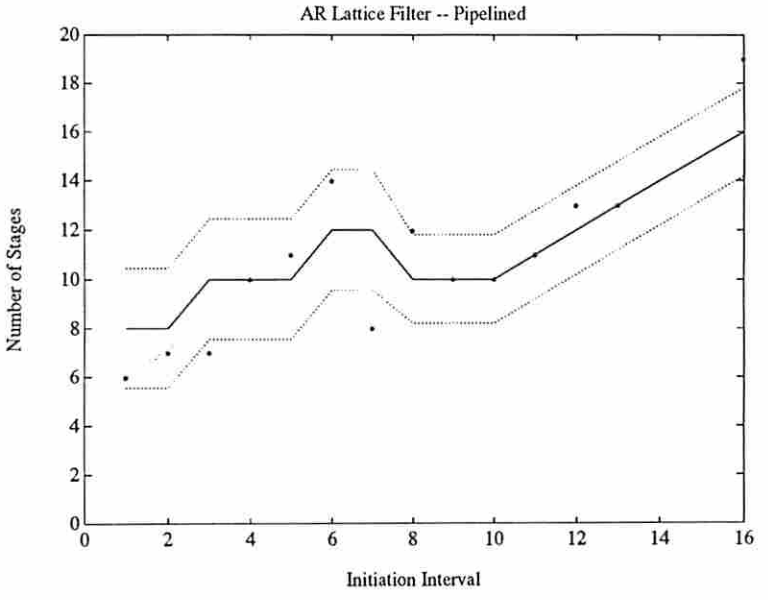

Figure 3: The variation of the number of stages

The register, multiplexing, PLA and wire delays (contributing to the main clock cycle length) predicted by BAD as well as the estimated delays of synthesized designs (using module set mul2, add3) were within 3% of the clock cycle (3000 ns) so their impact was minimal. But, when the predictions are performed with the same module set, a faster clock (300 ns) and multi-cycle operations, these delays increase to 30% of the clock cycle. When a technology with a smaller feature size is to be used, then these delays might easily become comparable to operator delays. Therefore, these delays could be crucial depending on the specific application and the technology.

4 A Constraint-Driven System-Level Partitioner

Although many digital designs are too large to fit on a single chip, at present much chip partitioning is performed manually by designers. With the increasing complexity of designs and quick turn-around time requirement, fast partitioning methods must be used.

Partitioning to meet design constraints can be performed at the behavioral level, the RT (Register Transfer) level or the logic level. If a design were synthesized as if it were a single chip design, but later partitioned onto multiple chips, the chances of synthesizing an inferior design would be considerably higher due to the fact that some of early design decisions were made without information about partition boundaries. In fact, if partitioning were done after behavioral synthesis, it may not be possible to find any feasible partitions. In such a case, the synthesis process would somehow have to be repeated to find a feasible partitioning. Alternatively, if the partitioning were performed at the behavioral level without any implementation information, then determining the feasibility of any partitioning would involve a complete synthesis process. Both cases might suffer severely from long iteration times to reach a feasible solution.

CHOP [31] is a constraint-driven interactive behavioral partitioner which is based on determining the feasibility of tentative partitions at the behavioral level using accurate prediction methods. CHOP is a fast tool since predictors are used in place of synthesis tools to supply feedback to the partitioner.

4.1 Overview of CHOP

The overall operation of the partitioner CHOP is shown in Figure 4. BAD (Behavioral Area-Delay Predictor) is embedded in the partitioner. The ovals indicate data and the rectangles indicate CHOP's and designer's actions.

CHOP allows the designer to interactively create and modify partitions of behavioral specifications onto multiple chips. Each chip may have more than one partition which might or might not be implemented independently. The partitioning is not performed purely in an abstract space; the approach deals with physical design parameters, constraints and goals.

