FD-HGAC: A Hybrid Heuristic/Genetic Algorithm Hardware/Software Co-synthesis Framework with Fault Detection

John Conner[1*], Yuan Xie[1], Mahmut Kandemir[1], Robert Dick[2], and Greg Link[1]

[1]The Pennsylvania State University — {*jconner*,yuanxie,kandemir,link}@cse.psu.edu
[2]Northwestern University — dickrp@ece.northwestern.edu

Abstract— **Embedded real-time systems are becoming increasingly complex. To combat the rising design cost of those systems, co-synthesis tools that map tasks to systems containing both software and specialized hardware have been developed. As system transient fault rates increase due to technology scaling, embedded systems must be designed in fault tolerant ways to maintain system reliability. This paper presents and analyzes FD-HGAC, a tool using a genetic algorithm and heuristics to design real-time systems with partial fault detection. Results of numerous trials of the tool are shown to produce systems with average 22% detection coverage that incurs no cost or performance penalty.**

I. Introduction

As technology scales and computer systems become more complex, the design time and engineering cost of embedded systems is increasing dramatically. Hardware/software co-synthesis is the automated design of special-purpose hardware/software systems. It may be used to combat the rising complexity of designing system-on-chip-style real-time embedded systems. By mapping a description of system functions onto a selection of general-purpose and special-purpose processing elements, co-synthesis tools design low-cost, heterogeneous, real-time computer systems without requiring the user to specify which tasks are mapped into hardware and software.

FD-HGAC, the Fault Detecting Hybrid Genetic Algorithm Co-synthesis Framework, is a tool that implements a set of algorithms to perform hardware/software co-synthesis with the incorporation of system features to detect transient faults. Because of the pervasive trend of technology to experience greater numbers of transient faults as system density increases and transistor size decreases, fault detection will be a vital feature of future computer systems. Moreover, the co-synthesis problem is NP-complete, so optimally solving large problem instances is intractable. FD-HGAC combines a stochastic genetic algorithm with two deterministic heuristics. By running the fast heuristic inside the genetic algorithm, FD-HGAC designs better systems than a deterministic heuristic, but runs in less time than a guaranteed-optimal design tool.

FD-HGAC uses a fault detection methodology that inserts redundant tasks into an pre-designed schedule. This allows the algorithm to focus on designing low cost, high performance systems while also increasing system reliability markedly over the base case. The insertion of fault-finding nodes into the system has the effect of increasing reliability without impacting system cost or performance.

This paper contributes to the field of co-synthesis by

1. Presenting a design tool that produces real-time, partially fault-tolerant systems with good cost and performance.
2. Significantly generalizing previous work.
3. Adapting co-synthesis techniques for low run time and parallel scalability.

This paper is organized as follows: First, we discuss previous work. Then, co-synthesis is explained and a recurring example is introduced in Figs. 1–4. Then, details on the implementation are given, followed by experimental results and analysis. Finally, possibilities for future work and conclusions are discussed.

II. Previous Work

R. Dick introduced a GA co-synthesis tool that optimizes part selection and scheduling simultaneously [3]. Chakraverty's work on GA co-synthesis uses stochastic constraints and implements co-synthesis by handling each design step in sequence instead of simultaneously [1]. FD-HGAC differs from these by including system-level fault detection and by avoiding stability issues that are present in pure-GA tools. Also, B. Dave introduced a heuristic tool for fault tolerant co-synthesis [2] that uses additional hardware to produce redundancy. FD-HGAC does not add hardware in this fashion.

FD-HGAC's heuristics are improved from work done with heuristic co-synthesis [8, 9]. The scheduling and allocation heuristic is expanded in FD-HGAC over the previous work to function on all types of processing elements instead of only general-purpose types [9]. Similarly, the fault detection algorithm in FD-HGAC is a faster adaptation of that in [8].

