Simulated Evolution (SimE) Based Embedded System Synthesis Algorithm for Electric Circuit Units (ECUs)

Umair F. Siddiqi[1], Yoichi Shiraishi[1], Mona A. El-Dahb[1], and Sadiq M. Sait[2]

[1] Department of Production Science & Technology, Gunma University, Japan
{umair,siraisi,mona}@emb.cs.gunma-u.ac.jp
[2] King Fahd University of Petroleum & Minerals, Dhahran, Saudi Arabia
sadiq@kfupm.edu.sa

Abstract. ECU (Electric Circuit Unit) is a type of embedded system that is used in automobiles to perform different functions. The synthesis process of ECU requires that the hardware should be optimized for cost, power consumption and provides fault tolerance as many applications are related to car safety systems. This paper presents a Simulated Evolution (SimE) based multiobjective optimization algorithm to perform the ECU synthesis. The optimization objectives are: optimizing hardware cost, power consumption and also provides fault tolerance from single faults. The performance of the proposed algorithm is measured and compared with Parallel Re-combinative Simulated Annealing (PRSA) and Genetic Algorithm (GA). The comparison results show that the proposed algorithm has an execution time that is 5.19 and 1.15 times lesser, and cost of the synthesized hardware that is 3.35 and 2.73 times lesser than the PRSA and GA. The power consumption of the PRSA and GA (without fault tolerance) are 0.94 and 0.68 times of the proposed algorithm with fault tolerance.

Keywords: Electric Circuit Unit, Embedded Systems, Synthesis, allocation, assignment, scheduling, Simulated Evolution.

1 Introduction

In automotive electronics, Electric Circuit Unit (ECU) refers to any embedded system that is responsible for controlling one or more electrical systems or subsystems. Automobiles can have from dozens to hundreds of ECUs. The ECUs are generally partitioned by their domains. The two broad classes of ECUs are: (1) hard real-time control of mechanical parts, and information-entertainment, and (2) information management, navigation, computing, external communication and entertainment [1]. Examples of ECUs include the following: Engine control unit that controls the fuel injection, ignition timings, idle speed, valve timing, and electronic valve. On-board diagnostics is another type of ECU that is responsible for controlling and managing the vehicles' self-diagnostic and reporting

A. Dobnikar, U. Lotrič, and B. Šter (Eds.): ICANNGA 2011, Part I, LNCS 6593, pp. 400–409, 2011.

system. ECU for transmission control uses sensors from the vehicle and data provided by the ECU to determine when and how to change gears in an automobile for optimum performance, fuel efficiency and shift quality. The propulsion system of Electric Vehicles (EV) uses microprocessors, digital signal processors (DSPs), and microcontrollers. In all types of automobiles the ECU plays an important role and improvement in its design brings an overall improvement.

Recently, new applications have been deployed in automobiles and the complexity of the ECU design increases from a single microcontroller to a system that includes processors, Field Programmable Gate Arrays (FPGAs) and/or Application Specific Integrated Circuits (ASICs). Some examples of new applications for the ECUs are: Driver Assistance (DA) functions which help drivers in parking vehicles, maintaining safe distance from other cars, inform the drivers from threats that they cannot see, and many other useful tasks. The DA functions use a number of near-range sensors (ultrasonic & cameras) that are located all around the vehicle. Similarly, the night vision systems use cameras with infrared (IR) illumination and thermal imaging technology and also use the navigation system display of the automobile to show heat-sensitive or IR images to the driver. In all such applications, ECUs are responsible for processing the signals and generating responses.

The requirements for effective real time operation of most of the types of ECUs are: meeting hard time deadlines, fault-tolerance, minimizing power, and minimizing the hardware cost. The proposed synthesis process is designed to optimize the same four parameters. The proposed synthesis process is referred to as ECU synthesis process in the rest of the paper.