The proposed partitioning approach is comprehensive. Detailed prediction techniques addressing most digital design aspects are used. The prediction techniques include: design style selection (pipelined/nonpipelined), module selection, operator, register and multiplexer allocation, wiring area/delay, controller area/delay, memory bandwidths, off-chip communi-

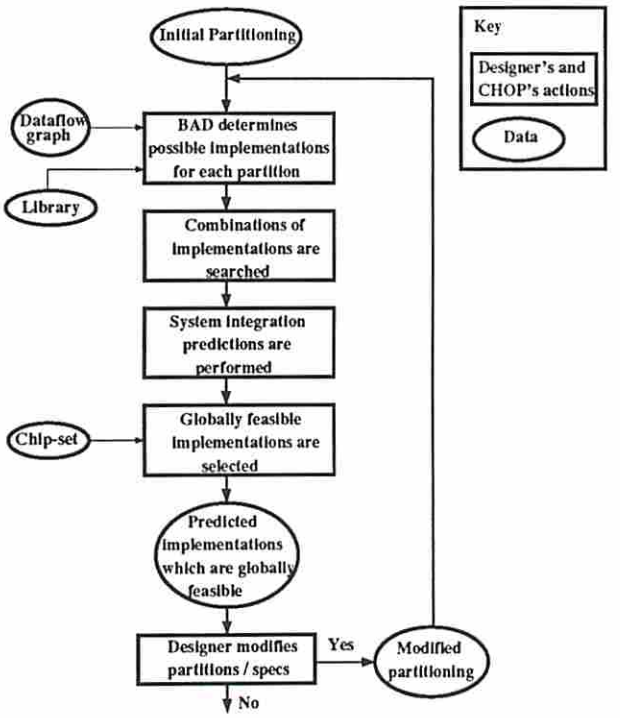

Figure 4: The overall operation of CHOP

cation bandwidths, performance matching between independent datapaths, data buffering, and a distributed control mechanism. CHOP uses probabilistic methods to determine the feasibility of tentative partitions.

After the feasibility of a partitioning is checked, the partitioning process can be continued as the designer modifies the specifications and constraints, based on the feedback from the CHOP partitioner. Modifications consist of 4 major groups:

- modifying behavioral partitions,
- modifying the memory hierarchy,
- modifying the target chip set,

- modifying constraints.

4.2 Results

To illustrate the results, the same AR Lattice filter will be used. Three partitioning schemes were evaluated using different design parameters. The first partitioning had a single partition, the second had two almost equal size partitions, and the third had three partitions of approximately equal size. In all cases, each partition was manually assigned to a separate chip.

Table 1 shows the set of chip packages considered in the experimentation. Table 2 shows the best designs found for each set of experiments. The type of heuristic used for a given run is shown under the field called H. E is for the limited enumeration heuristic and I is for the iterative heuristic results. All CPU times reported in Table 2 are in seconds on a Solbourne Series 5e/900 (Sun 4 compatible).

No	Width *mil*	Length *mil*	Number of Pins	Pad Delay *ns*	Pad Area mil^2
1	311.02	362.20	64	25.0	297.60
2	311.02	362.20	84	25.0	297.60

Table 1: A subset of MOSIS Standard Chip Packages

It can be seen from Table 2 that two times higher performance can be obtained easily by doubling the available chip area. If slightly longer system delays can be tolerated, then up to 3 times data input frequency increase is possible.

The effect of pin limitations can be seen in Table 2. Using 64 rather than 84 pin chip packaging causes a slight increase in the system delay of the predicted designs, mainly due to longer data transfer times of inputs and outputs.

It can also be seen from Table 2 that partitioning a design onto more and more chips in order to improve the performance or system delay characteristics may not always be possible. Partitioning a design onto more chips generally increases the usage of chip pins to transfer data between the chips and chip pins become the bottleneck in high-performance designs.

Partition Count	Package Type	H	CPU Time	Partitioning Imp. Trials	Feasible Trials	Initiation Interval	Delay	Clock Cycle ns
1	2	E	0.07	5	1	60	67	312
1	2	I	0.06	13	1	60	67	312
2	2	E	0.59	156	2	30	57	310
	2					20	79	309
2	2	I	0.21	9	2	30	57	310
	2					20	79	309
2	1	E	0.43	156	2	30	59	310
	1					20	80	309
2	1	I	0.22	9	2	30	59	310
	1					20	80	309
3	2	E	1.98	1050	1	30	77	308
3	2	I	0.27	9	1	30	67	308

Table 2: Results of experiment 1

5 SOS: Synthesis of Multiprocessor Systems

We now shift our focus to multiprocessor systems. SOS is a method and mathematical model for synthesis of multiprocessor systems for given applications.