III. Problem Formulation

In hardware/software co-synthesis, a system description is mapped onto a collection of processing elements (PEs) [7] that are either general-purpose processors or special-purpose processing elements, such as digital signal processors or arithmetic units. Using a description of the work that a system must
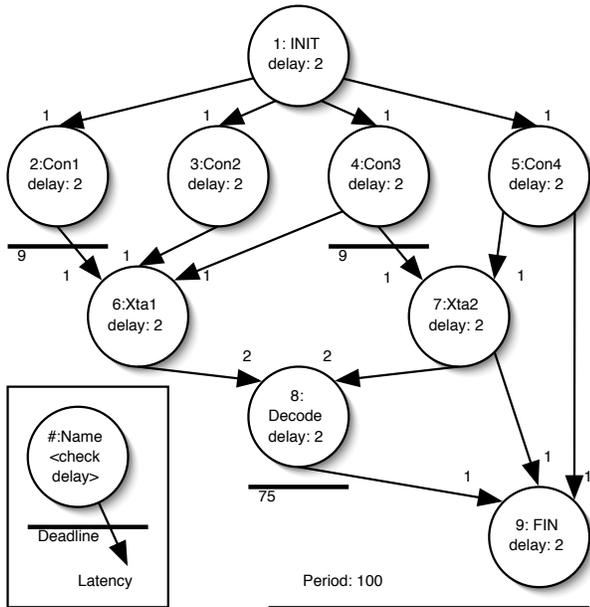
Fig. 1. Example Task Graph

PE	Name	$	1	2	3	4	5	6	7	8	9
1	CPU1	15	6	9	9	9	9	16	16	200	5
2	CPU2	30	3	6	6	6	6	10	10	75	2
3	CONV1	5		5	5	5	5				
4	CONV2	15		2	2	2	2				
5	DEC	10								20	

Fig. 2. Example PE Library - Numbered columns are execution latencies

perform and of the capabilities of the various PEs, the tool allocates tasks either as hardware on special-purpose PEs or as software on general-purpose PEs. The system has constraints on data transfer between tasks and timing deadlines. In addition, the description includes information that describes the execution latency of each task on each PE.

A. The Task Graph and Processing Element Library

The system specification used in this work is a directed, acyclic task graph, a common model used in the field [7]. The graph models the functionality of the system in which data arrives through source nodes and leaves through terminal nodes. The nodes represent computational tasks that must be performed by timing deadlines and which are repeated at a rate defined by the system period. Tasks in this work are general and contain only timing and data dependence information.

An example task graph is shown in Fig. 1. Each node in the figure contains an ID number, name, and checking delay, which is the time necessary to compare the output of the node with the output of a duplicate node. Edges between nodes represent data dependencies with the labels indicating the data transfer time if the producer and consumer must exchange data between PEs. Black bars indicate hard timing deadlines.

The PE library describes the available hardware resources that the tool may use in designs. Each PE models a particular available functional unit of any scale from simple on-chip adders to complex separately-packaged CPUs. Each PE has an ID number, name, cost, and execution latencies for each task in the task graph. An example PE library is shown in Fig. 2. Each row describes a different PE and gives the dollar cost and the execution latencies for the various tasks from Fig. 1. A PE cannot handle tasks with no latency listed.

B. The Co-Synthesis Process

Co-synthesis has five steps, with a sixth step in FD-HGAC:

1. A selection of hardware is chosen from the PE library.
2. Bus hardware is selected from the bus library.
3. The hardware is connected by the chosen busses under contention constraints.
4. Tasks are assigned to instances of PEs.
5. Tasks are scheduled to run on their assigned PEs.
6. Redundant copies of tasks and result checkers are inserted into idle time slots (slack) in the schedules.

C. Fault Detection in Co-Synthesis

To detect faults, it is necessary to replicate some tasks in the schedule in order to compare the outputs for correctness. There are two major ways to introduce redundancy in in a schedule: Duplicate tasks can be scheduled without considering them differently than necessary tasks, or the duplicates can be inserted into the schedule after all necessary tasks are present. FD-HGAC chooses the latter method because it is cheaper.