The inputs to the synthesis process are directed task graphs in which the nodes represent the tasks or computation and edges represent the communication between the tasks. The process of synthesis of embedded systems consists of three sub-problems: (a) Allocation, (b) Assignment, and (c) Scheduling. The allocation refers to selecting the suitable hardware units (also called as processing elements (PEs)) and communication resources (CRs). The assignment process involves binding the tasks to specific PEs and edges to CRs. If the nodes at the two ends of any edge are assigned to the same PE then no CR is required for that edge. The scheduling problem is to find the order of execution of tasks on each PE. In our research we allowed dedicated connection between PEs, therefore, the CRs do not require scheduling. The assignment/allocation and scheduling problems are known to be NP-Complete [3] [4]. Fault tolerance is an important feature of ECUs in which the task graphs will complete their execution and met hard time deadlines even if faults occurs during the execution. Many ECUs perform safety related functions like DA functions and fault tolerance can enable the task graphs to complete their execution even if faults occurs during the execution. Faults can be of two types: (1) Permanent faults, and (2) Transient and intermittent faults. The permanent faults includes microprocessor failure, etc. The tolerance from permanent faults is provided through redundant hardware. Transient faults can occur due to electromagnetic interference and intermittent faults can appear and disappear repeatedly. The behavior of transient

and intermittent faults appear very similar to the fault tolerance techniques and therefore, same methods can provide tolerance from both these types of faults. The ratio of transient and intermittent faults to permanent faults is 100:1, and therefore the former two types form the majority of the faults. The hardware solution that provides tolerance from n permanent faults also provide tolerance from n transient and intermittent faults. However, the solutions for tolerance from permanent faults are expensive and the transient and intermittent faults are much more frequent then the permanent faults therefore, separate methods are developed to provide tolerance from transient and intermittent faults.

In this paper, the authors have introduced a method of synthesizing optimized ECU hardware for the input task graph with hard time deadlines. The proposed method optimizes the system cost and energy consumption while ensures that all hard time deadlines are met and implements fault tolerance from single fault per task. The input is any task graph with information on time deadlines, execution times and energy consumption of PEs and time delays of CRs. The output from the method is the assignment of tasks and edges to the PEs and CRs and the order of execution (or schedule) in which the tasks will be executed in each PE. All outputs are valid solutions in which no hard time deadline is violated. SimE is found to be very effective in solving circuit design problems [6] [7] with less computational and memory requirements. The work in this paper is unique from previous works because it evaluated Simulated Evolution (SimE) [5] to solve the multi-objective ECU synthesis problem. The SimE maintains one solution at a time. In the previous works, the algorithms like GA or PRSA that maintains several solutions at a time were considered. This paper shows that SimE can be used to solve ECU synthesis problems and can yield good results. SimE is less computational as compared to GA or PRSA.

The paper is organized as follows: Section 2 will provide a survey of the relevant previous work. Section 3 contains a detailed description of the proposed method. Simulations and comparison of the proposed method with another method is shown in section 4. Finally, conclusions will be presented in section 5.

2 Previous Work

This section will summarize some of the previous work that is most relevant to the proposed method. SLOPES by L. Shang, R.Dick, and N. Jha [2] is a hardware-software cosynthesis algorithm that uses Parallel Re-combinative Simulated Annealing (PRSA) [8] to solve the allocation/assignment problem. Their algorithm maintains library of hardware units like processors, memories, dynamically reconfigurable FPGAs, and communication resources. The FPGA can configure any task by loading the appropriate bit stream from external memory. The reconfiguration latency of tasks is called reconfiguration time. They solved the scheduling problem by two approaches. In the first approach the scheduling sequence i.e. the order in which the tasks are scheduled is based on static slack based priority. The second approach that is recommended for FPGAs, the scheduling sequence is determined dynamically by task priorities that