5.1 Overview of SOS

The SOS method is intended for design of the system architecture, which is the first step in the design of an application-specific multiprocessor system. The application domain is assumed to be specified in terms of a task data flow graph. The task data flow graph specifies a set of subtasks (nodes in the graph) that need to be performed and the data precedence between them (arcs in the graph). Given the task data flow graph, the goal is to synthesize a multiprocessor architecture which meets various cost and performance requirements and constraints. Synthesizing an architecture involves making decisions about the number and types of processing elements selected, the overall interconnection between the processing elements, and the scheduling of subtasks on the processing elements.

The SOS method produces a custom multiprocessor architecture, as well as mapping the subtasks onto the architecture and providing a schedule for the task execution. *It performs the automatic design of the multiprocessor*

architecture itself, not merely the mapping of tasks onto a given architecture. Another distinguishing feature is the fact that SOS produces a truly heterogeneous system, in terms of the functionality and the cost-speed characteristics of the processing elements, which allows a more precise tailoring of the synthesized architecture to a specific application. SOS can also be used to explore different interconnection styles; e.g., bus, point-to-point, ring, or a mixture of these. SOS assumes there is no global clock and communications between subtasks are asynchronous.

The approach used by SOS involves creation of a formal model of the multiprocessor synthesis problem using mathematical programming and the solution of this model. This approach is a natural outgrowth of the work described by Chu [6], Talukdar [37], and Hafer [17].

The problem inputs are: a task data flow graph which specifies the overall application task including the set of subtasks (nodes) to be executed, the data precedence requirements (arcs) and the volumes of data to be transferred between the subtasks; a set of processing elements with varying functionality, cost and performance; communication link characteristics with its cost and performance; constraints on total system cost; and constraints on timing of arbitrary events in the task execution.

The problem outputs are: a multiprocessor architecture including the chosen set of processing elements and the interconnection style for the elements; a schedule for the subtasks and their assignment to processing elements; and detailed timing information for computation and transfer of data.

A complete mathematical programming formulation of the problem requires specification of an objective function that has to be optimized and a set of constraints that have to be satisfied. The objective function can be whatever the designer wishes; e.g., the total system cost, or the overall system performance. The set of constraints consists of the constraints that must be satisfied for the overall task to be performed correctly as well as the arbitrary timing and cost constraints imposed by the designer. The constraints that must be satisfied for the overall task to be performed correctly consist primarily of the relations that ensure proper ordering of the subtasks and the data transfers, taking into account the timing involved and the relations that express the conditions for complete and correct system configuration. In order to express the various constraints and the objective function, we define certain variables related to the system. The necessary variables fall into two basic categories: the *timing variables* (real variables which represent timings of various critical events in the operation of the system) and the *binary variables* (0-1 variables which represent the implementation decisions regarding the system configuration).

For the point-to-point interconnection style, the SOS model involves

		Execution Time			
Proc.	Cost	S_1	S_2	S_3	S_4
p_1	4	1	1	-	3
p_2	5	3	1	2	1
p_3	2	-	3	1	-

Table 3: Execution Time and Cost Table

6 types of timing variables, 4 types of binary variables, and 12 types of constraints. The exact details of the model can be found in [44]. Several constraints comprising the model are non-linear relations. These relations are linearized and the model is converted into a MILP (Mixed Integer-Linear Programming) formulation. *Bozo* [2] [18] invokes a commercial linear programming package, XLP, developed by XMP Software, Inc. SOS uses this program to solve the MILP model.

