Versatile Multichip Digital System Architecture Synthesis Tools

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

CENG 96-26

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

October 18, 1996

Versatile Multichip Digital System Architecture Synthesis Tools

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

October 16, 1996

Abstract

We present a comprehensive model and tools for multichip system design. This model can be used to explore the system architecture design space rigorously and quickly to find a good system architecture. One tool, EDEN, is implemented by the direct linearization of the model into Mixed Integer Linear Programming. The other tool, GARDEN, is based on a genetic algorithm. These tools can be used for a variety of multichip design applications. An MPEG multimedia example was designed as an example.

1 Introduction

System architecture design is an important step in the design of a digital system which sets the cornerstones for the system. The system architecture designer makes decisions on a number of system design issues such that the overall system characteristics satisfy constraints on system metrics such as the cost, performance, and development time. Among such early system design issues for the hardware part of a system are *datapath architecture selection* for each functional task, *memory architecture selection*, *interface architecture selection*, *partitioning of functional tasks*, *package selection for each functional partition*, and *scheduling executions among functional tasks*. Because functional tasks can be implemented in a number of different datapath architectures[1], it is necessary to choose an architecture. For memory architecture selection, the locations and sizes of memory blocks are to be chosen[2, 3]. Interconnection architecture selection determines the sharing of buses among the communication channels between functional tasks[4] and the types of buses used. The assignment of a functional task to a partition is determined during functional task partitioning[5, 6].

*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.

Scheduling is the ordering of execution of functional tasks. The package type for a partition of functional tasks is selected during *package selection*. Another partitioning step in which dies are partitioned into groups is required if Multi-Chip Modules(MCMs) are considered [7].

Hardware design is important, not only because the hardware part of a hardware/software systems must be designed but also because a full-hardware implementation has a strong performance advantage. Designers cannot solve the system design problem by merely synthesizing the individual components because considerations on the overall system characteristics and budgeting available resources such as cost, performance, and power among components must be handled prior to the datapath design. Individual component designs can start only after the system architecture is determined.

Assessing the influence of a system design decision is a difficult and complex task because one decision is closely tied to another and detailed information about the implementation is not available. For example, let us consider the design subproblem of selecting an implementation style for a system. A modern digital system is composed of many different implementation styles such as FPGA, gate array, and standard cell [8], which play a key role in determining system characteristics. Commercial off-the-shelf (COTS) components can also be another possible choice. In selecting a style for each functional task, maximum clock cycle, the cost of a chip, and the development time should be considered. In evaluating the cost and performance of a system, we also need to know about other system design decisions, e.g. how many chips of what size are necessary for the system, and which functional task is assigned to which partition.

The complexity of multichip system architecture design is threefold. First, the number of possible choices is huge for each system design issue. Second, design problems are interconnected to each other in complex ways and make independent optimization impossible. Finally, the way that one decision influences the system metrics is not easily assessed. Because grasping all the interrelations among subproblems together with their influence on the system metrics at the same time is extremely difficult, currently the system design process relies on human design experience; no comprehensive model that can handle the entire system architecture synthesis problem has been developed. Such an approach by human designers is based on a construct-improve type heuristic and considers subproblems sequentially[9], which increases the possibility of requiring another iteration in the design process and the delay of the development schedule which could be fatal in the market. Current design methodologies and their problems are well described in the paper by Shaw *et al.*[9].

To avoid problems which arise from such a sequential approach, we should solve all aspects of the multichip system architecture problem concurrently. Rigorous modeling of multichip system architecture synthesis that expresses the interdependencies between design subtasks is required to solve the problem concurrently. In this paper, we first present our model for multichip system architecture synthesis and then present two tools that solve the model to generate a number of different multichip system architectures. We make some simplifying assumptions in order to demonstrate a prototyping design environment, but we believe our tools remain more comprehensive than most other attempts to solve these problems.

2 Related Work

Gupta and De Micheli [10] pioneered automated hardware/software codesign. Kalavade and Lee [11] reported a constructive heuristic which traverses a task-flow graph and maps nodes into hardware or software based on an appropriate objective function. Adams and Thomas [12] focus on behavioral-level transformations for hardware/software codesign, such as splitting a task or moving code from one task to another. Srivastava and Brodersen[13] proposed a system design framework, SIERA, in which an application-specific system is developed by partitioning the network of processes model of a system into a layered architectural template. Parker and Prakash [14] considered the problem of synthesizing application-specific multiprocessors. For system architecture synthesis, Gajski *et al.*[15] proposed a system design methodology called Specify-Explore-Refine aiming at considerable reduction of the system development time. Clustering of functional objects to perform hardware partitioning was proposed [6]. Filo et al. [4] 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 from a specification in VHDL to minimize the handshaking in communication [16]. Chen [5] used mathematical programming and a genetic algorithm to partition processes across different packages. For package selection and structural partitioning, 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 [7], which shows that task-level partitioning and package selection are the main factors in determining the total system cost. Shih et al. [17] proposed an algorithm for structural partitioning of a system into chips on an MCM.

3 The Multichip System Architecture Synthesis Problem

We model the specification of a digital system as a set of communicating multiple processes and represent it with a *task-flow directed hyper-graph*, $G(E, V)$, which consists of task nodes V and communication edges E. A task node represents a functional task in which the behavior of the task can be described algorithmically with a hardware description language. Communication among functional tasks is represented with an directed hyperedge.

Our target architecture is a *hierarchical bus-based multichip system.* The multichip architecture synthesis problem can be defined as the following set of design subtasks that derive the target architecture from the given task-flow graph representation of system behavior:

1. *style selection (SS)* in which an implementation style for each task in the specification is selected from the given design styles such as FPGA, gate array, and standard cell,
2. *architecture selection (AS)* in which a datapath architecture for each task is selected from possible datapath architectures,
3. *task partitioning (TP)* in which tasks are partitioned into groups,

4. *die selection (DS)* in which a die type such as 10K-gate die is selected for each task partition from the given die library if the task partition is implemented with either FPGA or gate array,

5. *die partitioning (DP)* in which dies are partitioned into groups for multi-chip modules,

6. *package selection (PS)* in which a package type that houses each die partition is selected from the given package library,

7. *channel partitioning (CP)* in which channels are partitioned into groups such that channels in a group share a single physical bus,

8. *bus selection (BS)* in which the bus type for each channel partition is selected,

9. *task scheduling (TS)* in which the ordering and overlapping of execution of tasks are determined, and

10. *memory architecture selection (MS)* in which the locations and sizes of memory blocks are determined.