consider both real-time constraints and frame-by-frame reconfiguration overhead information. Vida et al. [9] proposed the use of Strength Pareto Evolutionary Algorithm (SPEA) [10] that is a multi-objective optimization heuristics. The objective of their synthesis process is to simultaneously minimize multiple objectives for which the corresponding values cannot be improved without degradation in another. The SPEA uses Pareto dominance in calculating fitness function, and maintaining population density. They also used clustering to reduce the complexity of assignment and scheduling problems. In clustering, all the tasks that should be mapped to the same PE are included in one cluster. Kuchcinski proposed to solve the problem of embedded system synthesis using constraint logic programming [11]. They consider two types of constraints, the first type belongs to the system functional specification and the second type belongs to the non-functional requirements. The system is represented by the constraints over finite domain variables (FDV). Each FDV eventually obtained an integer value that specifies a solution. Different constraints solving techniques are used to find optimal and suboptimal solutions in which the given constraints are satisfied and the cost function is optimized. P. Eles et al. [12] proposed a Tabu Search [5] based method that optimizes the assignment of fault tolerance policy i.e., rollback recovery with checkpointing, active replication, passive replication, or a combination of the techniques to each process and also optimizes the mapping of processes to PEs (or hardware nodes). The method finds the upper bound for the number of checkpoints using formula derived by Punnekkat et al. [13] and the Tabu search based method performs the further optimization. The method yields solutions in which fault tolerance is provided from upto k (where $k > 0$) faults and the hard time deadlines are met and the cost constraints are satisfied.

3 Proposed Algorithm

This section describes the method proposed for the synthesis of ECU. As described in the first section the input consists of task graphs with hard time deadlines. The rest of the section shows the design of SimE for the ECU synthesis problem. The proposed method uses two types of PEs: (a) Uniprocessor PEs, in which only one task can be executed at a time. (b) Multiprocessor PE, in which n number of tasks can be executed concurrently, where n is the number of processors. FPGAs can also fall into the category of multiprocessors when several tasks are configured in them in parallel.

3.1 SimE for the ECU Synthesis Problem

SimE is an iterative heuristic for the combinatorial optimization problems. It is a general randomized search algorithm that is based on concepts learned from biological evolution. SimE can be tailored for specific applications. This paper shows SimE tailored for solving the allocation/assignment problem in the ECU Synthesis. The allocation/assignment problem in ECU Synthesis is NP-complete optimization problem in which the assignment of tasks and edges to different PEs

B= input value ∈[-0.2,0.2], VS = null
ϕ = Population of all tasks & edges in the task graph
Initialization: ϕ= Random_Assignment(ϕ)
while (stopping criteria is not met) {
    Evaluation: for i=0 to ϕ_{size}
        $find\_goodness(m_i)$; $m_i \in \phi$,
    Selection: for i=0 to ϕ_{size}
        S= S ∪ m_i if Selection(m_i) returns true
    Scheduling: Schedule tasks in each PE
    if check_validity(ϕ)=Yes then VS=VS ∪ ϕ
    Sort(S)
    Allocation: for i=0 to S_{size}
        Allocation(n_i), $n_i \in$ S }
where ϕ_{size}= total elements in the population,
S_{size}= total elements in the set S

Fig. 1. Simulated Evolution (SimE) Algorithm with time deadline checking

and CRs need to be optimized in order to meet the real time deadlines and minimize the system cost and energy consumption. The PEs and CRs are randomly selected from the library (allocation). However, the goodness measure in the SimE favours selection of PEs and CRs that benefits the optimization objectives. The SimE algorithm is shown in Fig. 1. In SimE, the population consists of all tasks and edges in the task graph. It maintains one solution at a time. However, as shown in Fig.1 all valid solutions can be stored into a set VS. A solution is valid if it does not violate any hard time deadline. The algorithm starts with an initial solution in which the tasks and edges are randomly assigned to any PEs and CRs. This is done through the function *Random_Assignment*. The value of B is input to the algorithm, B value can vary from [-0.2 to 0.2]. The algorithm loop continues until one of the stopping criteria can be met. The criteria could be the maximum number of valid solutions obtained. In the loop, the first step is the calculation of evaluation function. In which the goodness of each element in the population is calculated. The goodness function is described in Fig.2. The goodness function is based on finding ranking of different quantities. The factor[0] is the rank that is given by the number of alternate assignments for the node or edge i that are not better than the current assignment in terms of execution time or time delay, or in other words the current assignment for the node or edge i is better than that many number of other assignments. In the same way, factor[1] is the rank of hardware cost that is associated with the node