In essence, the method creates a formal model for the multiprocessor synthesis problem. The mathematical model allows a deep understanding of the problem. Such an approach allows us to modify, extend and enhance the model to include more design possibilities and variations easily. The exact form of constraints used can be tailored to meet the characteristics of the design problem at hand. Also, the approach offers a great degree of flexibility in handling arbitrary constraints as they can be expressed using the timing and binary variables defined in the model. The MILP model has been solved for some example design problems. For small size problems, the model is solved fairly quickly. However, for larger problems it may be necessary to incorporate some heuristics into the branch-and-bound solution process.

5.2 A Synthesis Example Using SOS Method

An example data flow graph is shown in Figure 5. There are three types of processors available: p_1, p_2, and p_3. The costs of these processors and the execution times of various subtasks on the processors are given in Table 3. An entry of '—' in the table implies that the particular processor is functionally not capable of performing the particular subtask. As is obvious from the table, different processors have different cost-speed-functionality characteristics.

[2] A branch-and-bound program to solve an MILP problem which has been developed by L. J. Hafer of Simon Fraser Univ.

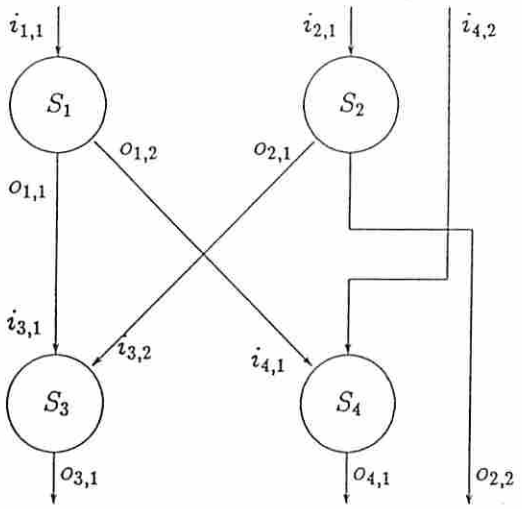

Figure 5: An Example Task Graph

For this example, architectures with point-to-point interconnection style are to be synthesized. The volume of data that needs to be communicated is one unit at each arc in the graph. The cost of a communication link is one unit; and the time taken in communicating a unit volume of data over the link is also one unit.

The MILP model for the example consists of 66 variables, 21 timing and 45 binary, and 94 constraints. Three non-inferior architectures were generated. These different architectures were generated by changing the constraint value for the total cost of the system, and optimizing the overall performance of the system. The total cost of the system is the sum of the costs of the processors selected and the costs of the links created. The performance of the system is the time at which the task is completed (all the subtasks are completed). Cost and performance for the three designs are given in Table 4. A brief discussion of these designs follows.

Design 1 consists of 3 processors: p_{1a} - a processor of type p_1, p_{2a} - a processor of type p_2, and p_{3a} - a processor of type p_3. Processor p_{1a} performs subtask S_1, processor p_{2a} performs subtasks S_2 and S_4 in that order, and processor p_{3a} performs subtask S_3. There are three communication links: $l_{1a,2a}$, $l_{1a,3a}$, and $l_{2a,3a}$. Data $i_{4,1}$ gets transmitted on link $l_{1a,2a}$, data $i_{3,1}$ gets transmitted on link $l_{1a,3a}$, and data $i_{3,2}$ gets transmitted on link $l_{2a,3a}$. As an illustration, this architecture is shown in Figure 6.

Design 2 consists of 2 processors: p_{1a} - a processor of type p_1, and p_{3a} -

Design	Cost	Performance
1	14	3
2	7	4
3	5	7

Table 4: Statistics for Non-Inferior Architectures

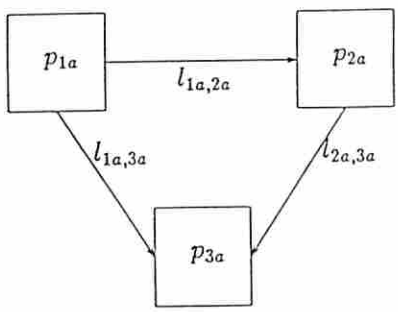

Figure 6: Synthesized Multiprocessor System I

a processor of type p_3. Processor p_{1a} performs subtasks S_1 and S_4 in that order, and processor p_{3a} performs subtasks S_2 and S_3 in that order. There is a communication link: $l_{1a,3a}$. Data $i_{3,1}$ gets transmitted on link $l_{1a,3a}$.