The complete model for multichip system architecture synthesis should include all the above design subtasks but the complexity of the model including all subproblems is too high. Therefore, for this prototype system, we built a simplified model which is practical and also complete enough to verify its usefulness. Among design subtasks included in our current version of the multichip system architecture synthesis model are **style selection, architecture selection , task partitioning, die selection, package selection, and bus selection.** *style selection* and *architecture selection* are merged into **style and architecture selection (SAS)** which can be called *implementation selection* because style and architecture are two aspects of an implementation.

Design decisions on each design subtask in our simplified multichip system architecture synthesis can be represented as a set of binary variables. In Figure 1, a binary variable representation of a design decision for task partitioning is illustrated. The binary variable $y_{pt} = 1$ if task t is assigned to partition p, and y_{pt}=0 otherwise.

In our model, a library represents a set of existing choices for a design subtask. An element in a library is represented with a set of parameters. For example, an implementation i in an implementation library can be denoted as $ISIZE, IDELAY$ (the size and delay of i).

We can define the binary decision variables and their sets which represents decisions on the corresponding subtasks as follows: z_{ti} denotes the selection of implementation i for task t, x_{bp} denotes the selection of die type b for task partition p, w_{kp} denotes the selection of package type k for task partition p, and d_{eg} denotes the selection of bus type g for channel e.

A multichip system architecture A can be represented as a set of all above decision binary variables, set to 0 or 1. Not all possible system architectures are valid. There are a number of conditions that should be met in order for a solution to be valid, for example, a task cannot be assigned to two partitions and a package selected for a die should be big enough to hold the die. We call such conditions which decision binary variables are subject to *validity constraints* , in order for a system architecture to be valid.

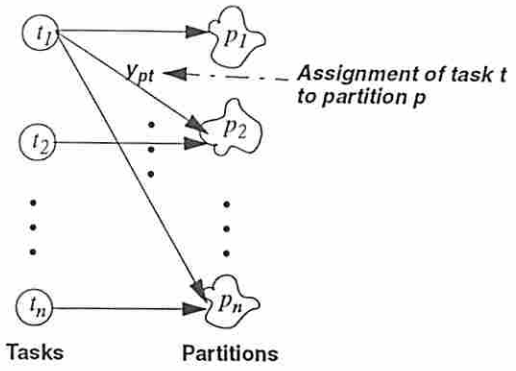


Figure 1: A model for the decision in a design subtask (task partitioning)

There are two types of validity constraints. The first type of validity constraint is rather trivial and comes from the characteristics of a subtask. For example, one such constraint for SAS is that one and only one implementation i can be selected for task t and can be written as follows:

$$\sum_{i \in I} z_{ti} = 1 \tag{1}$$

There are 5 such validity constraints of this type in our model.

The second type of validity constraint is called a *parametric validity constraint* because constraints are imposed on characteristic parameters of design entities, for example, the size of a task partition. If the design entity involving in a parametric validity constraint is an element from a library, then such parameters can be read from the library. However parameters for a design entity created during the design process such as a task partition must be computed. We call such metric functions *partitioning metric functions*. For example, in order to find the metrics for task partition p, we define the following partition metric functions:

1)The size $TPartSize(p, Z, Y, X, D)$ of partition p is a function of not only implementation selection Z and task partitioning Y but also that of die selection X because parameters related to a chosen die type such as the routability W_b , the average gate size G_b, and the average I/O pad size P_b change the physical size of a task partition although the datapath architecture remains the same. 2)The number of pins for partition p $TPartPin(p, Y, D)$ is a function of task partitioning and bus selection because task partitioning determines whether a channel is cut by a task partition boundary. The number of pins required for each channel is determined by the selected bus type for the channel.

With the above partition metric functions, parametric validity constraints for the size and pin requirements between die selection and task partitioning are defined as follows:

$$\sum_{b \in B} x_{bp} \times DSIZE_b \;\geq\; TPartSize(p, Z, Y, X, D) \tag{2}$$

$$\sum_{b \in B} x_{bp} \times DPIN_b \geq TPartPin(p, Y, D) \quad (3)$$

Those constraints ensure that the physical dies selected meet partition size and pin requirement. There are 7 such parametric validity constraints in our model.

Relationship between design subtasks and validity constraints can be depicted as a hypergraph as shown in Figure 2. In this graph representation of the multichip system architecture synthesis model, each node corresponds to a design subtask and the edge corresponds to a validity constraint. Decisions on nodes connected by an edge involve a validity constraint represented by the edge. The validity constraints expressed in Equation 1, 2, and 3 are marked in Figure 2. The rest of the edges in Figure 2 are other constraints.

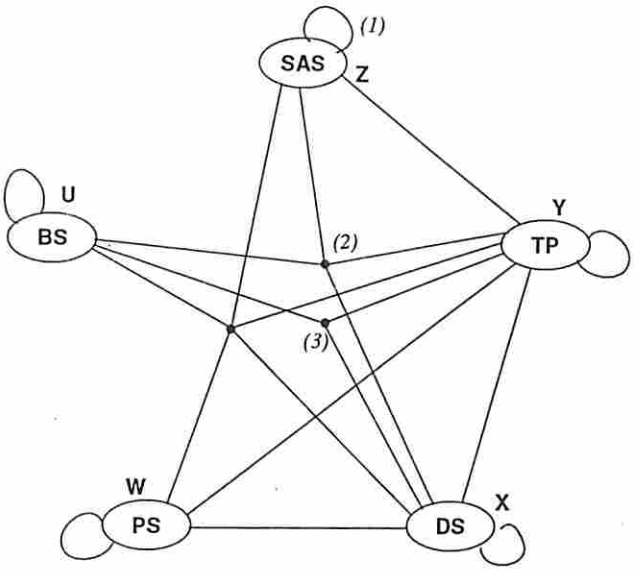


Figure 2: Graph representation of the multichip architecture synthesis model

By solving the above formulation, we produce a set of design decisions A which represent a valid multichip system architecture. Since the goal of the system architecture is not merely finding a valid system architecture but one that satisfies user-specified constraints on system metrics, we need to derive a system metric function as a function of the system architecture A. Among system metrics, we consider the cost $SystemCost(A)$, performance $SystemPerf(A)$, and time-to-market $SystemTTM(A)$ of a system in our model.