factor[0] = rank($Exec\_Time_i or Delay_i$)
factor[1] = rank(HW_Cost_i)
factor[2] = rank($Power_i$)
factor[3] = rank(Minimum($\frac{T_p}{M_p}$, $\frac{M_p}{T_p}$))
$goodness_i = \frac{1}{4}(\sum_{i=0}^{3} \frac{factor[i]}{Total\ number\ of\ PEs\ or\ CRs})$
where rank(x) function returns the number of possible assignments to other PEs or CRs that are better then the current assignment
$T_p = Number\ of\ tasks\ ready\ for\ Parallel\ Execution$
$M_p = Maximum\ number\ of\ tasks\ that\ can\ execute\ in\ parallel\ on\ the\ PE$
$goodness_i$ value ranges from 0 to 1

Fig. 2. Ranking based goodness function

or edge i, and factor[2] is the rank of energy consumption of the node i (factor[2]=0 & factor[3]=0 for edges). The factor[3] is the rank of the ratio between the number of tasks that can execute in parallel (i.e. the tasks that are independent from each other and also ready for execution at the same time) to the parallel processing capability of the assigned PE_i. The minimum value is taken between the ratio and its inversion that is to make sure that the value remains between 0 to 1 and 1 is the optimum value when the PE's allowed parallelism is fully used. The total goodness value can vary from 0 to 1. The next step in the SimE is to apply Selection function on all elements of the population. The selection function is given as, Selection function: if (random $<$ 1 - $goodness_i$ + B) return true else return false. *random* refers to a any number that is randomly generated within the range of 0 to 1. $goodness_i$ is the goodness of the node or edge i. The nodes or edges for which the selection function returns true will be selected into the set S. In the next step, the tasks in each PE are scheduled according to the scheduling scheme. The scheduling scheme will be described in the next subsection. The next step is to find the validity of the solution by checking all hard time deadlines. If all deadlines are met then the solution is placed in the set VS. The next step is sorting, in which the nodes and edges in the set S are sorted in the ascending order of their goodness. The allocation is the next step, in which for each node and edge in the set S an exhaustive search [14] is carried out that compares their current goodness values with the goodness values of other possible assignments. At the end of the search the node or edge can either be re-assigned to the PE or CR that has maximum goodness value with probability p_i or reassigned randomly to any PE or CR. The p_i should be kept high (e.g. p_i=0.90).

3.2 Scheduling Algorithms

The static slack based scheduling scheme [11] is used. In which the Earliest Start Time (EST) and Latest Start Time (LST) for the node is calculated using forward and reverse topological sorts. The idea behind the scheme is to assign low priority to tasks that can tolerate some delay in their execution. The processor scheduling scheme is shown below, in which priority of a task i is given by: $Priority_i = -(EST_i + LST_i)$.

3.3 Fault Tolerance (FT) Schemes

Different techniques exist to provide tolerance from transient and intermittent faults. The work in this paper uses two techniques namely: Rollback Recovery with Checkpointing (RRC) and Active Replication (AR). In RRC, the task is checkpointed at one or more points. When a checkpoint is reached during execution then the values of the inputs to that checkpoint are stored in a separate static memory. If any fault occurs after the checkpoint then the task can be reexecuted from the last checkpoint using the values stored in the static memory. In that way, the reexecution time decreases as compared to executing the complete task after fault occurrence. In AR a replica of the task will

