Rapid Synthesis of Multi-Chip Systems

Dong-Hyun Heo, C.P. Ravikumar and Alice Parker

CENG 96-25

Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, California 90089-2562
(213) 740-6709

October 18, 1996

To appear in *10th International Conference on VLSI Design*

Rapid Synthesis of Multi-Chip Systems

Dong-Hyun Heo and Alice Parker *
Department of EE-Systems
University of Southern California
Los Angeles, CA 90089, USA

C.P. Ravikumar
Department of Electrical Engineering
Indian Institute of Technology
Delhi, India

Abstract

There is increasing demand for multi-chip implementations of a complex digital system from the consumer market. Although making correct system-level design decisions in the early design stages is a non-trivial task and has far-reaching impact on the final quality of the system, this process has not been fully automated. Among many considerations in designing a multi-chip system, reducing the time-to-market is the one of the most critical. In this paper, we present EDEN, a software package which is based on a mixed integer-linear programming formulation of the multi-chip system synthesis problem. Our objective is to synthesize system architectures that have minimum prototyping time and satisfy the user-specified cost and timing constraints. We belive EDEN can speed up the system-level design process by helping the designer make early system-level design decisions through its capability to search the vast design space and to visualize feasible system architectures. Experimental results of EDEN on the synthesis of an MPEG description are presented.

1. Introduction

The decreasing feature size and consequently increasing density of integrated circuits has made it possible to implement entire systems on a single chip or a small set of chips. When confronted with the problem of designing such a system, a system design engineer faces the challenges of confining the cost of the system, meeting the speed requirements of the system, and reducing the time-to-market. The design space that the engineer must consider is indeed very large, given the flexibility of (a) partitioning the problem for implementation on k chips, where k is an unknown, (b) deciding the implementation style of each task in the system, (c) deciding the architecture for each individual chip, and (d) deciding the overall system architecture including the system busses and control logic. In this paper, we report on a CAE tool called EDEN which provides an Early Design ENvision environment to a system design engineer. EDEN identifies design choices that are highly likely to satisfy the specified area and timing constraints and have short time-to-market.

EDEN supports three implementation styles for chip design, namely, gate array, standard-cell, and FPGA. At present, EDEN assumes that the entire system is implemented in hardware, with hardware-software tradeoffs a future consideration. EDEN looks at mixed implementation styles for the system i.e. in the final implementation of the system, one or more chips may be realized in each of the three ASIC implementation styles. However, the user has the option of restricting the number of styles so as to generate design points that are, say, purely gate-array or purely FPGA. This feature is useful during the prototyping stages of the design, where the engineer can generate FPGA implementations for design verification purposes.

Given the specification of the system in the form of a task flow graph, EDEN performs the following functions:

1. selection of implementation architectures and styles for each task in the system specification,
2. task-level partitioning of the system specification,
3. selection of die type for each partition,
4. selection of package type for each die, and
5. selection of bus architecture.

EDEN makes use of a behavioral-level estimator to obtain predictions of area and delay for each task. A mixed integer-linear programming model is used to concurrently solve the five subproblems mentioned above. EDEN then generates a set of design points that are feasible (in terms of

*This research is funded by the Advanced Research Projects Agency and monitored by the Federal Bureau of Investigation under the Grant J-FBI-94-161. The information reported here does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred.

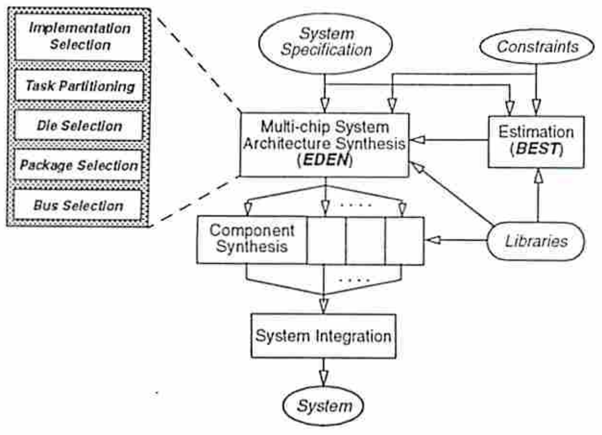

Figure 1. System Design Methodology assumed in EDEN

cost and timing constraints) and which have short time-to-market. The user can graphically visualize the design points in a multidimensional output design space in order to compare the solutions.