The cost of a multichip digital system is the sum of amortized development cost, costs for all components, and the assembly cost. Each component's cost is further composed of the manufacturing cost and the testing cost. For this prototype system, we approximate the system cost function as follows:

$$SystemCost(A) = NRE/n + \sum_{chip \in system} COST_{chip}$$

where NRE is the development cost, and n is the number of copies of the system that will be manufactured. NRE can be estimated by the parametric cost estimation(PCE) technique borrowed from software engineering [18] and input by the user. $COST_{chip}$ is approximated by the following equation:

$$\begin{aligned} COST_{chip} &= DieCost + PkgCost \\ DieCost &= \begin{cases} \frac{f_c \cdot SIZE_{chip}}{Yield_{chip}} & \text{if } STYLE_{chip} = \text{standard cell} \\ \sum_{b \in B} x_{b,chip} \times DCOST_b & \text{if } STYLE_{chip} = \text{FPGA or gate array} \end{cases} \end{aligned}$$

where f_c is a cost parameter associated with the fabrication facility and has the unit dollars per unit area. The proposed non-linear model for yield[19] along with the assumption of rectangular defect density distribution is used to estimate $Yield_{chip}$ i.e. $Yield_{chip} = \frac{1}{2D_0 SIZE_{chip}}$, where D_0 is the number of defects per unit area. $SIZE_{chip}$ can be calculated from $TPartSize(p, Z, Y, X, D)$.

Our model for the system performance metric function is composed of execution time on *paths* between two selected nodes (src, dst) in the task flow graph. Then, from the maximum delay for every path *path* from *src* to *dst* in the TFG $DELAY_{path}$ (see Figure 9), we have the following system performance metric function:

$$SystemPerf(A) = DELAY_{path} = \sum_{t \in path} COMP(t) + \sum_{e \in path} COMM(e)$$

where $COMP(t)$ is the delay of task t determined by implementation selection and $COMM(e)$ is the delay of channel e determined by bus selection and task partitioning.

Finally, TTM for a multichip digital system is modeled as follows: We divide TTM into two major components, namely prototyping time $TProto$ for a system and production time $TProd(n)$ for n copies of a system.

$$TTM = TProto + TProd(n)$$

We consider only $TProto$ in our prototype model. $TProto$ is further divided into two components, namely components prototyping time $CProto$ and system prototyping time $SProto$. For multichip systems, we assume system prototyping can be done in parallel with component prototyping and takes less time than component prototyping since the complexity of prototyping a board is much less than developing ASICs. Therefore TTM can be simplified as follows:

$$TTM \approx max(CProto, SProto) \approx CProto$$

$CProto$ is composed of the design time, the manufacturing time and the testing time of chips. In general, these terms depend on the availability of resources. For example, design time is a function of the number of engineers and tools available while manufacturing time and testing time vary with the availability of equipment. Design time is especially hard to characterize because tasks in the design process are not only dependent on the number of engineers and their skillfulness but also on other factors like design methodologies and

characteristics of the system being designed. Since no known available model can estimate design time realistically, we assume that the design time is rather independent of design decisions on our subproblems considered in this paper. Testing time is not considered in this simplified model. Under the above assumptions, $SystemTTM(A)$ can be expressed as follows:

$$SystemTTM(A) = \sum_{chip} TFab_{chip} + \sum_{chip} TPkg_{chip}$$

where $TFab_{chip}$ depends on the design style, which is decided in SAS, and $TPkg_{chip}$ is known from the library information. The reader is cautioned that these simplified models are used in the prototype system to demonstrate other research contributions. An industrial set of system architecture design tools would of necessity have a more complex model of costs and time-to-market customized for that industry and perhaps that particular organization.

4 EDEN - Mixed Integer Linear Programming

By linearizing the above multichip system architecture synthesis model, we can solve the problem with Mixed Integer Linear Programming(MILP). In Figure 3, we show the overview of our software package called Early Design ENvision environment(EDEN). The linearized model, *Multichip Synthesis Formulation*, is written in a language called *M* which can describe the linearized constraints in mathematical form. The algorithmic description of the task behavior is translated into the internal control-data-flow graph representation. The estimates for the size and delay of different datapath architectures are generated by the behavioral estimator BEST[20] from a CDFG representation of the task behavior and different design style libraries. A compiler called *GEM* translates the generic linearized formulation into the standard linear programming matrix representation called MPS for the specific system design problem. The MPS formulation is solved by a commercial MILP solver called *LAMPS*[21].

5 Genetic Algorithm Approach

MILP is a powerful tool to express the complex model rigorously and conveniently but even a simple model requires lengthy time to solve. The restriction of linearity on the constraints is another drawback of MILP because many factors in the model are easier to express in non-linear form. If it is possible to linearize the non-linear constraints, the linearization step increases the number of constraints exponentially.

In order to overcome the aforementioned problems of MILP, we developed another multichip system architecture synthesis tool called GARDEN which is based on the genetic algorithm (GA) approach[22]. In principle, GA emulates the evolution process in nature. Intuitively, we can build a better system by taking the good traits of two systems that perform the same function. As we will show in our experimental results, a genetic algorithm finds comparably good solutions to MILP in much shorter time while it does not require tailoring of a specialized heuristic. Incorporating the non-linearity of the model is not a problem in the GA. Finally, GA has a strong advantage in parallelizing the code because

