Unified System Construction

Chih-Tung Chen, Pravil Gupta & Alice C. Parker

CENG Technical Report 93-32

Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, California 90089-2562
(213)740-4476

August 1993

Unified System Construction*

Alice C. Parker, Chih-Tung Chen and Pravil Gupta
Department of Electrical Engineering - Systems
University of Southern California
EEB 300, MC 2562
Los Angeles, CA 90089-2562, USA

Abstract

This paper describes the Unified System Construction project underway at the University of Southern California. The goal of the project is to automate the construction of heterogeneous, application-specific systems. Key elements of the USC system include multiprocessor synthesis, multi-chip data path synthesis, memory-intensive synthesis, and multi-chip partitioning.

1 Introduction

Communications, entertainment, and other electronic systems are in widespread use. These systems are generally multi-chip, heterogeneous, and application-specific. Chip-level synthe-

[0]This work was supported by the Defense Advanced Research Projects Agency and monitored by the Federal Bureau of Investigation under Contract No. JFBI90092. The views and conclusions considered in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

To appear in Proceedings of the Fourth SASIMI Workshop '93, Synthesis and Simulation Meeting and International Interchange, Nara, Japan, Oct., 1993.

sis tools are invaluable for the rapid production of such systems, and such tools are becoming available for general use. System-level tools can also be used to significantly increase a designer's ability to meet a schedule along with a set of performance and cost constraints, but few of these tools have been available in the past.

The Unified System Construction (USC) project at the University of Southern California involves the production of an integrated set of system-level tools for synthesizing multi-chip, heterogeneous application-specific systems which meet cost, performance and power constraints. The focus of the USC project is on real-time systems, such as entertainment and communication technologies, but does not exclude other applications requiring specialized system design. A block diagram of the system is shown in Figure 1.

The user of the USC software first selects a style for the system. Styles currently supported include

- heterogeneous multiprocessors consisting of
  - processors interconnected with point-to-point connections, operating in a non-pipelined fashion[1],
  - non-pipelined processors in a ring,
  - non-pipelined processors connected by a bus,
  - pipelined processors with point-to-point connections,
- multiple custom VLSI chips, communicating asynchronously,
- multiple custom VLSI chips, communicating synchronously, with common clock, and
- memory-intensive modules consisting of a custom VLSI chip and a separate memory chip.

[1]Each processor can be pipelined but execution between processors is not pipelined.