IV. THE FD-HGAC IMPLEMENTATION

FD-HGAC is implemented in C++ using the GALib library [6]. Its algorithm is a nesting in which the GA does PE selection and its objective function calls the allocation and scheduling heuristic to schedule tasks and insert duplicate tasks for fault tolerance. This allows many of the benefits of the genetic algorithm to come through in the algorithm and allows the objective function to be fast. Bus selection and PE interconnection are idealized and discussed later.

A. The Genetic Algorithm

GAs are a type of stochastic optimization algorithm which takes a group of potential solutions, i.e., a population of chromosomes, and improves the population quality through successive generations. Each generation requires evaluating the fitness of each chromosome using an "objective function" and performing randomized operations, crossover and mutation, on the population to improve its aggregate fitness. In the interest of space, this paper does not discuss GAs in detail, instead deferring to the standard text [5].

FD-HGAC uses a standard GA to select system hardware from the PE library. The chromosome structure is an array of integers that encode the quantity of each PE type that is present

```
      PE 1   PE 2   PE 3   PE 4   PE 5
       0      1      2      0      1
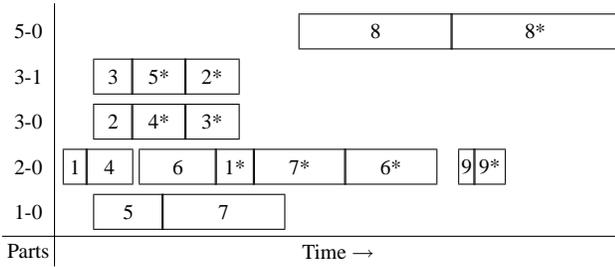```

Fig. 3. Example Result Chromosome



Fig. 4. Example Result Schedule — Each boxed number represents a scheduled task. Starred tasks are duplicates. This schedule is 100% reliable.

in the system. As a special case, an additional PE 1 is always added to the system. This guarantees that all possible chromosomes are valid, as PE 1 is required to be a general-purpose PE. For example, the chromosome shown in Fig. 3 encodes for one of PE 1 (because of the special case), one of PE 2, two of PE 3, zero of PE 4, and one of PE 5. The resulting schedule from this chromosome for the example task graph and PE library is shown in Fig. 4. The sample schedule shows complete task duplication, and is cost-optimal under the deadline constraints with a cost of 65 units.

FD-HGAC's objective function takes into account the cost of the chromosome's encoded PEs and the performance of the schedule that fits the chromosome. The algorithm for this is simple: The objective function calls the scheduler and returns the reciprocal of the sum of the encoded PEs, penalized if the schedule does not meet all deadlines. A typical penalty for a non-deadline-conformant schedule is 90%. This extreme bias of preferring performance over cost assures that the system produced will meet the timing constraints if at all possible, but introduces some significant difficulties in the objective function that can hinder convergence to an optimal solution.

A significant idealization that FD-HGAC uses is that it assumes a strongly connected, no-cost, high-bandwidth bus topology between PEs. Tasks do have data transfer latencies (see Fig. 1), and it is assumed that a bus will always be available for one task to talk to another and that no contention for busses will occur. The impact of the idealized bus model can be minimized by adding a constant transfer latency penalty between tasks that is large enough to approximate the realistically random bus usage. This is a strong direction for future work.

### B. The Allocation and Scheduling Heuristic

The allocation and scheduling heuristic is called from the GA's objective function as a vital step in evaluating the fitness of a specific hardware selection. Because the scheduling heuristic is called many times per execution, it must be very fast. In addition, the heuristic must also produce high quality

results, as the GA cannot produce a high quality hardware selection unless the results of the heuristic are as close to optimal as reasonable.