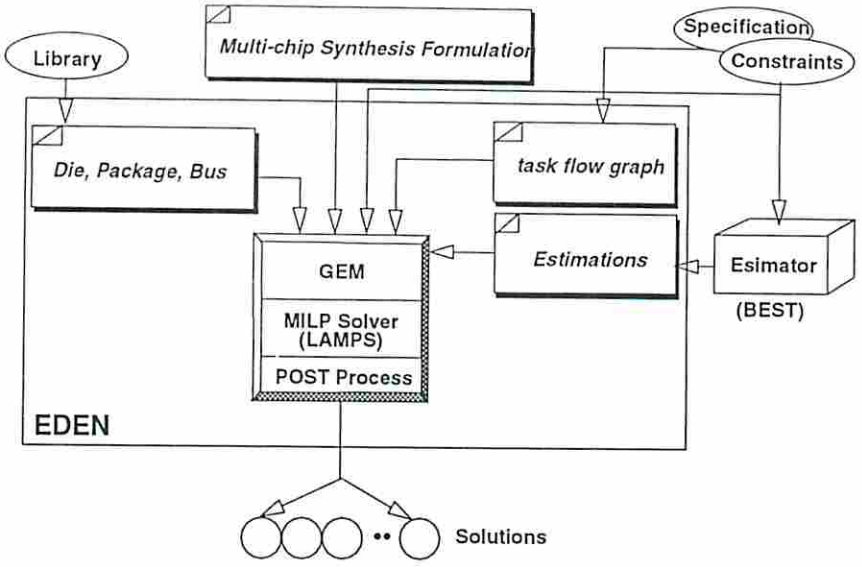


Figure 3: An overview of the EDEN implementation

the processes in GA are highly independent therefore easily parallelized to either test more complex models or handle a bigger design population[23].

Our model for the multichip system architecture synthesis in Section 2 serves as a framework for the development of GARDEN. Since the role of binary variables in shaping the system characteristics closely matches the role of genes in the reproduction process of living organisms, it is plausible to build the chromosome representation of the multichip system architecture from binary decision variables. Figure 4 illustrates how such a chromosome representation for the system architecture can be constructed from binary decision variables. There are two types of binary decision variables, one is related to tasks in the task-flow graph and the other is related to channels. Therefore, we use two chromosomes, namely a task chromosome and a channel chromosome, to represent a multichip system architecture. An initial population could be constructed by either random generation or heuristics. In random generation, a randomly generated number within the size of the library or partitions could be used as a blind decision for each design subtask in our model. For a randomly generated system architecture, the partition metric functions could be applied, then the validity constraints in our model could be checked to screen out invalid solutions. Those solutions that passed the validity constraints would constitute the initial population. However applying metric functions and validity constraints is computationally very expensive while the chances of generating a valid solution randomly are extremely low. Therefore, we use a heuristic in creating the initial population, which generates solutions that are guaranteed to be valid.

Crossover and mutation are primary mechanisms that generate new solutions. In the generic GA, crossover is done by cutting a chromosome at a fixed point and swapping genes between two chromosomes at that point. Although this technique is simple to implement, it

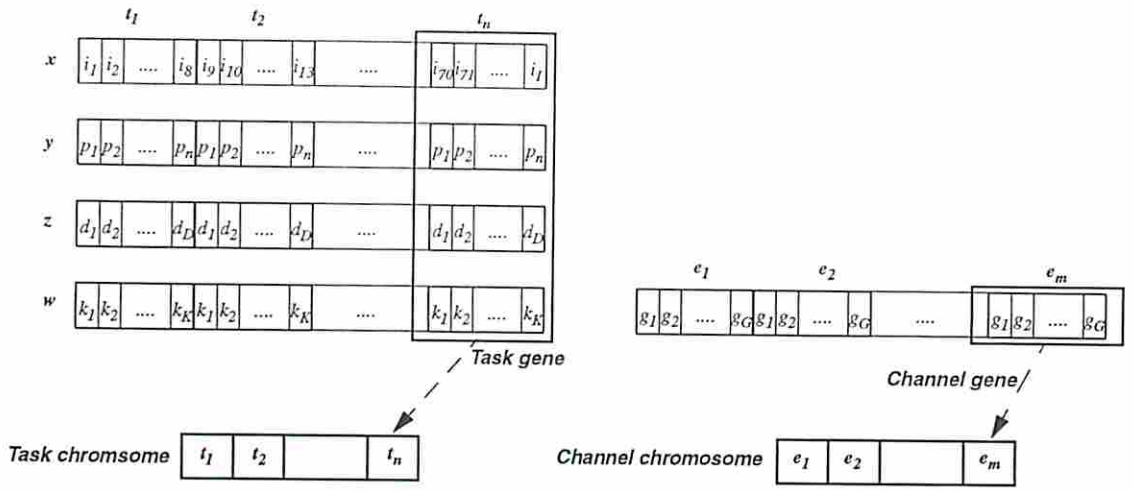


Figure 4: Constructing chromosomes from the binary decision variables for the system architecture

is not appropriate for multichip system architecture synthesis. Cutting the task chromosome corresponds to splitting the system into two parts. Depending on the solution, such splitting and merging can lead to invalid offspring solutions. The possibility of producing such invalid offspring solutions is very high. Therefore, validity checking is required, which slows down the crossover procedure intolerably, as in the random initial population creation. In order to avoid producing invalid solutions as a result of crossover, we need a crossover operation that does not disturb the validity of a solution. Such a crossover operation has been developed by observing the fact that new valid designs can be produced by exchanging part of a design at the boundary of packaging. Figure 5 illustrates the concept of such a crossover operator. The algorithm for the crossover operation that guarantees to generate valid offsprings is shown in Figure 6. In this algorithm, the chosen system architecture A is divided into two parts randomly at the package boundary. Then the rest of the population is scanned to find another system architecture B that satisfies the following compatibility conditions:

Compatibility condition Let A and B be chromosomes which are a set of task genes, $A = \{t_1, t_2, ..., t_n\}$ and $B = \{t_1, t_2, ..., t_n\}$. Let A_0 and A_1 be two disjoint and exhaustive subsets of A, i.e. $A = A_0 \cap A_1$ such that $pkg(t_i) \neq pkg(t_j)$, $\forall i \in A_i$ and $\forall j \in A_j$ where $pkg(t_i)$ is a function that returns the package for task t_i. Let B_0 and B_1 be two disjoint and exhaustive subsets of B. B is **compatible** with A if $A_0 = B_0$, $A_1 = B_1$ or $A_0 = B_1$, $A_1 = B_0$. We call the process of finding a compatible solution *mating* .