For each node n_i *in Population* do
pe_1 PE to which n_i is assigned
t_1= start time of n_i, t_2 = end time of n_1 as found by the scheduling operation
    set AR=0
    For each element $pe_2 \in PE_{Assigned}$ and $pe_2 \neq pe_1$ do
    if pe_2 is free during the time slot t_1 and t_2
        Apply Active Replication (AR) and assigned the replica of n_i to pe_2 & set AR=1
    end loop
    else (if AR==0)
        Apply Rollback Recovery with Checkpointing (RRC) with m number of checkpoints,
        m is randomly chosen between 1 and $cp \times(\alpha + \lambda) < t_{node}$
end loop
$PE_{Assigned}$ is the set of only those PEs to which task(s) are assigned in the assignment operation

Fig. 3. Proposed Technique to Apply Fault Tolerance using AR or RRC

be simultaneously executing on another PE. In that way, when a fault is detected on any task then its output can be obtained from the replica. Both RRC and AR increases the execution time of tasks. The execution time of tasks when fault occurs ($t_{faultcase}$) and execution time when no fault occurs ($t_{nofaultcase}$) can be determined by using the following formulas: When RRC is used: $t_{nofaultcase} = t_{node} + cp \times (\alpha + \lambda), t_{faultcase} = t_{nofaultcase} + t_{slack}$, and $t_{slack} = \frac{t_{node}}{cp} + \mu$. When AR is used: $t_{nofaultcase} = t_{node}, t_{faultcase} = t_{node} + \alpha$. where α refers to the time required to detect the fault and λ refers to the time required in checkpointing at one point. cp refers to the number of checkpoints in the task. t_{node} is the tasks execution time. μ is the recovery overhead. The FT is applied during initialization and reapplied after allocation in the optimization loop. The approach used to apply FT is shown in Fig. 3. FT can provide tolerance from single fault per node, i.e. computation at any node can still proceed to completion even if a fault occurs during its execution. No fault tolerance is provided in replicas' execution. The algorithm proposed to apply RRC and AR to the nodes is shown in Fig. 3.

4 Simulations

This section shows results of the synthesis process using the proposed SimE based algorithm and their comparison with the previous algorithms. The input task graphs, PE, CRs are generated using the research tool TGFF [15]. The TGFF can generate task graphs of different sizes that have hard time deadlines and can resemble real world applications. It can also generate PEs, and CRs that have information on tasks execution times, communication delays, cost, power consumption and parallel processing capability of the PEs. The software development is performed using UML and Java. The UML model defines the structure of the task graphs, PEs, CRs, and structure of the SimE and relationships among them. The SimE functions are implemented as operations in the UML model. The functions in the task graphs computations are included in the task graph structure so that the same task graphs can be used with any algorithm. The fault tolerance parameters are $\alpha = 10\%$, $\mu = 30\%$, and $\lambda = 5\%$ of the task execution time. The platform used is Intel Core i5 running at 2.27 GHz

and with 4 GB of memory. The SimE performance is compared with: (1) PRSA architecture that is described in SLOPES [2]. The PRSA is implemented with parameters: population size =10, number of clusters= 6 and stopping criteria is that the optimization loop is executed for 15 minutes, and (2) Genetic Algorithm (GA) [5], in which the population size is 12 and stopping criteria for the optimization loop is 1.5 minutes. PRSA and GA does not implement fault tolerance (FT). Total 60 (30/30) PEs and CRs are generated in the resource library using TGFF. The comparison results are shown in Table 1. All results include only valid solutions. The first column contains the task graph characteristics i.e., task graph name and number of nodes (N) and edges (E). The second column contains the name of the algorithm to which the remaining values in the same row belongs to, i.e. Proposed algorithm with fault tolerance (Proposed(FT)), proposed algorithm without fault tolerance (Proposed(nonFT)), PRSA, and GA. Third column contains the average execution time taken by one iteration. Fourth and fifth columns contain the minimum hardware cost and minimum power consumption that was obtained from the algorithms. Last column contains the average numbers of nodes that uses active replication (AR) or rollback recovery with checkpointing (RRC) for fault tolerance. The values in the last column are only for the Proposed(FT) algorithm. At the bottom in table 1 the average gain (in terms of number of times) that is obtained using the proposed algorithm over PRSA and GA is shown. The average gain obtained are:(1)Proposed algorithm with FT has an execution time that is on average : 5.19, 1.15 times lesser