FD-HGAC uses a heuristic modified from [9] to schedule instead of handling all scheduling and allocation in the GA because a heuristic allows the objective function to be small and fast. Complex techniques to avoid producing many invalid schedules in pure-GA optimizers [3] complicate the objective function and slow down the algorithm. The heuristic allows FD-HGAC to be sure that all chromosomes are valid.

### C. The Task Replication Heuristic

The task replication heuristic in FD-HGAC is adapted from [8]. The heuristic works by ranking all tasks according to criticality, finding the slack in the cost-optimal schedule, and filling the slack with redundant tasks in the order of decreasing criticality. Criticality is measured as the number of dependent nodes a particular node has. An idle slot in the schedule (slack) is selected for a task of particular criticality by checking for slots that meet dependency requirements for the task, and then choosing the slot that results in the earliest completion time for the redundant task. The algorithm attempts to replicate all tasks in turn, and ends when no more tasks can be replicated, even if some tasks have no duplicates. The checking itself is considered to be outside the scope of the cosynthesis problem in FD-HGAC. Time is left in the schedule via the checking delay on each task for an external system to capture and compare the task results.

## V. EXPERIMENTAL ANALYSIS

### A. Experimental Platform

Example runs were performed with fixed GA parameters except generation count. Population size was 200, crossover rate was 90%, mutation rate was 0.5%, and genes were seeded uniformly randomly from 0 to 5. These parameters were found to provide adequate results for a variety of inputs, including the nine sample inputs used for benchmarking. The input files were provided by the author of [8] and contain task graphs generated using [4]. The sample input specifications are summarized in Table I. The "Deadline" column in that table lists the hard deadline for completion of all tasks in the graph.

### B. Experimental Results and Analysis

Costs and fault detection coverage percentages for tests with different generation counts are provided in Table II. All tests were averaged over a minimum of five trials. For all trials, the performance deadlines of the system were met, making the end results comparable on cost and detection coverage. Cost is the total cost of PEs in the designs. Fault detection coverage is defined as the percentage of total system PE run time in which the system is not vulnerable to transient faults, calculated according to the following equation, counting all tasks

TABLE I
TABULATION OF SAMPLE INPUTS

| # | Tasks | Edges | CPUs | ASICS | Deadline |
|---|-------|-------|------|-------|----------|
| 1 | 15 | 30 | 3 | 30 | 625 |
| 2 | 18 | 37 | 2 | 36 | 700 |
| 3 | 23 | 50 | 3 | 46 | 625 |
| 4 | 26 | 57 | 3 | 52 | 750 |
| 5 | 32 | 69 | 3 | 64 | 600 |
| 6 | 39 | 91 | 2 | 78 | 1000 |
| 7 | 40 | 85 | 3 | 80 | 800 |
| 8 | 44 | 95 | 3 | 88 | 1000 |
| 9 | 51 | 110 | 3 | 102 | 1000 |

TABLE II
TABULATION OF RESULTS

| | 1K Gen. | | 5K Gen. | | 10K Gen. | |
|---|---------|-------|---------|-------|----------|-------|
| # | Cost | %Det. | Cost | %Det. | Cost | %Det. |
| 1 | 94$ | 29% | 94$ | 31% | 94$ | 31% |
| 2 | 102 | 17 | 100 | 11 | 104 | 21 |
| 3 | 155 | 24 | 153 | 26 | 152 | 25 |
| 4 | 148 | 20 | 141 | 18 | 143 | 21 |
| 5 | 217 | 34 | 201 | 26 | 209 | 26 |
| 6 | 209 | 19 | 202 | 18 | 206 | 19 |
| 7 | 235 | 27 | 214 | 19 | 216 | 15 |
| 8 | 209 | 20 | 191 | 12 | 196 | 12 |
| 9 | 279 | 22 | 258 | 18 | 239 | 13 |

separately, including those that overlap.

$$\%detection = 1 - \frac{\sum_{unduplicated} time}{\sum_{all} time}$$