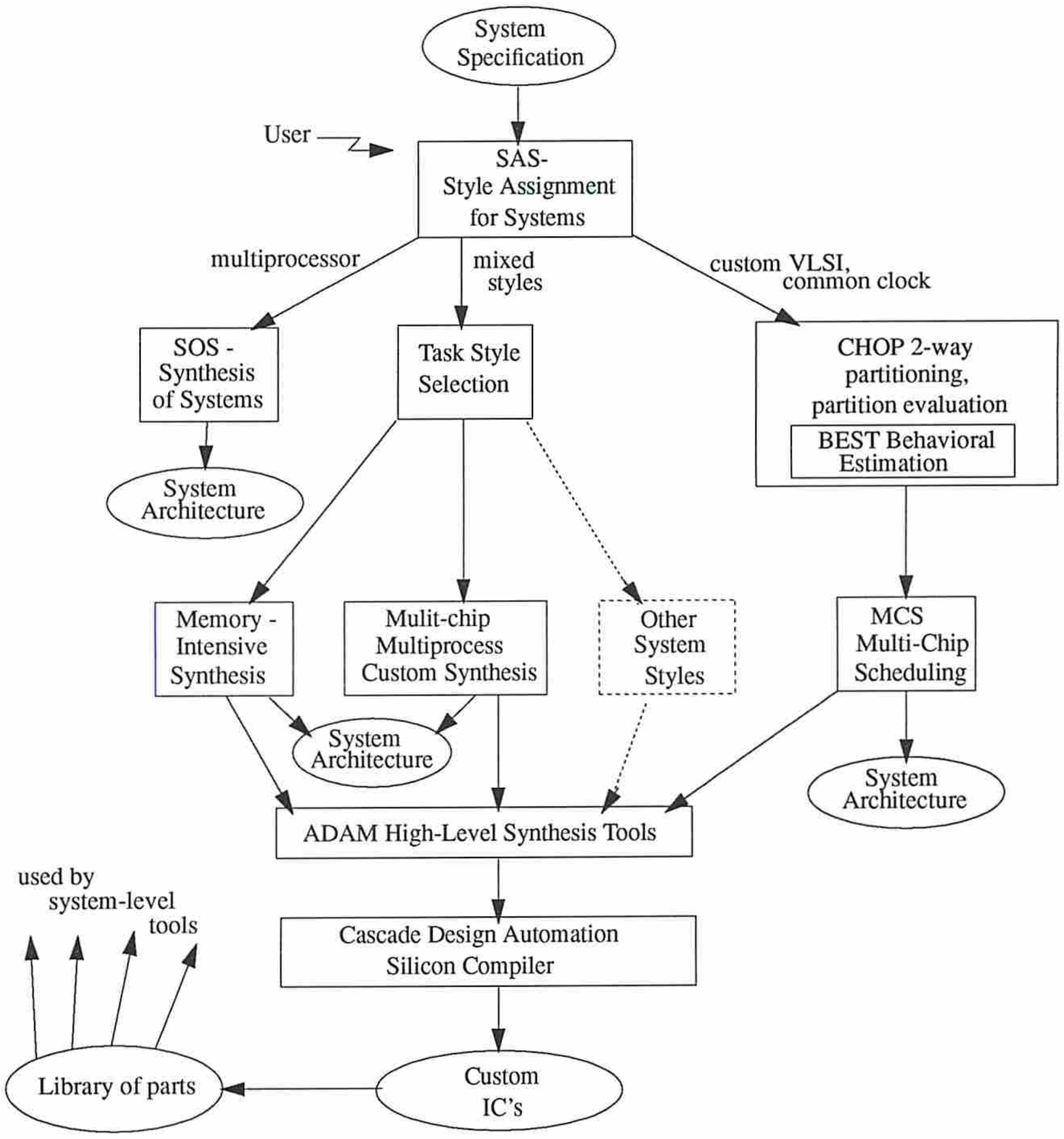
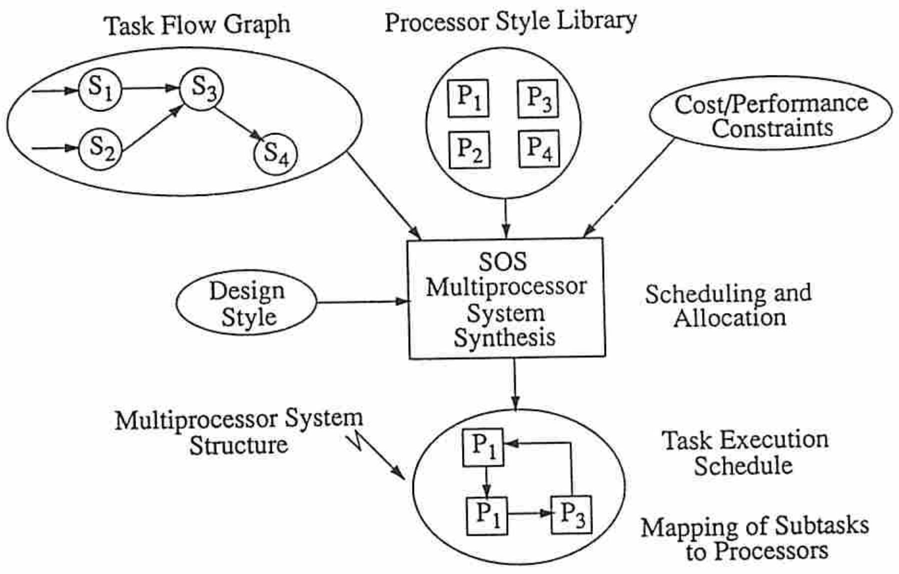

System Specification
User
SAS- Style Assignment for Systems
multiprocessor
mixed styles
custom VLSI, common clock
SOS - Synthesis of Systems
Task Style Selection
CHOP 2-way partitioning, partition evaluation
BEST Behavioral Estimation
System Architecture
Memory - Intensive Synthesis
Mulit-chip Multiprocess Custom Synthesis
Other System Styles
MCS Multi-Chip Scheduling
System Architecture
System Architecture
ADAM High-Level Synthesis Tools
used by system-level tools
Cascade Design Automation Silicon Compiler
Library of parts
Custom IC's

Figure 1: Block diagram of the Unified System Construction (USC) Project

Many other styles of systems are currently under development. Once a style is selected, specialized tools are invoked to complete the design process. Ultimately, any custom VLSI chips which must be synthesized are then processed by the ADAM high-level synthesis system, which produces a cell netlist. This netlist is input to the Cascade Design Automation Chipcrafter Silicon Compiler, and a chip layout is produced.

The remaining sections give an overview of each major style of design, and describe examples of systems synthesized with each set of tools.

2 Multiprocessor Synthesis



Task Flow Graph
S1
S2
S3
S4
Processor Style Library
P1
P3
P2
P4
Cost/Performance Constraints
Design Style
SOS Multiprocessor System Synthesis
Scheduling and Allocation
Multiprocessor System Structure
P1
P1
P3
Task Execution Schedule
Mapping of Subtasks to Processors

Figure 2: Overall operation of the SOS tools

Multiprocessors of various styles can be synthesized using the SOS (Synthesis of Systems) set of tools. The input to SOS is a specification in the form of a task flow graph. SOS decides on the number and type of processors to be used, the interconnections between them, and the schedule of execution of tasks onto processors. The tools all use mixed integer-linear programming (MILP) to model the synthesis problem. Some of the tools can generate the MILP constraints automatically, and others currently rely on constraints written by the user.