Design 3 consists of just 1 processor: p_{2a} - a processor of type p_2. The processor performs the subtasks S_2, S_1, S_4, and S_3 in that order.

Efforts are underway to automate the production of constraints from a high-level specification of the problem, and to devise some heuristics for branch-and-bound search.

6 3D Scheduling

As feature sizes decrease, the gap between functional unit delays and interconnection wiring delays is narrowing. Module assignment can affect the performance of the subsequent physical implementation greatly due to long interconnections between operators. "Optimal" register-transfer level schedules can actually be quite suboptimal when interconnection delays are ignored. Obviously, high-level synthesis tools using submicron technology will not be able to make intelligent scheduling decisions without considering interconnection delays. However, interconnection delays can be determined

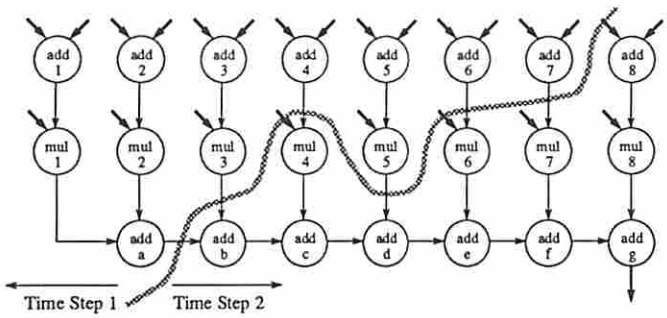

Figure 7: A 2-time-step Non-pipelined FIR Filter Design

accurately only after floorplanning is completed.

The goal of this task is to develop a new technique for scheduling, module allocation and module assignment called 3D scheduling[3] which incorporates interconnection delays during high-level synthesis process using floorplanning. Since interconnection delays are considered during scheduling, we can obtain a more realistic estimate for a design than traditional approaches, which may accept a design during high-level synthesis and then find that constraints are not satisfied in a later implementation.

6.1 Overview of 3D Scheduling

The core of datapath synthesis [35] is divided into three subtasks: scheduling, module allocation and module assignment. Scheduling assigns operations to appropriate time steps. Figure 7 shows an example of a dataflow graph used as input for high-level synthesis. Every node in the graph represents an operation. The scheduling of this dataflow graph into two time steps is shown with the cross-hatched line in the figure. A sufficiently large set of modules for each time step is then allocated. Modules may be shared among operations in different time steps. This task is called module allocation. The phase called module assignment binds the operations to specific modules. Most current datapath synthesis programs [41] [42] [40] only deal with scheduling and module allocation as a first step, but some also perform module assignment concurrently [36].

The basic idea behind this work is the following: as operations are scheduled and functional modules are allocated, we decide their shape and

[3] 3D scheduling refers to the problem of simultaneously scheduling time and the X-Y plane.

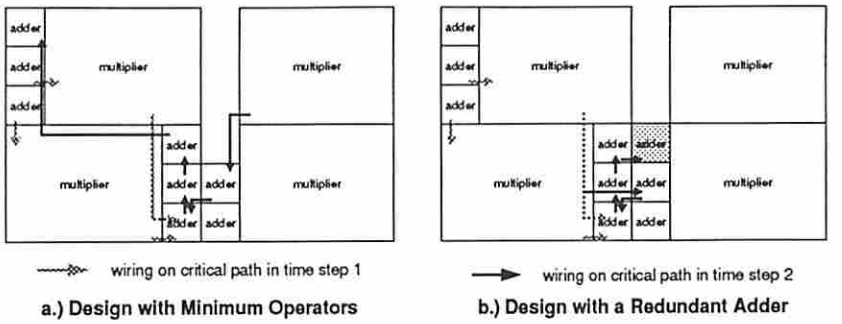

Figure 8: 2-time-step Non-pipelined FIR Filter Floorplans

position on the floorplan concurrently. The approach taken is to schedule the operations along the critical path first and assign (and place, if the assigned operator is newly allocated) operators as close as possible to their predecessor(s) according to their data dependencies without violating user specified constraints. Then, the off-critical path operations are scheduled and assigned to operators.