Figure 1 shows the methodology for system-level design using EDEN. The input to EDEN is a specification of a digital system in the form of a multi-process VHDL description. Internally, the system description is represented using a task flow graph. The user specifies a cost constraint as an upper bound on the cost of a single unit of the product. A timing constraint is an upper bound on the processing delay between an input and an output node in the task flow graph. The behavioral estimation tool BEST [9] is used to derive area and delay estimates for alternate implementations of each task. The output of EDEN is a set of descriptions of the multi-chip system implementations. Using the information from EDEN, a high-level synthesis system can synthesize the chips.

2. Previous Work

Existing literature on system-level synthesis at the level we are considering has focussed mainly on hardware-software codesign. Gupta and De Micheli [6] assumed a single processor/ single bus architecture and proposed a hardware-software partitioning algorithm. Kalavade and Lee [8] reported a constructive heuristic which traverses the directed acyclic graph representation of a task flow graph and maps nodes into hardware or software based on an appropriate objective function. Chen [2] used mathematical programming and genetic algorithm to partition processes across different packages. Prakash and Parker [10, 4] considered the problem of synthesizing application-specific multiprocessors from task flow graph(TFG) descriptions. Clustering of functional objects to perform of hardware partitioning was proposedby Vahid and Gajski [11]. Filo et al. [5] proposed an algorithm which optimizes the interface by reducing the amount of blocking communication. PUBSS is a system to generate a relatively scheduled I/O description called Behavioral FSM from a specification in VHDL and then solves a set of linear equations to minimize the handshaking in communication [12]. A trade-off study comparing the cost and performance of a single big chip to that of several smaller chips packaged into an MCM was reported[3]. The trade-off was carried out for the SUN MicroSparc System. It shows that task-level partitioning and package selection are the main factors in determining the total system cost and such decisions should be considered in the early design stages.

3. Problem Definition and Approach

A *die* is defined as a set of functional blocks connected by a set of busses and a silicon substrate. A *chip* is composed of a set of dies connected by a set of busses and a package. A **hierarchical bus-based multichip system** is composed of a set of chips interconnected by a set of busses. For a given system behavior specification in task flow graph $G(V, E)$ where V is a set of nodes that represent the behavior of functional tasks and E is a set of edges that represent the communication between functional tasks, the **multichip architecture synthesis** problem is to find a number of multichip system architectures which perform the given system behavior while satisfying the given cost and performance constraints and short prototyping time by determining partitions of design entities at the corresponding level and selecting components from implementation, bus, die, and package libraries.

We used the Mixed Integer Linear Programming(MILP) to solve the above multichip system architecture synthesis problem. Although it is well-known that MILP is not a practical tool to solve a large problem because of its long running time, we believe that, in the beginning stage of solving new optimization problem, the mathematical formulation process required by MILP enables the deeper understanding of the problem and provides a solid model, which can be used as a framework in building an efficient heuristic.

4. MILP Formulation

In this section, a set of constraints associated with each of the EDEN subproblems described in Section 1 is given

4.1. Design Style and Implementation Selection

For every task, EDEN must select an implementation style and design point (area and delay) in the design space.

Let I_t be is the set of all implementations of task t, regardless of the design style. An implementation $i \in I_t$ is characterized by the following :

1. F_i is the functionality of the task t, (e.g. Discrete Cosine Transform, Huffman Encoding),
2. $STYLE_i$ is the design style for implementation i, (e.g. standard-cell or gate array),
3. $AREA_i$ is the area requirement of implementation i of task t. We obtain an estimate of $AREA_i$ using the behavioral-level estimator BEST [9].
4. $DELAY_i$ is the delay associated with implementation i, which is also estimated from the CDFG of task t.