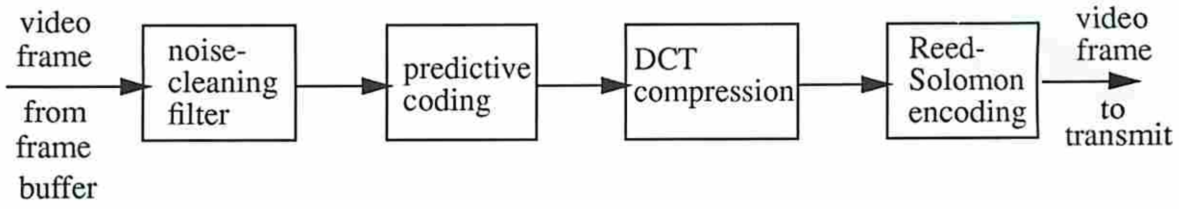


Figure 3: Example system specification (only one channel shown)

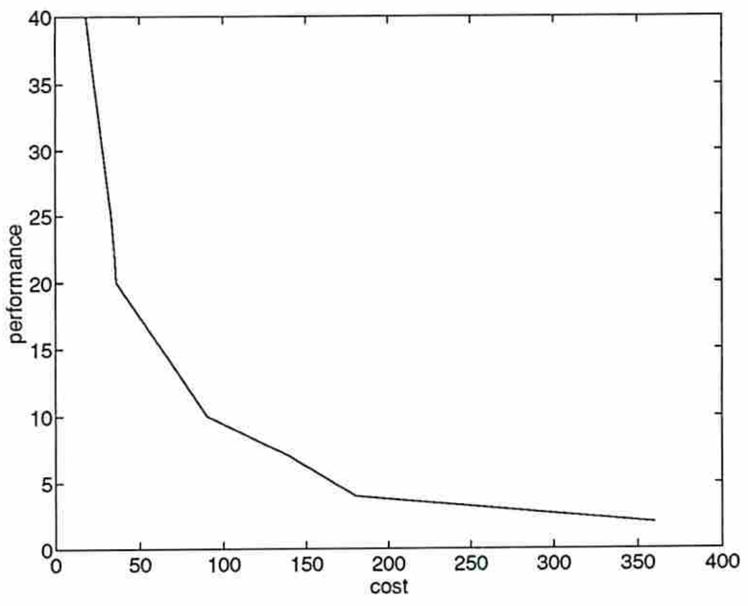


Figure 4: Family of designs synthesized by SOS

The tools rely on a branch-and-bound solver called BOZO (available from L. Hafer at Simon Fraser U. in Burnaby, B.C. Canada). Figure 2 shows the overall operation of the SOS tools. An example system specification is shown in Figure 3 and a set of designs synthesized by SOS are shown in Figure 4. Further information about SOS can be found in [11] [13] and [12].