The technique at all times tries to minimize the interconnection length along the critical path(s). Once all the operations have been scheduled, a pairwise interchange improvement procedure is executed. *Redundant operators*, which are not required for the minimum feasible design, may be introduced during the improvement process. The redundant operators may be allocated to alleviate interconnection delays along the critical path. Interconnection delay is estimated by a first-order equation.

6.2 Results

We applied the 3D scheduling program to an FIR filter design, as shown in Figure 7. The 3D-scheduler generated a 2-time-step design, minimizing the number of operators as shown in Figure 8a. Another design with a redundant adder was created by the scheduler and is shown in Figure 8b.

For comparison purposes, we linearly scaled down each functional unit area and delay from 3-micron process parameters. We used different device parameters for different fabrication processes to calculate wiring delay in this program. Our experiments shows interconnection delay time contributes 20% of functional unit delay to overall delay with 1.6 micron fabrication process. The overall delay time could be reduced to 5% above functional delay by introducing redundant operators along the critical path.

To compare the differences between current floorplan packages and our program, we used a quadratic-based floorplan program [49] to generate a

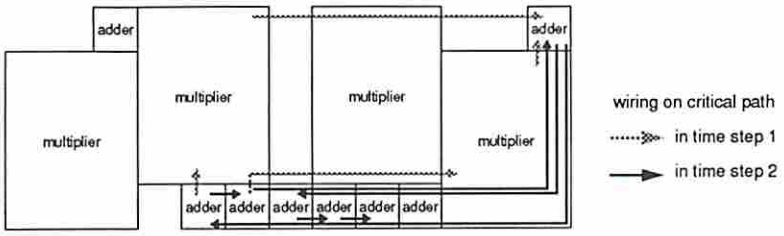

Figure 9: Floorplan Created by a Quadrisection-based Floorplan Program

2-time-step FIR filter floorplan, using sequential ordering for module assignment. The design is shown in Figure 9. Total wiring cost was optimized during floorplanning. However, since floorplanning was performed without taking into account the wiring along the critical path, the critical path wiring runs back and forth in this case[4]. One of our examples indicates a design using sequential module assignment followed by a quadratic-based floorplanning technique results in increased delay time of 10% as compared to our technique. The example was rerun with an actual scaled module library to confirm our results.

For finer feature sizes, the interconnection delay plays a more important role, due to more complex designs fitting into one chip and the higher system performance required. Redundant operator allocation becomes a feasible solution due to the smaller area of operators. Our predictions show the interconnection delays will become dominant when submicron fabrication process are used.

References

[1] M. Barbacci and D. Siewiorek. The cmu rt-cad system: An innovative approach to computer aided design. In *American Federation of Information Processing Societies Conference Proceedings, Vol. 45*, pages 643–655. Amer. Fed. of Information Processing Societies, June 1976.

[2] J. Bhasker and S. Sahni. A Linear Algorithm to Find a Rectangular Dual of a Planar Triangulated Graph. In *Proc. of 23th Design Automation Conference*, June 1986.

[3] W. C. Brantley. *Automatically Decomposing Signal Processing Applications on Multiprocessors.* PhD thesis, Carnegie-Mellon University, 1979.

[4] In all fairness, pairwise interchange might improve the performance of this design. However, we left the floorplan "as is" to illustrate the impact of totally ignoring timing.

[4] F. D. Brewer and D. D. Gajski. A System for Constraint Driven Behavioral Synthesis. *IEEE Transactions on Computer-Aided Design*, CAD-9(7), July 1990.

[5] M. Burstein and M. Youssef. Timing Influenced Layout Design. In *Proc. of 22nd Design Automation Conference*, June 1985.

[6] W. Chu, L. Hollaway, M. Lan, and K. Efe. Task Allocation in Distributed Data Processing. *Computer*, 13(11):57–69, November 1980.