Let the binary variable z_{ti} be set to 1 if and only if the implementation $i \in I_t$ is selected for task t. Since each task can be assigned at most one implementation style,

$$\sum_{i \in I_t} z_{ti} = 1$$

In the selected implementation selection for a task t, let $TAREA_t$ be the contribution of task t to the total area. Similarly, the delay of task t, and the design style for task t are denoted by $TDELAY_t$, and $TSTYLE_t$ respectively. It is clear that

1. $TAREA_t = \sum_{i \in I_t} z_{ti} \cdot AREA_i$,
2. $TDELAY_t = \sum_{i \in I_t} z_{ti} \cdot DELAY_i$, and
3. $TSTYLE_t = \sum_{i \in I_t} z_{ti} \cdot STYLE_i$.

4.2. Task Partitioning and Die Selection

EDEN must determine how tasks are clustered into partitions, and which die types are used for each partition. Let I_s denotes the set $\{i_1, i_2, \cdots, i_n\}$ of implementations chosen for the tasks in Section 4.1. A partition of I_s is a collection of mutually disjoint, non-empty sets $P_1, P_2, \cdots, P_k$, such that $\bigcup P_j = I_s$. A partition of I_s is said to be *valid* partition if, for $1 \le j \le k$, it can be asserted that

$$i_\alpha \in P_j, i_\beta \in P_j \;\Rightarrow\; STYLE_\alpha = STYLE_\beta \quad (1)$$

Constraint (1) ensures that two implementations that correspond to two different design styles are not placed in the same partition. For example, we cannot group a standard-cell realization of a task t_1 with a gate-array realization of task t_2. Constraint (1) is non-linear, but can be rendered linear by introducing intermediate variables and further constraints on the intermediate variables [7].

Let B denote the set of die types that are available. Die types are characterized by three parameters, namely, size, pin count, and design style. $DIESIZE_b$, $DIEPIN_b$, and $BSTYLE_b$ denote these parameters for die type b. After partitioning a set of task implementations, it is necessary to select a die type b to house the partition p. A constraint ensures that only one die type is selected for a partition.

The type of die assigned to a partition p, indicated by $PSTYLE_p$, is given by a second constraint.

An assignment of die types to partitions is said to be *valid* if all tasks in a partition fit within the area of the die selected, as checked by a constraint. We used the following estimation model to predict the area requirement of a partition. The "core" of the partition consists of functional area and wiring area corresponding to the task implementations that are mapped to the partition. In addition to core area, we also considered the area taken up by the I/O pads.

Another constraint makes sure that the selected die type for p can satisfy the pin-count requirement of p. When a number of tasks are implemented on the same partition, it is not straightforward to compute the number of I/O pins for the partition. An analysis of the behavioral specification of the tasks is necessary for computing the number of I/O pins for each partition. A simple upper bound for $PINCOUNT(p)$ is the sum of the bit-widths of all "external channels". The upper bound is used to compute the pin count of a partition.

To ensure that we select the die from the appropriate design style category, a constraint checks that the design style is valid.

4.3. Package Selection

The problem of package selection is to select a package type for every partition p. A package type K is characterized by its size $CAVITYSIZE_K$ and its pin count $PKGPIN_K$. Constraints that check the size and pin count of a selected package can be constructed to ensure that the selected package type is appropriate for a partition. Let $w_{K,p}$ be a binary variable which is set to 1 if and only if the package type K is selected for a partition p. Since at most one package type can be selected for a given partition, we have

$$\sum_K w_{K,p} = 1 \quad \forall p$$

Let $CAVITYSIZE_K$ and $PKGPIN_K$ respectively be the cavity area parameter and pin count associated with package type K. Then the packaged area and pin count constraints for partition p is given by

$$\sum_K w_{K,p} \cdot CAVITYSIZE_K \ge DIESIZE_p \quad \forall p$$

$$\sum_K w_{K,p} \cdot PKGPIN_K \geq PINCOUNT(p) \quad \forall p$$

4.4. Interface Configuration

The problem of interface configuration is that of selecting a bus implementation style for each edge e in the task flow graph. Every edge e in the TFG is characterized by the volume of data v_e communicated on e. For a given partition of tasks, the variable b_e denotes whether or not the edge e is an external edge. A bus implementation style g is characterized by its width ω_g, the delay $IDELAY_g$ associated with the transport of one unit of data over the bus, and a Boolean variable b_g which denotes the bus is external (internal) when it is set to 1 (0). After bus implementation styles have been selected for the edges, we can associate a delay $CDELAY_e$ and a width ω_e with edge e. Three constraints select width, delay, and whether the communication is external or internal.