3 Synthesis of Common-Clock Multi-Chip Systems

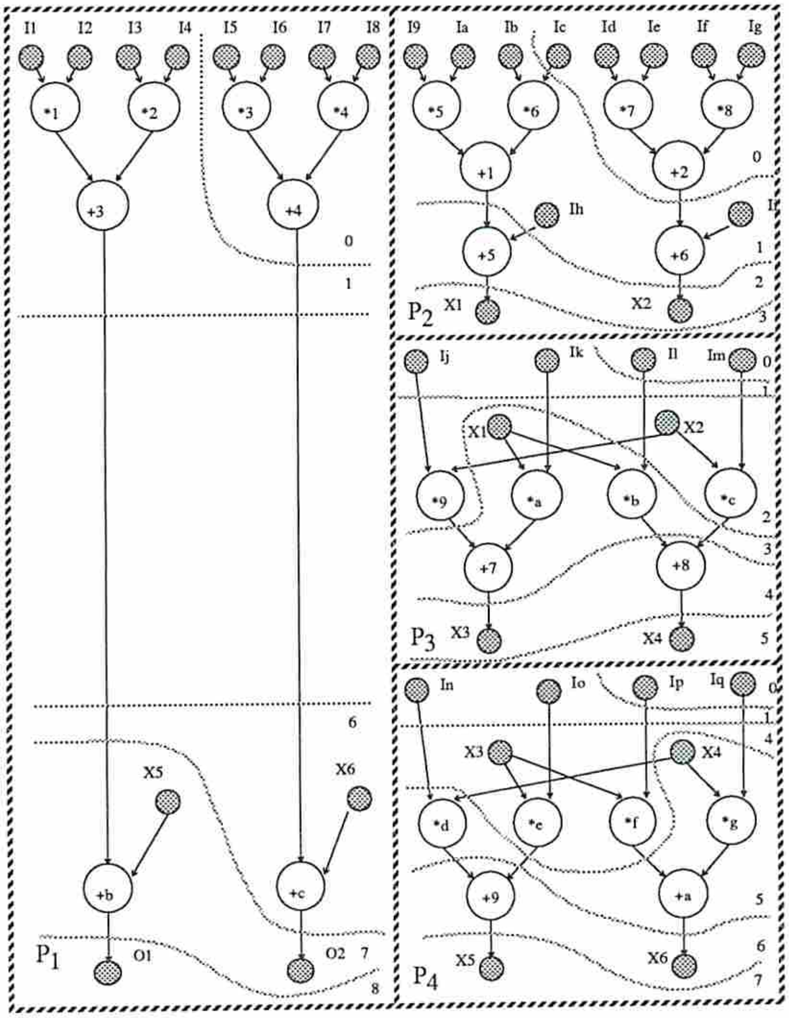


Figure 5: Output schedule of the partitioned AR filter produced by MCS

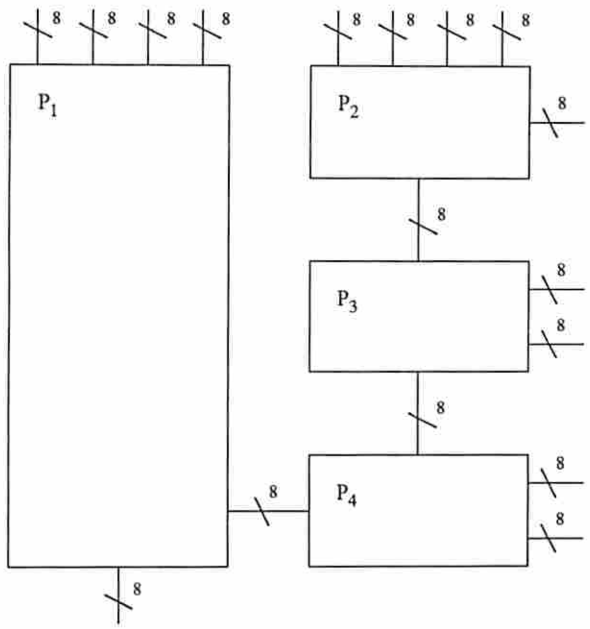


Figure 6: Interconnect structure synthesized by MCS

The use of multiple-chip systems with a common clock is restricted to those cases where several chips are required to provide adequate processing capability to meet performance requirements, generally for real-time, pipelined computations, like signal and video processing. This style is well-suited to multi-chip modules (MCMs).

The USC system contains three packages for synthesis of multi-chip systems with common clocks. CHOP is a partition evaluation package which will determine the feasibility of a given behavioral partitioning, given the number of chips allowed, and packaging information. BEST is a behavioral estimation package invoked by CHOP to predict the performance and die size of chips, given a partitioning of the behavioral specification. MCS is a multi-chip scheduling package which takes a set of partitions of the behavior and performs data path and pin scheduling. MCS also synthesizes the interchip interconnection structure required to realize the schedule it produces.

At present, CHOP can perform a two-way automatic partitioning. If the designer has a different partitioning in mind, he or she can present the partitioning to CHOP for evaluation. CHOP performs a selection of module styles for every operation in the input specification (e.g. carry-look-ahead adder), and determines whether a given partition is feasible by using BEST to predict the cost and performance of all possible designs using implicit enumeration. CHOP takes into account the contributions to chip area and delay caused by operators, registers, multiplexers, control, wiring and memories. CHOP ensures that there are adequate pins to transfer data between chips, and that a feasible schedule exists for such data transfers.

MCS uses the partitions evaluated by CHOP to produce a pipelined schedule for operations and interchip data transfers, making sure that there are adequate pins, and scheduling transfers onto pins. A partition input to MCS and the schedule produced by MCS are shown in Figure 5 and the interconnect is shown in Figure 6.

More information about CHOP, BEST and MCS can be found in [9, 6, 8].

4 Synthesis of Memory-Intensive Systems

This research focuses on the automatic synthesis of memory-intensive application-specific systems, with emphasis on hierarchical storage architecture design [4, 5]. The storage architecture is closely connected to the data path of the system, and isolating its synthesis from data path synthesis may not result in an efficient solution. Therefore, the design of the data path and storage architecture must be coordinated.

Our problem is to perform high-level synthesis of an integrated system consisting of a data path and a two-level memory hierarchy, Figure. 7, from a given behavioral specification and with constraints on cost and performance.