[7] E. Dirkes. Architectural Partitioning for System Level Design. In *Proceedings 26th Design Automation Conference*. ACM/IEEE, June 1989.

[8] G. Estrin. A methodology for design of digital systems - supported by SARA at the age of one. In *Proceedings of National Computer Conference*, volume 47, pages 313–324. NCC, 1978.

[9] A. Dunlop et. al. Chip Layout Optimization Using Critical Path Weighting. In *Proc. of 21th Design Automation Conference*, June 1984.

[10] W. Donath et. al. Timing Driven Placement Using Complete Path Delays. In *Proc. of 27th Design Automation Conference*, June 1990.

[11] E. Girczyc. *Automatic Generation of Microsequenced Data Paths to Realize ADA Circuit Descriptions.* PhD thesis, Department of Electronics, Carleton University, July 1984.

[12] J.J. Granacki and A.C. Parker. The effect of register-transfer design tradeoffs on chip area and performances. In *Proceedings of the 20th Design Automation Conference*, June 1982.

[13] Pravil Gupta. PLA and Wire Delay Analysis. Internal Report, in progress.

[14] Rajesh Gupta and Giovanni De Micheli. Partitioning of Functional Models of Synchronous Digital Systems. In *Proceedings International Conference on Computer Aided Design.* ACM/IEEE, November 1990.

[15] E. K. Haddad. Optimal Load Allocation for Parallel and Distributed Processing. Technical Report TR 89-12, Department of Computer Science, Virginia Polytenic Institute and State University, April 1989.

[16] L. Hafer and A. Parker. Automated synthesis of digital hardware. *IEEE Transactions on Computers*, C-31(2):93–109, February 1981.

[17] L. Hafer and A. Parker. A Formal Method for the Specification, Analysis, and Design of Register-Transfer Level Digital Logic. *IEEE Transactions on Computer-Aided Design*, CAD-2(1), January 1983.

[18] L. J. Hafer. Bringing up Bozo. Technical report, Department of Computing Science, Simon Fraser University, May 1990.

[19] B. S. Haroun and M. I. Elmasry. Architectural synthesis for DSP silicon compilers. *IEEE Trans. CAD.*, 8(4), April 1989.

[20] S. Hayati. *The Synthesis of Control-Dominated Application-Specific Integrated Circuits Using Goal-Based Design Management.* PhD thesis, Department of Electrical Engineering, University of Southern California, December 1990.

[21] C. E. Houstis. Module Allocation of Real-Time Applications to Distributed Systems. *IEEE Transactions on Software Engineering*, 16(7):699–709, July 1990.

[22] C. Hwang, Y. Hsu, and Y. Lin. Optimum and Heuristic Data Path Scheduling Under Resource Constraints. In *Proceedings 27th Design Automation Conference*, pages 65–70. ACM/IEEE, June 1990.

[23] T. Barnwell III and D. Schwarz. Optimal implementation of flow graphs on synchronous multiprocessors. In *Int. Conf. on ASSP 84*, 1984.

[24] B. Indurkhya, H. S. Stone, and L. Xi-Cheng. Optimal Partitioning of Randomly Generated Distributed Programs. *IEEE Transactions on Software Engineering*, SE-12(3):483–495, March 1986.

[25] R. Jain, K. Kucukcakar, M. J. Mlinar, and A. C. Parker. Experience with the ADAM Synthesis System. In *Proceedings 26th Design Automation Conference.* ACM/IEEE, June 1989.

[26] R. Jain, M. J. Mlinar, and A. C. Parker. Area-Time Model for Synthesis of Non-Pipelined Designs. In *Proceedings International Conference on Computer Aided Design.* ACM/IEEE, November 1988.

[27] R. Jain, A. C. Parker, and N. Park. Predicting Area-Time Tradeoffs for Pipelined Designs. In *Proceedings 24th Design Automation Conference.* ACM/IEEE, June 1987.

[28] B. W. Kernighan and S. Lin. An Efficient Heuristic Procedure for Partitioning Graphs. *Bell System Technical Journal*, 49(1), January 1970.