The crossover operation enables GA to search the design space for all possible combinations of existing designs. Therefore, the search range is limited by the initial solution creation. In addition, the characteristics of the population converge rapidly as the search progresses over generations because certain value of genes are depleted[23].

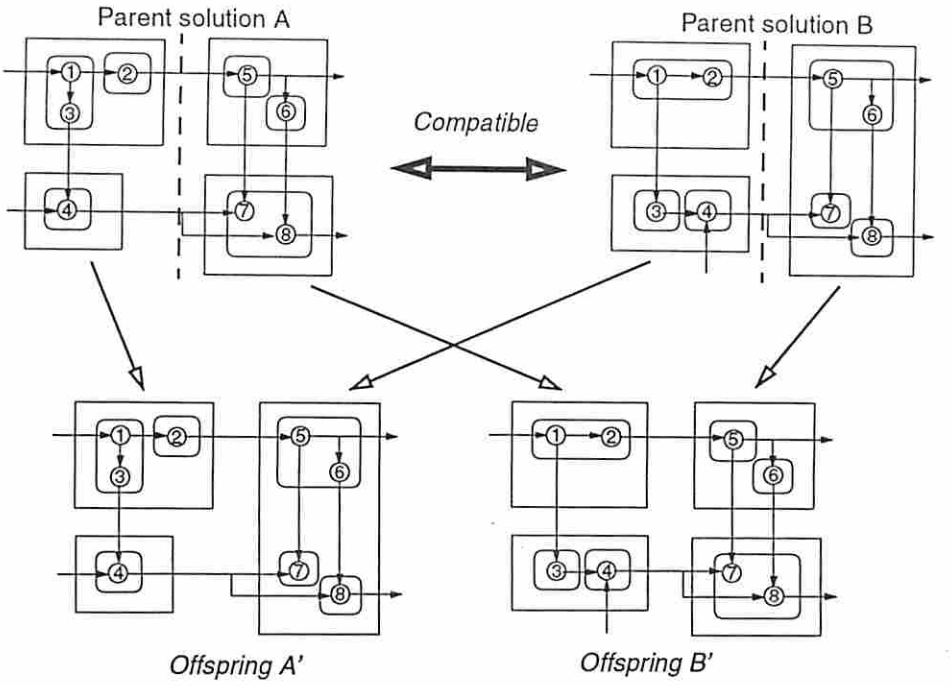


Figure 5: Illustration of crossover operations based on compatibility

```
cross_over() {
   randomly pick S[i] from solution array S;
   randomly split S[i] into two set of genes,
     S_0[i] and S_1[i] at the package boundary;
   j = mate(S[i]);
   split S[j] into S_0[j] and S_1[j];
   merge S_0[i] and S_1[j] into S'[i];
   merge S_0[j] and S_1[i] into S'[j];
}
```

Figure 6: Algorithm for the crossover operation

Not only to reintroduce lost gene values but also to explore the rest of design space which cannot be reached by crossover, a mutation operation is necessary. In the traditional mutation operation, the value of a gene is changed with a given probability. In multichip system architecture synthesis, each gene has an unique meaning in determining the system characteristics in a complicated way, for which a simple change of gene value is not appropriate. For example, if a die type of a task gene is changed, all other die types of task genes that belong to the same task partition with the mutated task gene must change together. To simplify the implementation of the mutation operation, the mutation is applied to each design subtask rather than an individual gene i.e. the mutation occurs for each design subtask with a given probability.

Unfortunately, the mutation operation cannot be completely controlled to generate a valid solution after the crossover operation because of the complex interdependencies of design subtasks. For example, when mutating task partitioning moves a task from one to another partition, such movement should not violate all validity constraints connected to the node TP in Figure 2 to preserve the validity of a solution. It is possible to avoid violating certain validity constraints easily by restricting the movement. For example, we can prevent a task of one design style from migrating to other partitions of different design styles. But, satisfying all constraints simultaneously would restrict possible migration too much. To retain the freedom of mutation, we introduce new operation called *repair*; i.e. a solution that becomes invalid after a mutation is repaired to be valid by correcting another part of the chromosome. For example, as a result of task migration among partitions, if the selected die type for the destination partition cannot hold the new task partition, the die type is replaced with another die type that can accommodate the new partition.

Each constructed solution is evaluated with a metric function called a *fitness function.* Those solutions which have higher fitness will have higher probability of generating offspring. The behavior of GA is very sensitive to the form of the fitness function. Because of the peculiar characteristics of the design automation problem, once again, the direct application of the traditional fitness function is not suitable. In multichip system architecture synthesis, we need to optimize the design in terms of the prototyping time while satisfying the cost and performance constraints. The fitness function should be designed such that it can eliminate solutions violating the cost and performance constraints while minimizing the prototyping time. It should distinguish solutions that satisfy the user-specified constraints from others while retaining solutions that fail to meet any constraint but are close so that the selection mechanism reproduces more offsprings from the better solutions although they all violate the user constraints. To have this characteristic, the fitness function should be insensitive to the objective metric and equally sensitive to all the constraint metrics until a solution satisfies the user constraints, then it should behave otherwise once a solution satisfies the user constraints. Devising a fitness function that has the above characteristics as a linear or non-linear combination of three separate fitness functions, $f(SPERF)$, $g(SCOST)$, and $h(TTM)$ for each metric is very difficult.

Therefore, in GARDEN, we use three fitness functions separately and optimize the population in a successive manner as shown in Figure 7. In the beginning, the solution is optimized for either performance or cost of the system, which are constraint metrics, then optimized for the prototyping time after both average fitness regarding performance and

cost of the system reach the critical value. The critical value for the performance and cost are defined as $CriticalPerfFitness = f(CPerf)$ and $CriticalCostFitness = g(CCost)$ respectively where $CPerf$ and $CCost$ are user-specified performance and cost constraints. When solutions are optimized for prototyping time, the population characteristics can deteriorate in terms of cost and/or performance. Then, the population is again optimized for the metric violating the corresponding *optimizing condition*. In Figure 7, such conditions for the optimization state transition are denoted by P and C which are defined as follows:

$$AvgPerfFitness \geq CriticalPerfFitness \longrightarrow P = 1$$
$$AvgCostFitness \geq CriticalCostFitness \longrightarrow C = 1$$

In Figure 7, $P' = notP$ and $C' = notC$. An optimizing state transition occurs whenever a condition becomes true.