1. On-chip foreground memory. This consists of two subparts: datapath memory to store the intermediate or temporary variables in the data path, and I/O buffers to

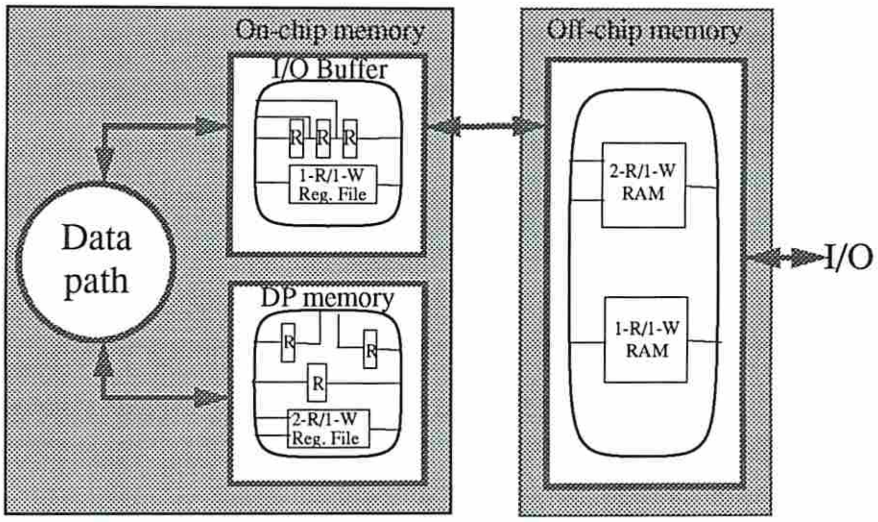


Figure 7: Target system architecture

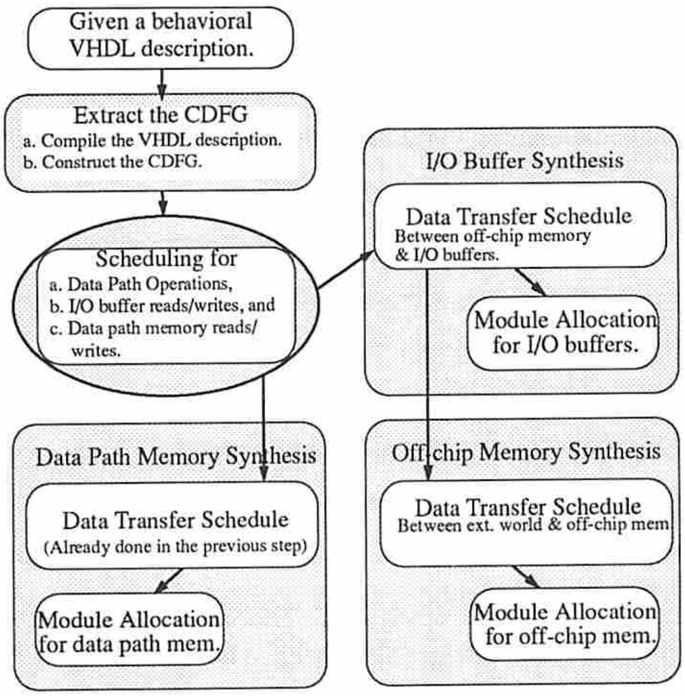
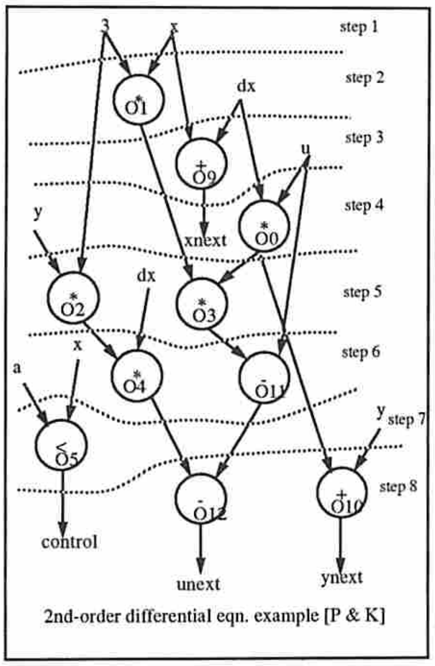

Given a behavioral VHDL description.
Extract the CDFG
a. Compile the VHDL description.
b. Construct the CDFG.
Scheduling for
a. Data Path Operations,
b. I/O buffer reads/writes, and
c. Data path memory reads/ writes.
I/O Buffer Synthesis
Data Transfer Schedule
Between off-chip memory & I/O buffers.
Module Allocation for I/O buffers.
Data Path Memory Synthesis
Data Transfer Schedule
(Already done in the previous step)
Module Allocation for data path mem.
Off-chip Memory Synthesis
Data Transfer Schedule
Between ext. world & off-chip mem
Module Allocation for off-chip mem.

Figure 8: Overall synthesis approach.



<u>Input data transfer schedule for on-chip I/O buffer</u>

```
STEP 1 --> WRITEs (1) --> x
           Memory Contents -> x
           READs (0) --> None
STEP 2 --> WRITEs (1) --> const3
           Memory Contents -> const3 x
           READs (2) --> x const3
STEP 3 --> WRITEs (1) --> dx
           Memory Contents -> x dx const3
           READs (2) --> x dx
STEP 4 --> WRITEs (1) --> u
           Memory Contents -> u x dx const3
           READs (2) --> dx u
STEP 5 --> WRITEs (1) --> y
           Memory Contents -> y x dx const3
           READs (2) --> const3 y
STEP 6 --> WRITEs (1) -->u
           Memory Contents -> u x dx
           READs (2) --> u dx
STEP 7 --> WRITEs (1) --> a
           Memory Contents -> a x
           READs (1) --> a x
STEP8 --> WRITEs (1) --> y
           Memory Contents -> y
           READs (1) --> y

MEMORY SIZE = 4
```

Figure 9: Output from memory-synthesis system.

temporarily store the inputs and outputs to the data path chip, allowing fast access and off-chip storage of the data.