[29] D. Knapp. Feedback-Driven Datapath Optimization in Fasolt. In *Proc. of the ICCAD-90 Conference*, November 1990.

[30] K. Kucukcakar and A. C. Parker. BAD: Behavioral Area-Delay Predictor. Technical Report CENG-90-31, Department of Electrical Engineering, University of Southern California, November 1990.

[31] K. Kucukcakar and A. C. Parker. CHOP: A Constraint-Driven System-Level Partitioner. Technical Report CENG-90-26, Department of Electrical Engineering, University of Southern California, November 1990.

[32] F. J. Kurdahi and A. C. Parker. PLEST: A Program for Area Estimation of VLSI Integrated Circuits. In *Proceedings 23rd Design Automation Conference*. ACM/IEEE, June 1986.

[33] Y. Lai and S. Leinwand. Algorithms for Floorplan Design Via Rectangular Dualization. *IEEE Trans. on Computer-Aided Design*, Dec 1988.

[34] Richard I. Levin and Charles A. Kirkpatrick. *Planning and Control with PERT/CPM.* McGraw Hill, 1966.

[35] A. Parker M. McFarland and R. Camposano. Tutorial on High-Level Synthesis. In *Proc. of the 25th Design Automation Conference*, July 1988.

[36] M. C. McFarland. Using Bottom-Up Design Techniques in the Synthesis of Digital Hardware from Abstract Behavioral Descriptions. In *Proceedings 23rd Design Automation Conference*. ACM/IEEE, June 1986.

[37] R. Mehrotra and S. Talukdar. Task Scheduling on Multiprocessors. Technical Report DRC-18-55-82, Department of Electrical Engineering, Carnegie-Mellon University, December 1982.

[38] Mitch J. Mlinar. *System Level Tradeoffs in VLSI Design.* PhD thesis, Department of Electrical Engineering, University of Southern California, 1991. In progress.

[39] R. Nair and et. al. Generation of Performance Constraints for Layout. *IEEE Trans. on Computer-Aided Design*, August 1989.

[40] N. Park and A. Parker. Sehwa: A Software Package for Synthesis of Pipelines from Behavioral Specifications. *IEEE Trans. on Computer-Aided Design*, Mar 1988.

[41] A. Parker, J. Pizarro, and M. Mlinar. MAHA: A Program for Datapath Synthesis. In *Proc. of the 23th Design Automation Conference*, July 1986.

[42] P. Paulin and J. Knight. Algorithms for High-Level Synthesis. *IEEE Design & Test of Computers*, Dec 1989.

[43] Thomas S. Payne and W. M. vanCleemput. Automated Partitioning of Hierarchically Specified Digital Systems. In *Proceedings 19th Design Automation Conference*. ACM/IEEE, June 1982.

[44] S. Prakash and A. C. Parker. Synthesis of Application-Specific Multiprocessor Architectures. Technical Report 90-25, Department of Electrical Engineering, University of Southern California, November 1990.

[45] Jan Rabaey and Hugo De Man. Computer aided design of digital signal processing systems. In *Proc. ICCD*, 1987.

[46] Martin L. Resnick. SPARTA : A System Partitioning Aid. *IEEE Transactions on Computer-Aided Design*, CAD-5(4), October 1986.

[47] Brian Schmult. Partitioning Strategies for Multi-chip VLSI Microprocessors. Technical Report CMUCAD-84-26, Department of Electrical and Computer Engineering, Carnegie-Mellon University, February 1984.

[48] H. S. Stone. Critical Load Factors in Two-processor Distributed Systems. *IEEE Transactions on Software Engineering*, SE-4:254–258, May 1978.

[49] P. Suaris and G. Kedem. A Quadrisection-Based Combined Place and Route Scheme for Standard Cells. *IEEE Trans. on Computer-Aided Design*, Mar 1989.

[50] T. Wang and D. Wong. An Optimal Algorithm for Floorplan Area Optimization. In *Proc. of 27th Design Automation Conference*, June 1990.