4.5. Cost Constraint

Let $COST_u$ be the user-specified upper bound on the cost of a single unit. Let $DCOST$, $FCOST$, and $PCOST$ be the development, fabrication, and packaging costs associated with the system under development. $DCOST$ includes the cost of design and prototyping such as mask fabrication. $DCOST$ is assumed to be supplied by the user. $FCOST$ includes the cost of fabricating the chips and the boards. $PCOST$ includes the cost of packaging the chips and the entire unit.

The $FCOST$ is calculated as the sum of all die costs as follows.

$$FCOST = \sum_p(\sum_b x_{b,p} \cdot COST_b) \tag{2}$$

where $COST_b$ is the cost of die b and $x_{b,p}$ is a binary variable for the die selection. If the style of partition p is either FPGA or gate array, the cost is known from the library. If the style of partition p is the standard cell, the cost should be calculated as follows:

$$COST_b = \frac{f_c \cdot PARTSIZE_p}{Yield_p} \tag{3}$$

where f_c is the cost parameter associated with the fabrication facility and has the unit dollars per unit area. A non-linear model for yield [1] along with the assumption of rectangular defect density distribution is used to estimate $Yield_p$ (i.e. $Yield = \frac{1}{2 \cdot DEFECT_0 \cdot PARTSIZE_p}$, where $DEFECT_0$ is the number of defects per unit area).

In order to linearize, We used the piece-wise linear approximation for the curve that plots the fabrication cost of a die as a function of the area of the die.

Let $COST_K$ be the cost of packaging a single chip using the packaging style K. Then

$$PCOST = \sum_p(\sum_K w_{K,p} \cdot COST_K) \tag{4}$$

Combining (2) and (4) we can write the user-specified cost constraint as

$$\frac{DCOST}{N_u} + FCOST + PCOST \leq COST_u$$

where N_u is the number of units manufactured.

4.6. Timing Constraint

The user specifies timing constraints in the form of upper bounds on execution time between two selected nodes in the task flow graph. A preprocessing step is used to convert these timing constraints into execution time constraints on *paths* in the task flow graph. We use the following equation to estimate the delay T_{papth} associated with the path

$$T_{path} = \sum_{t \in path} COMP(t) + \sum_{e \in path} COMM(e)$$

where t is a task that lies on the path p and e is an edge that lies on p. $COMP(t)$ is the computational delay associated with task t and $COMM(e)$ is the communication delay associated with edge e which is defined as

$$COMM(e) = \frac{v_e}{\omega_e} \cdot \tau_e$$

where v_e is the volume of data, ω_e is the width of the chosen bus for e and τ_e is the period of the bus clock cycle.

If $TIME_u$ is the upper bound specified by the user for an node pair (src, dst), then, for every path $path$ from src to dst in the TFG, we have the following constraint:

$$\sum_{t \in path} COMP(t) + \sum_{e \in path} COMM(e) \leq TIME_u$$

4.7. Objective Function

Presently, the objective function in **EDEN** is the prototyping time of the system under development. We assumed that the prototyping time is composed of five terms, namely, design time, fabrication time, packaging time, and test application time. It is difficult to estimate the prototyping time without assuming a model for the logistics of the operation of the design house. For example, we assume that all chips are sent out for fabrication concurrently, and are then packaged concurrently. We assume that first silicon passes all the tests and that no redesign efforts are necessary. Presently, we do not have a model to estimate the design time or test