2. Off-chip background memory. In our model this is the bulk storage required for the inputs and outputs. All the I/O data values from/to the external world are stored here.

The I/O buffers and data path memory on the chip can be made up of registers, register files, RAMs or their combination. We assume that the off-chip bulk storage is constructed using larger modules like RAMs. In an extreme case, a memory can degenerate into wires with no cost and no storage capability.

We perform a deterministic, worst-case analysis to ensure that the data required by the data path is prefetched onto the chip before it is required. Data produced is transferred to off-chip memory at an appropriate time, overlapped with the execution of the data path before it is required elsewhere. Because we are dealing with fixed algorithms running on our application-specific hardware, and because our applications generally have hard real-time constraints, our analysis must of necessity be deterministic and worst case, even in the presence of conditional branches and loops.

The synthesis is performed in the following two steps: First, data path synthesis with operation scheduling is performed combined with scheduling of local data transfers to/from memory. As a result of this scheduling, constraints are placed on the memory structure. During the second step, the storage hierarchy design is completed, which includes determining the data transfers between different levels of memory hierarchy and completing synthesis of the storage structures. We ensure that each step of the stepwise construction of the system takes into account the next step by looking ahead so that the next step is not overly constrained. Global design parameters, like the memory bandwidth and timing constraints, are considered when constructing the partial design in each step, tying the whole synthesis process together. The overall synthesis approach is shown in Figure 8.

Inputs to the synthesis system: User provides the behavioral VHDL description of the system[2], the area/performance constraints, the functional module library with their area/delay characteristics, storage module library, timing constraints on I/O data values, and bandwidth constraints between the on-chip and off-chip memory and clock cycle.

Outputs of the synthesis system: For the *data path* the software determines the number of functional units of each type required for the design and the assignment of all the operations in the control-data flow graph to appropriate control steps and to modules. The software also synthesizes a *two level memory hierarchy* as described above. The software decides the number of read/write ports on both the on-chip I/O buffers as well as the data path memory while minimizing the sizes of these subparts. It also produces all the data transfer schedules between various subparts: (i) between the on-chip I/O buffers and the

[2]The algorithmic description may contain inner loops and conditional branches, however the loop structure is assumed to be already transformed and optimized.

data path, (ii) between the data path and datapath-memory for the intermediate variables produced and consumed during data path execution, and (iii) between the on-chip and the off-chip memory for I/O values.

4.1 An example

We applied our approach to a representative example (using a prototype version of the software): a second-order differential equation example [10]. Figure 9 shows the outputs from our prototype which includes the data path schedule and the input data transfer schedule for the on-chip I/O buffers. The design was synthesized with 1 adder, 1 subtractor, 2 multipliers, 1 comparator, 2 read and 2 write ports on the I/O buffers and a bandwidth of 1 word/cycle between the on-chip and off-chip memory.

4.2 Ongoing work

We are continuing researching this area and are adding more features to our software in order to make it more versatile and complete. We are going to integrate the software with the USC system and provide an interface for a commercial silicon compiler to generate the final layouts of the designs. We are also in a process of developing a memory-intensive example application, which we will design using our software, layout using the commercial silicon compiler and analyze to show the impact of this research.

5 Synthesis of Asynchronous Multi-chip Systems

In practice, we find that DSP and other ASIC designs consist of multiple concurrent and interacting processes. Though high-level synthesis has received enormous attention over the years, most approaches were concentrated on synthesizing single process (one thread