Table 1. Simulation & Comparison Results

Task Graph Nodes/Edges	Algorithm	Execution Time (ms)	Min. Cost	Min. Power	FT Scheme
tg0(57/56)	Proposed(FT)/Proposed(nonFT)	1480.54/40.53	1054.96/1034.4	85.91/37.83	AR=48
	PRSA /GA	1514.71/ 519.38	3467.90/ 3171.9	77.3/58.3	RRC=8
tg1(45/48)	Proposed(FT)/Proposed(nonFT)	10.41/8.27	808.06/890.46	65.77/29.43	AR=29
	PRSA/ GA	461.92/ 70.15	2702.8 / 2424.1	60.2/ 44.6	RRC=15
tg2(44/43)	Proposed(FT)/Proposed(nonFT)	31.76/6.9	821.07/852.99	61.01/26.69	AR=37
	PRSA /GA	755.67/ 142.49	2757.30/ 2129.9	56.30/ 42.7	RRC=6
tg3(44/45)	Proposed(FT)/Proposed(nonFT)	14.38/7.84	823.58/763.72	60.690/26.44	AR=36
	PRSA/ GA	599.28/ 118.18	2634.60/ 2042.7	56.30/ 42.7	RRC=7
tg4(73/76)	Proposed(FT)/Proposed(nonFT)	158.21/2.87	1274.33/1401	111.1/50.44	AR=71
	PRSA/ GA	2323.91 758.65	4723.5/ 4069.7	100.3/ 79.4	RRC=1
tg5(60/62)	Proposed(FT)/Proposed(nonFT)	52.06/14.77	1206.66/1161.5	89.79/41.16	AR=43
	PRSA/ GA	752.87/ 244.15	3768.50/ 3214.4	91.20/ 60.5	RRC=16
tg6(48/53)	Proposed(FT)/Proposed(nonFT)	13.16/9.1	945.36/688.52	69.30/31.03	AR=24
	PRSA/ GA	566.52/ 81.46	2995.30/ 2341.7	68.10/ 45.5	RRC=23
tg7(39/42)	Proposed(FT)/Proposed(nonFT)	58.83/4.9	847.05/708.92	57.46/25.40	AR=25
	PRSA/ GA	752.63/ 23.89	2844.30/1898.8	57.70/36.4	RRC=13
tg8(40/43)	Proposed(FT)/Proposed(nonFT)	18.38/5.5	748.09/847.96	58.14/25.71	AR=31
	PRSA/ GA	469.92/ 78.91	2631.60/ 2130.5	53.80/ 39.8	RRC=8
tg9(47/46)	Proposed(FT)/Proposed(nonFT)	22.87/8.2	868.81/860.39	69.71/30.89	AR=40
	PRSA/ GA	1466.67/ 116.22	3047.5/ 2286.1	64.7/ 48.1	RRC=6
Average Gains	Proposed (FT) over PRSA	5.19	3.35	0.94	
	Proposed (FT) over GA	1.15	2.74	0.68	
	Proposed (nonFT) over PRSA	88.76	3.43	1.53	
	Proposed (nonFT) over GA	19.78	2.79	2.25	