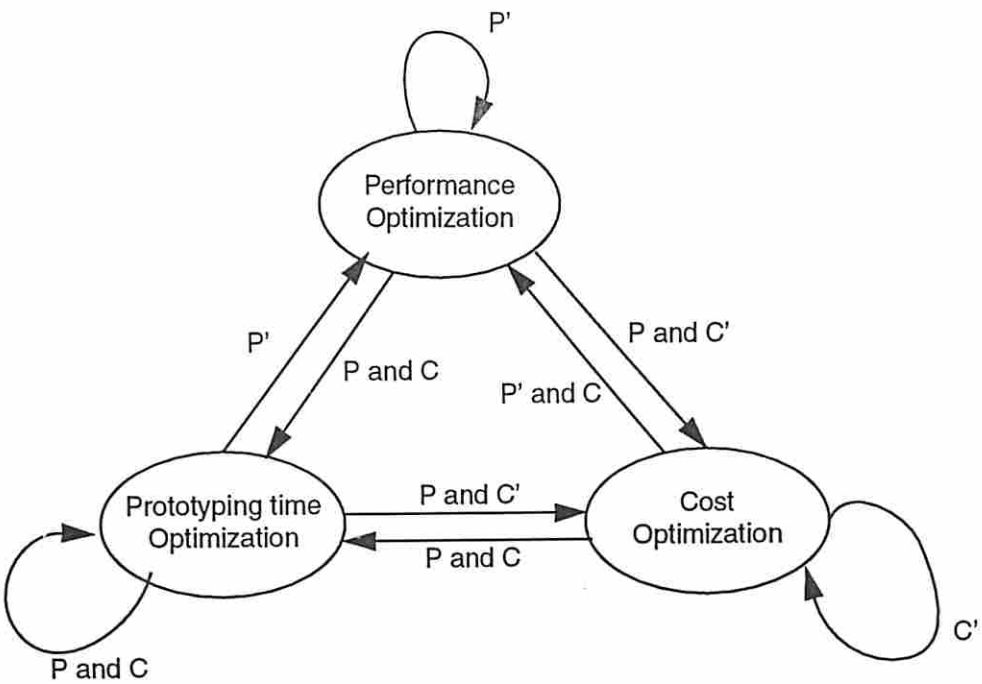


Figure 7: Optimization sequence in GARDEN

In Figure 8, the normalized average fitness values over generations is plotted. In Figure 8, B is the $h(TTM)$ of a best valid solution, $\bar{f}$ is the average $f(SPERF)$, $\bar{g}$ is the average $g(SCOST)$, and $\bar{h}$ is the average $h(TTM)$ of solutions in the population.

6 Experimental Results

We synthesized multichip system architectures for an MPEG encoder with both EDEN and GARDEN with different user-specified constraints. MPEG (Motion Picture Expert Group)

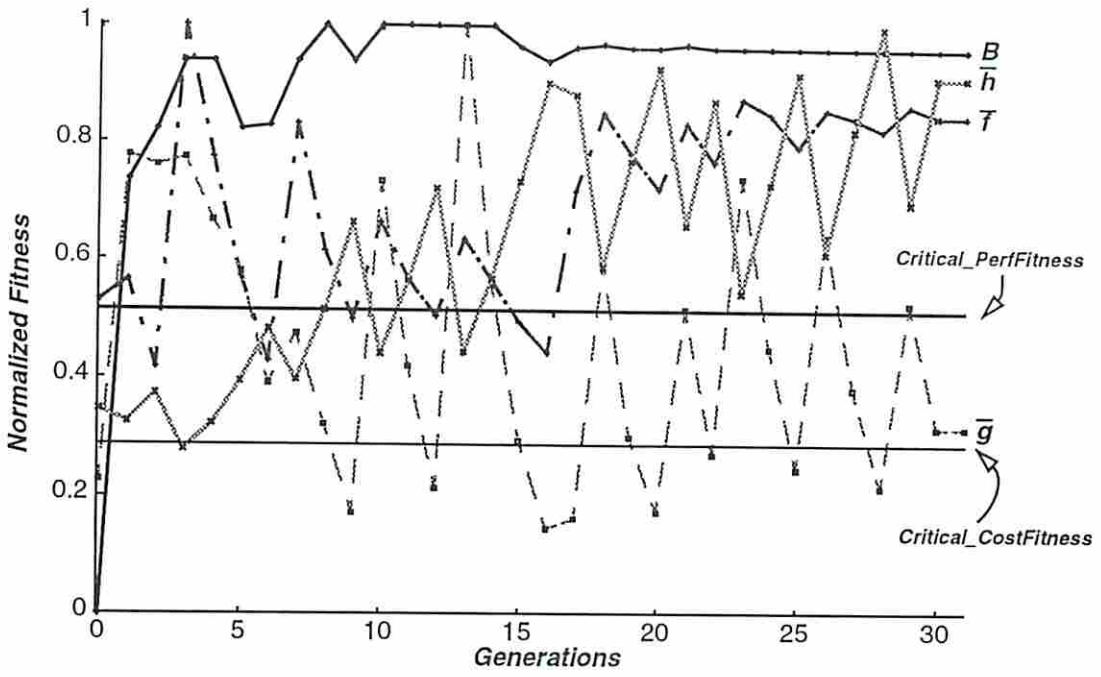


Figure 8: Normalized fitness functions

is an international standard on compression of motion video [24]. A simplified task-flow graph for the MPEG encoder is shown in Figure 9. The frame size for the 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. The volume of data on the edges of the task-flow graph is $16 \times 16 \times 8 = 2048$ bits.

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 FPGA (Xilinx XC4000 family). For each design style, parameters for different datapath architectures for each task are required to perform the implementation selection. For the COTS devices, this can be extracted from the documentation. If the function is a parameterized module, generators[25] can be used. Many researchers use a synthesis tool to obtain the exact parameters but it is very costly and time-consuming. In EDEN and GARDEN, we used a behavioral estimator called BEST[20]. By running BEST with module libraries in different design styles, we obtained the estimates on those parameters, from which we constructed the implementation library. 9 die types, 16 package types, and 12 bus types were used in the experiments.