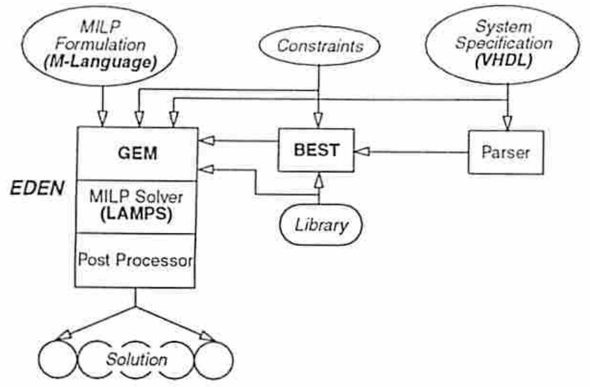

Figure 2. EDEN Software Organization

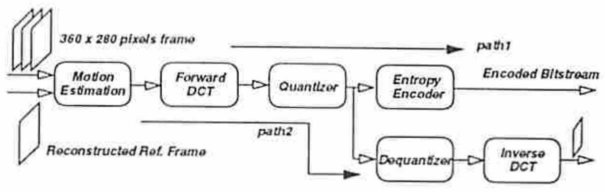

Figure 3. Task Flow Graph for MPEG Encoder

time for a chip p; we assume that the same are provided as input. Alternately, a model that considers the complexity of the chip in terms of gate count can be used.

Let $\mathcal{DT}_p$, $\mathcal{FT}_p$ and $\mathcal{PT}_p$ be the design time, fabrication time and packaging time respectively for a chip that corresponds to partition p. The time-to-prototype $\mathcal{PRT}$ can be given by

$$\mathcal{PRT} = \sum_p \mathcal{DT}_p + \sum_p \mathcal{FT}_p + \sum_p \mathcal{PT}_p \qquad (5)$$

Equation 5 was used in our implementation. The fabrication time $\mathcal{FT}_p$ and packaging time $\mathcal{PT}_p$ for a partition p can be obtained using

$$\begin{aligned} \mathcal{FT}_p &= \sum_b x_{b,p} \cdot FTIME_b \\ \mathcal{PT}_p &= \sum_K w_{K,p} \cdot PTIME_K \end{aligned}$$

where $FTIME_b$ and $PTIME_K$ are the time for fabricating a chip of die type b and packaging the chip using packaging style K. The constants $FTIME_b$ and $PTIME_K$ are available as part of technology library information.

5. Experimental Results

5.1. EDEN Software Organization

The organization of **EDEN** is shown in Figure 2. There are two principal inputs to **EDEN** – a behavioral specification of the system written in VHDL and a set of input-to-output timing constraints. A parser reads the VHDL specification and extracts a task flow graph. The library contains the area, delay and other summary information for hardware modules, die types, package types, and bus types.

We presently permit three different design styles, namely, standard-cell (VLSI Technology Inc.'s vsc350 family, 1.2 micron), gate-array (VLSI Technology Inc.'s vgt350 family, 1.2 micron), and field-programmable gate arrays (Xilinx XC4000 family). The module library includes using vendor-supplied modules and several custom-designed modules. Examples of custom-designed modules are 8-bit ripple carry adder and parallel 8-bit multiplier. Such modules are designed, laid out, back-annotated, and simulated with vendor tools such as the XACT tool suite and Compass tool set (ChipCompiler, GateCompiler, and Qsim).

The behavioral-level estimator BEST [9] is used to obtain area and delay estimates for implementations of each task with modules designed in different design styles in the library.

The linearized MILP formulation is written in the *M-Language*, which we developed to express the constraints in mathematical form. A compiler called GEM generates the standard matrix representation of the design problem from the above information. An MILP solver called LAMPS was used to solve the matrix representation of the design problem.

5.2. Multichip Architecture Synthesis Results of an MPEG encoder

The task flow graph for the example, the MPEG encoder, is shown in Figure 3.

The frame size for MPEG I standard is 360×288, and the frame rate is 30 frames/second. The pixel block sizes are 16×16 and 8×8 for motion estimation and DCT respectively. The real-time constraint for processing a 16×16 pixel block is 164,609 ns. when the frame rate is 15 frames per second and only Y components are considered.