Run times are dominated by the many iterations of the heuristics and range from one minute to two hours for the provided examples on a 1 GHz computer. Run times scale sub-linearly with increasing generation count due to a software cache and scale super-linearly with increasing PE library size, task count, and edge count. The task graphs used to benchmark FD-HGAC are realistically-sized and the tool is expected to scale well to much larger task graphs because precision and runtime are user-controllable through the GA parameters. In situations where a minimal runtime is desirable, the user can scale back the generation count and get a reasonable result quickly.

Fault detection coverage averages 22% for the samples. The results show that the cost of a system is related to the amount of fault detection that can be placed in the schedule. This is because expensive solutions will, in general, have more slack in their schedules, allowing for more room for redundancy. Trials in which the input task graph was explicitly altered to provide 100% redundancy for the system produced full coverage with an optimistic cost increase of 30–60%, averaging about 50%.

FD-HGAC exhibits some pathological scenarios in which timing-conformant solutions require large numbers of specific parts. In this situation, chromosomes with insufficient parts are

cheaper and equally good in relation to a valid solution because they do not meet timing, so the non-conforming chromosomes are selected against and cannot improve. As a result, the GA will select as best an inexpensive chromosome that will not meet timing constraints. This can be treated somewhat by using a larger and more diverse population, and future work will look into solving this problem.

## VI. CONCLUSIONS AND FUTURE WORK

In this paper, we have shown that a combined genetic algorithm and heuristic approach to designing partially fault-tolerant systems is practical and efficient. Results show excellent cost and performance characteristics and include, on average, twenty-percent temporal coverage of transient faults at no increase in cost over the performance-optimal case, compared with an average fifty percent increase in cost for complete fault detection. Run times are non-negligible but reasonable, and are easily controlled by the user at the expense of solution quality. Additionally, the algorithm may be easily parallelized with near-linear speedup.

There are many promising directions for future work on this topic. Complete bus modeling is a very strong direction for progress, and a solution to the previously explained pathological scenario must be found. The quality of the scheduling could be increased by using a more advanced duplicate task insertion heuristic. Additionally, other styles of GA-like stochastic optimizers could be explored for use in a similar tool.

## REFERENCES

[1] S. Chakraverty. Cosynthesis of multiprocessor architectures with high availability. In *Proceedings of the 17th International Conference on VLSI Design*, pages 927–932. IEEE, IEEE Computer Society, January 2004.

[2] B. P. Dave. *Hardware/Software Co-Design of Hetrogeneous Real-Time Distributed Embedded Systems*. PhD thesis, Princeton University, June 1998.

[3] R. P. Dick. *Multiobjective Synthesis of Low-Power Real-Time Distributed Embedded Systems*. PhD thesis, Princeton University, November 2002.

[4] R. P. Dick, D. L. Rhodes, and W. Wolf. TGFF: task graphs for free. In *Proceedings of the 6th International Workshop on Hardware/Software Codesign*, pages 97–101. IEEE Computer Society, 1998.

[5] D. E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison Wesley, Boston, 1989.

[6] Matthew Wall, 1995-1999. http://lancet.mit.edu/ga/.

[7] J. Staunstrup and W. Wolf, editors. *Hardware/Software Co-Design: Principles and Practice*. Kluwer Academic Publishers, Norwell, MA, 1997.

[8] Y. Xie, L. Li, M. Kandemir, N. Vijaykrishnan, and M. J. Irwin. Reliability-aware co-synthesis for embedded systems. In *ASAP 2004 Proceedings*, pages 41–50. IEEE, September 2004.

[9] Y. Xie and W. Wolf. ASICosyn: Co-synthesis of conditional task graphs with custom ASICs. In *ASIC 2001 Proceedings*, pages 130–135. IEEE, October 2001.