The experiments with 6 different sets of user-specified constraints was carried out for an MPEG encoder. Table 1 shows typical statistics of 100 solutions EDEN produced for (a) unconstrained(Ex.1) (b) cost-constrained(Ex. 2,3) (c) performance-constrained(Ex. 4,5) and (d) cost-performance constrained optimization(Ex. 6). Equation 3 is modified as follows

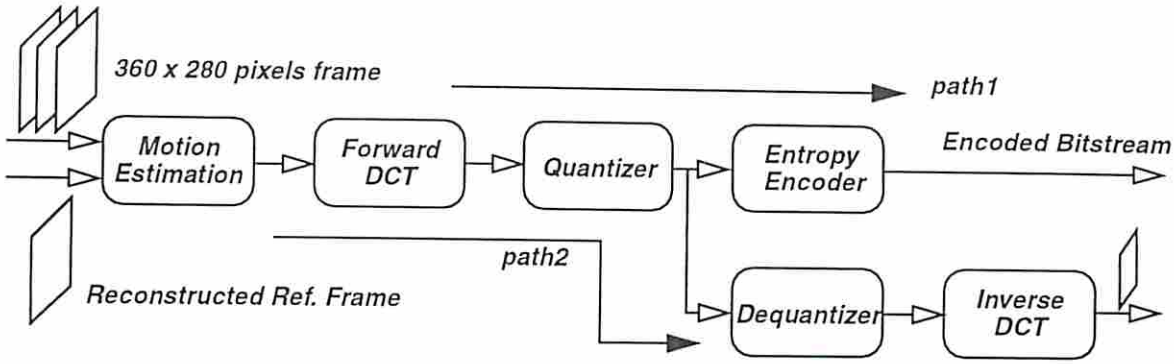


Figure 9: Task-flow graph for the MPEG encoder

and used as an objective function OF such that an architecture of smaller size is preferred when two architectures have the same TTM.

$$OF = \sum_{chip} TFab_{chip} * SIZE_{chip} + \sum_{chip} TPkg_{chip}$$

For the unconstrained experiment, in which the cost and performance constraints were set to a very large value, EDEN found an architecture that has 3 FPGAs and 2 gate array chips. The 2 gate array chips were for the encoder and decoder parts of the MPEG encoder system, for which FPGA implementation were not feasible from the behavioral estimation because of the long delay of the worst-case behavior. If there had existed an FPGA style implementation for each task, EDEN would have found a full FPGA architecture. As either the cost or performance constraint is tightened, EDEN attempts to reduce the cost and performance by employing more gate-array and standard-cell designs for implementing tasks. When only the performance is constrained(Ex. 2,3), EDEN improved the performance by using more standard-cell implementation styles but did not integrate the implementation to achieve lower cost. On the other hand, when the cost constraint was reduced(Ex. 4,5), EDEN not only used more slow and smaller standard-cell implementations but also integrated them into a single chip to reduce the cost of the system. When both cost and performance constraints were tight(Ex. 6), EDEN produced solutions in two chips; one has motion estimation and forward DCT tasks in the standard-cell style and the other has the rest of the tasks in the gate-array style. EDEN further integrated the design while employing a faster and bigger architecture for tasks.

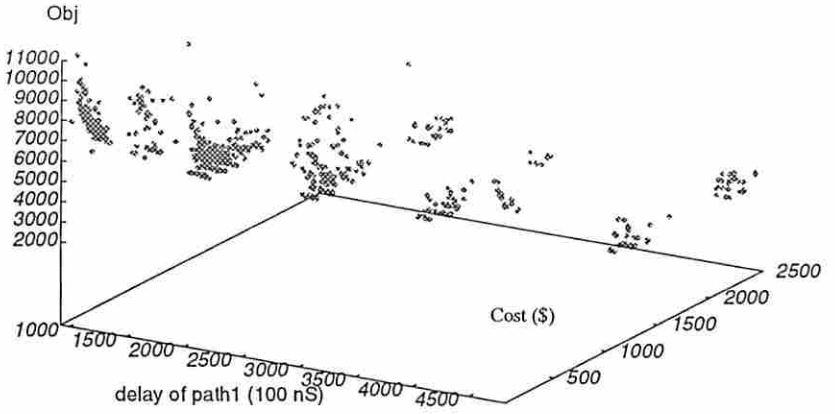
We conducted 6 GARDEN experiments with the same set of user constraints as with EDEN experiments. GARDEN was able to find solutions comparable to those from EDEN while the runtime for GARDEN to progress 31 generations was about 270 sec. and that for EDEN was about 5 to 10 hours. The distribution of solutions in the design space and an example of a solution generated by GARDEN are illustrated in Figure 10.

Exp. No.	Obj. Value	Cost	Delay (ns)		No. of Partitions			
			Path 1	Path	Total	FPGA	GA	SC
1	55	4105	499480	490720	5	3	2	0
2	2333	1891	188040	187840	4	1	1	2
3	2500	1247	159960	163800	5	1	1	3
4	82	1010	477240	480045	2	1	1	0
5	1415	446	492757	495717	2	0	1	1
6	2000	481	162921	162360	2	0	1	1

Table 1: Synthesized Designs for an MPEG Encoder by EDEN

Exp. No.	Obj. Value	Cost	Delay (ns)		No. of Partitions			
			Path 1	Path	Total	FPGA	GA	SC
1	169	3548	477700	487520	3	2	1	0
2	2515	631	173240	189880	4	2	1	1
3	3986	523	160080	164080	3	1	1	1
4	60	1984	480420	490240	4	3	1	0
5	3632	125	473540	481760	2	1	0	1
6	3717	125	107040	116800	2	0	1	1

Table 2: Synthesized Designs for an MPEG Encoder by GARDEN



(a) Distribution of solution characteristics in design Space for the 31st generation of Ex. 6