Initially, we attempted a virtually unconstrained optimization using very high values for the upper bounds on cost as well as the delays for Path 1 and Path 2 shown in Figure 3. Thus the upper bound on cost was \$10,000 and the upper bound on each path delay was 500,000 ns. The resulting design is shown in Figure 4. The design has five partitions, two of which have a suggested implementation of GA, and the remaining are implemented as FPGA. The value of the objective function for the resulting design is 55 units and the cost of the design is \$4,105. The achieved delays on Paths 1

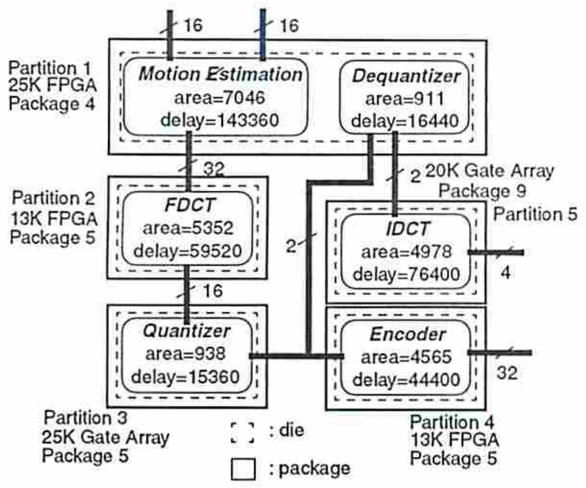

Figure 4. Solution generated by EDEN for the MPEG encoder with virtually no cost and timing constraints

and 2 are 499,480 ns. and 490,720 ns. respectively. The assignment of tasks to partitions along with the area and delay estimates for the implementations of tasks is shown in the figure. We also show the bus widths computed by the program. **EDEN** selected a bus clock of 100 ns. for all busses. Table 1 shows the salient information about the five partitions of the design. The reader will notice that the area utilizations of the gate-array packages selected for partitions 3 and 4 are poor. The reason for this is that our technology library was rather small and did not contain many package types.

P[a]	Partition		Die		Package	
	Size	Pin	Size	Pin	Size	Pin
1	33,761	68	62,400	256	70,000	208
2	24,115	48	34,400	192	70,000	299
3	1,354	18	20,893	388	70,000	299
4	6,625	34	16,906	350	70,000	208
5	22,425	6	34,400	192	70,000	299

[a]Partition number

Table 1. Partition Information for the MPEG Design 1.

Using the design shown in Figure 4 as the guideline, we ran **EDEN** again, tightening the timing constraint and cost constraint over successive executions. The salient features of some alternate designs obtained using **EDEN** are summarized in Table 2. In the table, the objective function is the sum of the products of fabrication time and chip size for all the chips. The path delays are shown in ns. and the cost is shown in dollars. After observing that Design 1 achieves path delays of 499,480 ns. and 490,720 ns., Design 2 was obtained by tightening the performance constraint; we reduced the upper bound on the path delays to 190,000 ns. This resulted in a design which used one gate-array, one FPGA and two standard cell chips. The cost improved from \$4,105 to \$1,891; Design 1 is more expensive due to the use of FPGA. The performance bound was further tightened to 164,600 ns. in order to generate Design 3. This design also uses all the three design styles; **EDEN** selected the standard-cell style for the FDCT, motion estimation, and the quantizer modules in order to meet the timing constraint.

#	Obj. Value	Cost	Delay (μs)		# of Partitions			
			Path1	Path2	F[1]	G[2]	S[3]	T[4]
1	55	4105	499	490	3	2	0	5
2	2333	1891	188	187	1	1	2	4
3	2500	1247	159	163	1	1	3	5
4	82	1010	477	480	1	1	0	2
5	1415	446	492	495	0	1	1	2
6	2000	481	162	162	0	1	1	2

1:FPGA, 2:Gate array, 3:Standard cell, 4:Total

Table 2. Designs for the MPEG Encoder

EDEN permits the designer to consider alternate possibilities of task partitions, implementation styles and design styles as described above. The designer can then tabulate the salient features of the designs and make a comparative evaluation of the designs. For example, although standard cell realizations have lower fabrication costs, they are not desirable if only a limited number of systems must be produced for a specialized application. A single run of **EDEN** for the MPEG example takes about 10 hours on a single CPU Sun SPARCStation 20 with 64 MB main memory.