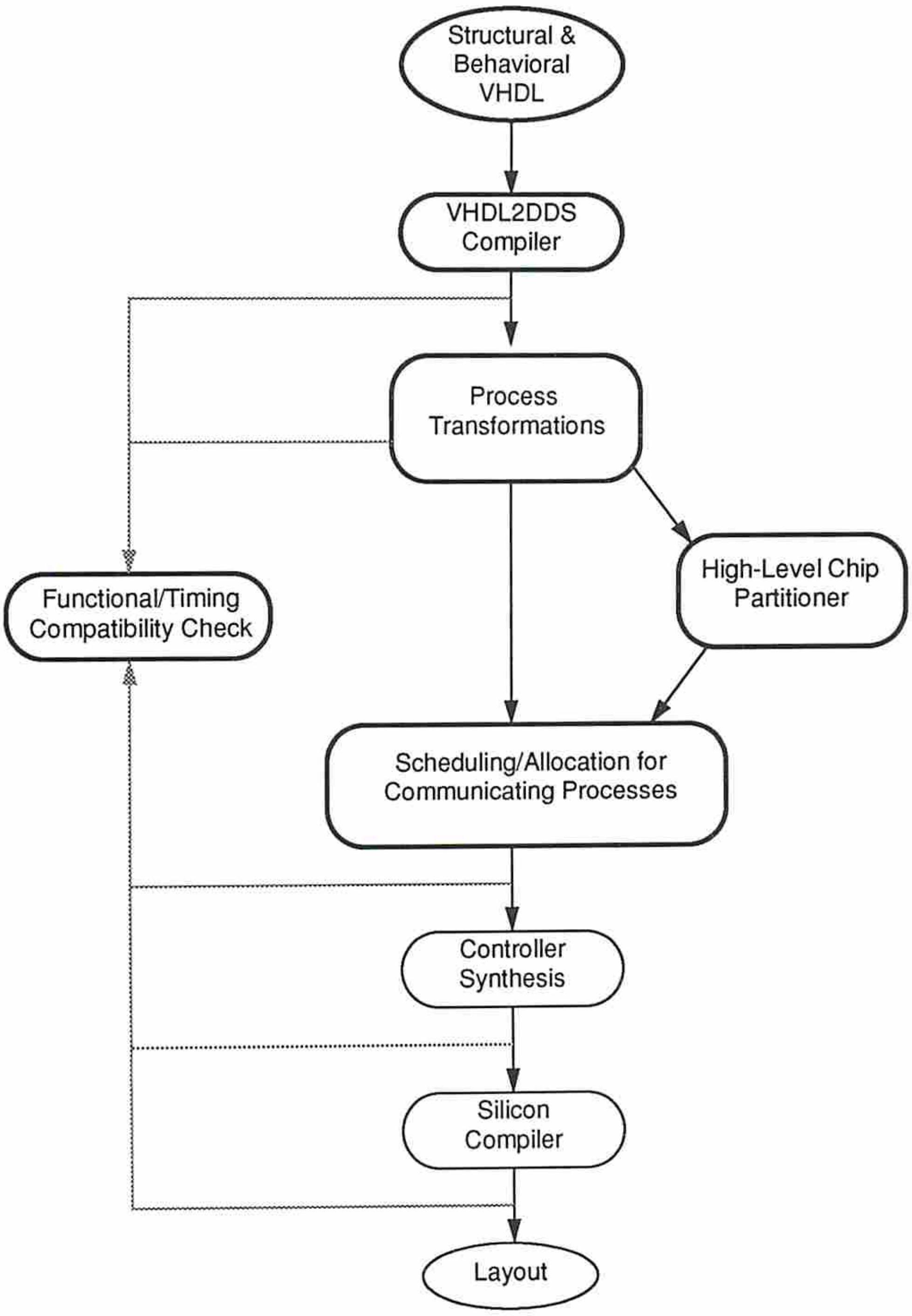

Structural & Behavioral VHDL
VHDL2DDS Compiler
Process Transformations
High-Level Chip Partitioner
Functional/Timing Compatibility Check
Scheduling/Allocation for Communicating Processes
Controller Synthesis
Silicon Compiler
Layout

Figure 10: A flow chart of the synthesis system for asynchronous multi-chip designs

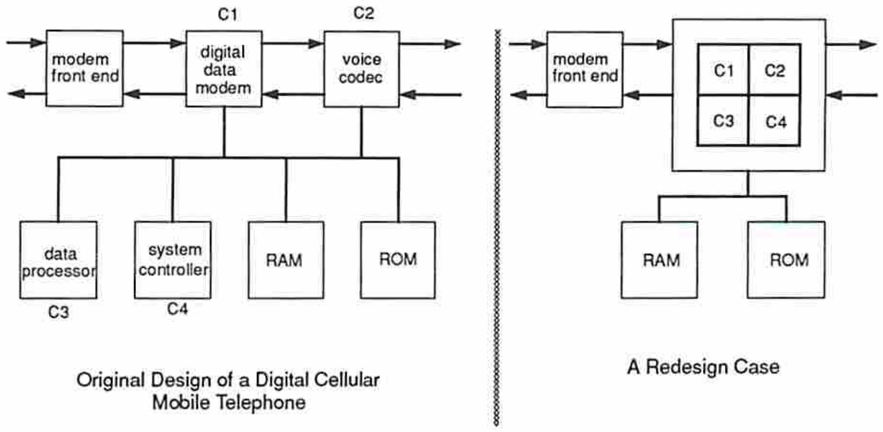


Figure 11: A digital cellular mobile telephone example

of control) designs. Synthesizing a design with multiple concurrent processes poses many new challenges. For example, since the processes interact with each other, the synthesis tool has to concurrently solve all the timing constraints imposed by one process on another. Furthermore, resource allocation for each process on a chip cannot be done without taking into account the area versus performance characteristics of each process since the total resources taken by all processes on a chip are limited by the chip package.