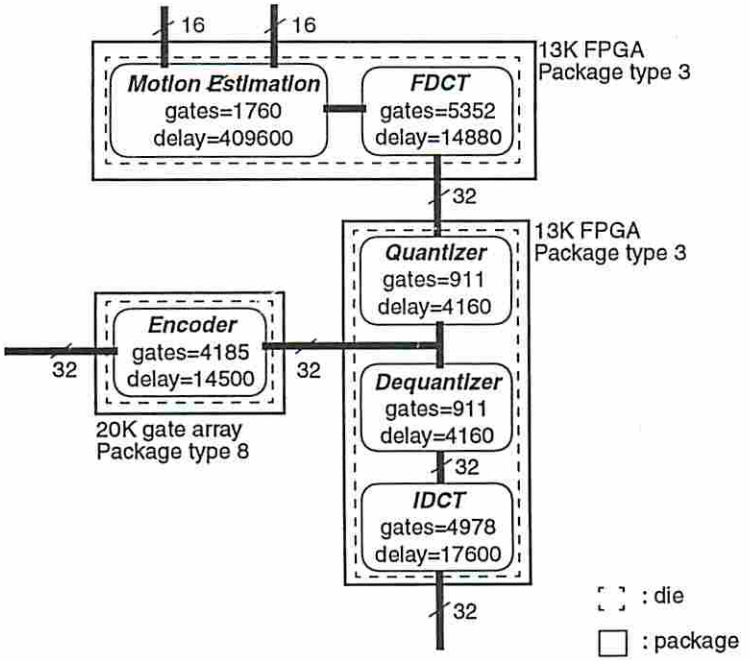


Figure 10: Distribution of solutions in the design space and an example solution from GARDEN

7 Conclusion

Multichip system architecture design is a complex problem composed of interrelated design subtasks, which shape the overall system characteristics. In order to automate multichip system architecture design, we developed a model that describes the interrelationships with a set of decisions and validity constraints to which decisions are subject.

Although we demonstrated the application for rapid prototyping, tools like EDEN and GARDEN can be used for a number of other purposes, for example, partitioning of a system into a number of FPGA chips for FPGA-based emulation or partitioning dies into a number of MCMs while optimizing the cost of a system with a further improvement. The GA-based tool GARDEN illustratesthe possibility of improving the existing design with a mix-and-match of promising designs. Such capability can be exploited in finding a better solution from solutions optimized for different system metrics.

The current model is not sophisticated enough to reflect all design concerns which the system designer is facing. The model for memory architecture and the consideration of the effect of the test circuits are a few examples of such concerns which have to be included in a multichip system architecture synthesis model to produce a more practical design and enhance the design exploration capability.

8 Acknowledgment

The authors would like to thank Xilinx and Compass for their contributions of tools and information for this research.

References

[1] R. Jain, A. C. Parker, and N. Park, "Predicting Area-Time Tradeoffs for Pipelined Design," in *Proc. of the 24th Design Automation Conf.*, pp. 35–41, IEEE and ACM, July 1987.

[2] F. Balasa, F. Catthoor, and H. D. Man, "Background memory area estimation for multidimensional signal processing systems," *IEEE Trans. on Very Large Scale Integration Systems*, June 1995.

[3] J. Vanhoof, K. Rompaey, I. Bolsens, G. Goossens, and H. DeMan, *High-Level Synthesis for Real-Time Digital Signal Processing.* Kluwer Academic Publishers, 1993.

[4] 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*, vol. 1, p. 268, September 1993.

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

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

[7] 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.

[8] M. A. Richard, "The rapid prototyping of application specific signal processors(RASSP) program:overview and status," in *Proceedings 1st Annual RASSP Conferences*, 1994.

[9] G. A. Shaw, J. C. Anderson, and V. K. Madisetti, "Accessing and improving current practice in the design of application-specific signal processors," in *Proc. of the Int'l. Conf. on Acoustics, Speech and Signal Processing*, 1995.

[10] R. Gupta and G. D. Micheli, "Hardware-software cosynthesis for digital systems," *IEEE Design & Test of Computer*, pp. 29–41, September 1993.

[11] 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.

[12] J. Adams and D. E. Thomas, "Multiple-process behavioral synthesis for mixed hardware/software systems," in *Proc. of 8th International Symposium on System Synthesis*, 1995.

[13] M. Srivastava and R. Brodersen, "SIERA: A unified framework for rapid-prototyping of system-level hardware and software," *IEEE Trans. on CAD*, vol. 14, p. 676, June 1995.

[14] S. Prakash and A. Parker, "Synthesis of Application-specific Multiprocessor systems including memory components," *Journal of VLSI Signal Processing*, vol. 8, pp. 97–116, October 1994.

[15] D. Gajski, S. Narayan, L. Ramachandran, and F. Vahid, "System design methodologies:aiming at the 100 h design cycle," *IEEE Trans. on Very Large Scale Integration Systems*, vol. 4, 1996.

[16] W. Wolf and R. Manno, "High-level modeling and synthesis of communicating processes using VHDL," *IEICE Trans. on Information and Systems*, vol. E76D, pp. 1039–1046, September 1993.

[17] M. Shih, E. Kuh, and R. Tsay, "System partitioning for multi-chip modules under timing and capacity constraints," in *Proc. of IEEE Multi-chip Module Conference*, 1992.

[18] J. Anderson, "Projecting RASSP benefits," in *Proceedings 2nd Annual RASSP Conference*, 1995.

[19] W. Bertram, "Yield and reliability," in *VLSI Technology* (S. M. Sze, ed.), p. 599, McGraw-Hill, 1983.

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

[21] Advanced Mathematical Software Ltd., *LAMPS USER GUIDE, Version 1.68*, 1994.

[22] Z. Michalewicz, *Genetic Algorithms + Datastruture = Evolution Programs.* Springer-Verlag, 1992.

[23] D. Whitley, "A genetic algorithm tutorial," Tech. Rep. CS-93-103, Colorado State University, 1993.

[24] D. L. Gall, "MPEG:a video compression standard for multimedia application," *Communications ACM*, vol. 34, April 1991.

[25] S. R. Powell and T. M. Cesear, "Rapid design and exploration of signal processing systems using a VHDL model generator based paradigm," in *Proceedings 2nd Annual RASSP Conference*, 1995.