Other than comparing different local-optimal design points generated using **EDEN** , a system designer can employ the tool in another way. During the initial phase of system development, when changes are still being made to the algorithm and the application-related parameters are being tuned, simulation is an expensive method of system verification. Emulation using FPGA is being actively considered by researchers as an alternative for system verification. A system designer can generate an all-FPGA design using **EDEN** by specifying large bounds on cost and delays. The designer can also generate alternate designs and plan a marketing strategy; e.g. an expensive, lower-performance design which has a low time-to-market can be launched initially to capture the market and concurrently an effort can be initiated to produce a second version of the product which is cheaper and has a superior performance.

5.3. Complexity of the MILP formulation

In the example of the MPEG encoder described in Section 5.2, there are six tasks and a maximum of 6 partitions are possible. We considered a technology library consisting of 11 die types and 16 packaging styles. There are a total of 81 implementation points of the 6 tasks. After linearization of the constraints, the number of variables is 2,324 and the number of constraints is 4,345.

6. Conclusions and Future Work

We have described **EDEN**, a system-level synthesis tool that permits the designer to navigate through the large design space. The points in the design space are characterized by (a) the number of partitions and the way in which tasks are partitioned across them, (b) implementation styles selected for each task, (c) design styles selected for the partitions, (d) schedules selected for the tasks, and (e) the configuration of busses used for inter-chip communication. Designs are compared on the basis of time-to-market. **EDEN** attempts to minimize the time-to-market while satisfying both cost and timing constraints. We believe ours is the first attempt to formulate the multi-chip design problem in this generality and breadth. **EDEN** can be effectively used by system developers to plan system development and marketing strategies. We are enhancing **EDEN** in several ways. We are reworking the MILP formulation to support multi-chip module realizations. We are also working on improving the run-time performance of our software through the use of heuristics.

References

[1] W. Bertram. Yield and reliability. In S. M. Sze, editor, *VLSI Technology*, page 599. McGraw-Hill, 1983.

[2] C. Chen. *System-level Design Techniques and Tools for Synthesis of Application-Specific Digital Systems*. PhD thesis, University of Southern California, 1994.

[3] P. Dehkordi, K. Ramamurthi, D. Bouldin, H. Davidson, and P. Sandborn. Impact of packaging technology on system partitioning: A case study. In *Proc. of IEEE Multi-chip Module Conference*, 1995.

[4] DeSouza-Batista and A. Parker. Optimal synthesis of application specific heterogeneous pipelined multiprocessors. In *The international Conf. on Application-Specific Array Processors*, 1994.

[5] D. Filo, D. Ku, C. Coelho, and G. D. Micheli. Interface optimization for concurrent systems under timing constraints. *IEEE Trans. on Very Large Scale Integration Systems*, 1(3):268, September 1993.

[6] R. Gupta and G. D. Micheli. System-level synthesis using reprogrammable components. In *Proc. of the European Conference on Design Automation*, 1992.

[7] D. Heo, C. Ravikumar, and A. Parker. An MILP-based formulation of system-level synthesis. Technical Report CENG-96-21, Computer Engineering Division, Department of EE-Systems, University of Southern California, 1996.

[8] A. Kalavade and E. Lee. A global criticality/local phase driven algorithm for the constrained hardware/software partitioning problem. In *Proc. of Codes/CASHE '94, Third Intl. Workshop on Hardware/Software Codesign*, 1994.

[9] K. Kucukcakar and A. Parker. A methodology and design tools to support system-level VLSI design. *IEEE Trans. on Very Large Scale Integration Systems*, 3(3), September 1995.

[10] S. Prakash and A. Parker. Synthesis of Application-specific Multiprocessor systems including memory components. *Journal of VLSI Signal Processing*, 8(2):97–116, October 1994.

[11] F. Vahid and D. Gajski. Clustering for improved system-level functional partitioning. In *International Symposium on System-level Synthesis*, 1995.

[12] W. Wolf and R. Manno. High-level modeling and synthesis of communicating processes using VHDL. *IEICE Trans. on Information and Systems*, E76D(9):1039–1046, September 1993.