The goals of this research are to provide an integrated system for synthesizing multi-chip designs with multiple concurrent processes as well as to speed up the redesign of multi-chip systems. Figure 10 shows a flow chart which illustrates the approach used in this synthesis system. First, the multi-process specification is translated from VHDL to a synthesizable representation called the Design Data Structure (DDS) [1]. The next step is to perform a number of process transformations in order to trade off among hardware sharing, control complexity, communication overhead and cost. A process-level chip partitioner is then used to find new cost-effective chip boundaries according to the up-to-date packaging library. In addition, the partitioner will distribute chip resources to the processes according to their performance-versus-area characteristics and determine the interconnection structure as well. Next, a concurrent approach for multiple-process synthesis is used to synthesize each process into its own data path and control path. Most important is that those timing, area and performance constraints as well as the communication synchronization will all be met by the solution. Finally, we use a hybrid symbolic verifier to check the functional and timing compatibility of the new implementation [2]. The RTL implementation is submitted to the ADAM system to obtain the final chip layout.

Figure 11 illustrates a typical use of our synthesis system. In order to make a mobile phone system more compact and perform better, we synthesize an ASIC which contains the functionalities of $C1$, $C2$, $C3$ and $C4$ and has a shorter turnaround time. Instead of doing physical merging, which may offer no significant improvements on performance, functionality, or even cost, our synthesis system may choose to merge the behaviors of $C1$ and $C2$, which both are DSP-type processes, during the process transformation step in order to improve data path utilization. Also, since $C1$, $C2$, $C3$ and $C4$ are to be reimplemented on the same chip, the inter-chip communication among them becomes unnecessary and can be modified to give a more efficient on-chip communication scheme with relaxed timing constraints. The chip partitioner in our system can select a new cost-effective packaging device from all the available ones and distribute the chip resources to the processes according to their constraints. In addition, our multi-process synthesis will not only ensure the synchronization of the inter-process communication but also meet the new performance and area constraints.

Currently, the prototype chip partitioning tool was successfully used to experiment with the mobile phone example and the GM powertrain control application [3]. Our multi-process synthesis approach has been shown to offer better results than the Ku's method [7] in handling designs with unbounded delays while additionally capable of trading of performance and cost, synchronizing inter-process communication and reducing the control cost. We also found that our hybrid symbolic approach for verifying synthesized designs is both effective and efficient. In fact, its experiments helped to identify some problems of the early control signal generator tool in the ADAM system. Our ongoing work is geared toward integrating individual work together, performing top-down experiments, and continuing the research and development of process transformations, generalized communication models and heuristic approaches for multi-process synthesis.

References

[1] Chih-Tung Chen and Alice C. Parker. VHDL2DDS: A VHDL Language to DDS Data Structure Translator. Technical Report CEng 91–21, Department of EE-Systems, University of Southern California, July 1991.

[2] Chih-Tung Chen and Alice C. Parker. A Symbolic Approach for Checking Functional and Timing Compatibility of Synthesized Designs. Technical Report CEng 93–26, Department of EE-Systems, University of Southern California, May 1993.

[3] Thomas E. Fuhrman. Industrial Extensions to University High Level Synthesis Tools: Making it Work in the Real World. In *28th ACM/IEEE Design Automation Conference*, 1991.

[4] P. Gupta and Alice C. Parker. Towards Synthesizing Memory Architectures for Application-Specific Systems. Technical Report 92-05, Department of EE-Systems, University of Southern California, 1992.

[5] P. Gupta and Alice C. Parker. HISS: A Prototype Program for Hierarchical Storage Synthesis. Technical Report 93-07, Department of EE-Systems, University of Southern California, 1993.

[6] Y-H Hung. *High-Level Synthesis with Pin Constraints for Multiple-Chip Designs.* PhD thesis, Department of EE-Systems, University of Southern California, December 1992.

[7] David Ku and G. De Micheli. Relative Scheduling under Timing Constraints. In *27th ACM/IEEE Design Automation Conference*, 1990.

[8] K. Küçükçakar. *Partitioning of Digital System Specifications.* PhD thesis, Department of Electrical Engineering, University of Southern California, 1991.

[9] K. Kucukcakar and A. C. Parker. CHOP - A Constraint-Driven System-Level Partitioner. In *Proceedings of the 28th Design Automation Conference.* ACM/IEEE, June 1991.

[10] P.G. Paulin and J.P. Knight. Force-Directed Scheduling for the Behavioral Synthesis of ASICs. *IEEE Tran. on Computer Aided Design*, pages 661–679, June 1989.

[11] S. Prakash. *Synthesis of Application-Specific Multiprocessor Systems.* PhD thesis, Department of EE-Systems, University of Southern California, January 1993.

[12] S. Prakash and A. C. Parker. Synthesis of Application-Specific Multiprocessor Architectures. In *Proceedings 28th Design Automation Conference*, pages 8–13. ACM/IEEE, June 1991.

[13] S. Prakash and A. C. Parker. SOS: Synthesis of Application-SPecific Heterogeneous Multiprocessor Systems. *Journal of Parallel and Distributed Computing*, 16:338–351, December 1992.