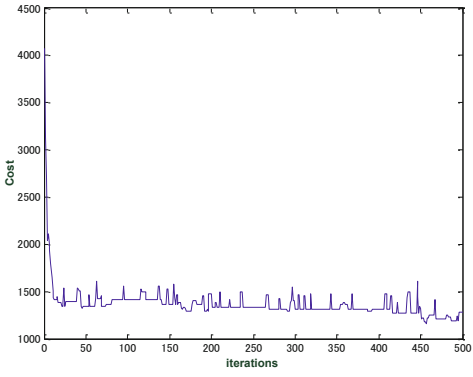

Fig. 4. Optimization behavior of the SimE on task graph tg5

than the PRSA and GA, and can produce hardware that have cost that is 3.35 and 2.74 times lesser than the PRSA and GA. The power consumption of the PRSA and GA are 0.94 and 0.68 times of the propsoed (FT) algorithm. (2) Proposed algorithm without FT has an execution time that is 88.76 and 19.77 times lesser than the PRSA and GA respectively. The hardware cost is 3.43 and 2.79 times and power consumption is 1.53 and 2.11 times lesser than the PRSA and GA respectively. The Fig. 4 plots the hardware costs of the solutions obtained in the VS set (Fig. 1) at the end of the optimization process for the task graph tg5. The x-axis contains the increasing order of the iterations in which valid solution were obtained and y-axis contains the corresponding hardware cost value of the ECU.

5 Conclusion

This paper has shown a new design of SimE that can solve the embedded system synthesis problem. The ECU specific optimization objectives were considered to make sure that it can be useful for the ECU synthesis. The goodnesses of nodes or edges is measured by taking average of the individual goodnesses of different optimizations terms (time, cost, power, and resource utilization). The fault tolerance (FT) from single faults per task is also provided by using a combination of active replication (AR) and rollback recovery with checkpointing (RRC) techniques. The FT scheme first attempts to apply AR without using any new PEs exclusively for FT. If AR cannot be applied to that node then RRC will be applied. The results have shown that the proposed algorithm can explore the search space quickly and apply alternate choices in less amount of time as compared to an architecture of PRSA proposed in an algorithm SLOPES and conventional GA.

References

1. Sangiovanni-Vincentelli, A., Di Natale, M.: Embedded System Design for Automotive Applications. Computer 40(10), 42–51 (2007)
2. Shang, L., Dick, R.P., Jha, N.K.: SLOPES: Hardware-Software Cosynthesis of Low Power Cosynthesis of Low-Power Real-Time Distributed Embedded Systems with Dynamically Reconfigurable FPGAs. IEEE Transactions on Computer Aided Integrated Circuits & Systems 26(3) (2007)
3. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman and Co., New York (1997)
4. Kwok, Y.-K., Ahmad, I.: Dynamic Critical-Path Scheduling: An Effective Technique for Allocating Task Graphs to Multiprocessors. IEEE Transactions on Parallel and Distributed Systems 7(5), 506–521 (1996)
5. Sait, S.M., Youssef, H.: Iterative Computer Algorithms with Applications in Engineering. IEEE Computer Society Press, Los Alamitos (1999)
6. Sait, S.M., El-Barr, A., Al-Saiari, U.S., Sarif, B.A.B.: Digital Circuit Design Through Simulated Evolution (SimE). In: The 2003 Congress on Evolutionary Computation (IEEE CEC 2003), Canberra, Australia, vol. 1, pp. 375–381 (2003)
7. Al-Saiari, U.S.: Digital Circuit Design Through Simulated Evolution, MS Thesis, King Fahd University of Petroleum & Minerals, Dhahran, Saudi Arabia (2003)
8. Mahfoud, S.W., Goldberg, D.E.: Parallel Recombinative Simulated Annealing: A Genetic Algorithm. Parallel Computing 21(1), 1–28 (1995)
9. Kianzad, V., Bhattacharyya, S.S.: CHARMED: A Multiobjective Co-Synthesis Framework for Multi-mode Embedded Systems. In: Proc. 15th IEEE Conference on Application Specific Systems, Architectures and Processors (ASAP 2004), pp. 28–40 (2004)
10. Zitler, E., Laumanns, M., Thiele, L.: SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization. In: Evolutionary Methods for Design, Optimization, and Control, pp. 95–100 (2002)
11. Kuchcinski, K.: Constraints-Driven Scheduling and Resource Assignment. ACM Transactions on Design Automation of Electronic Systems 8(3), 355–383 (2003)
12. Pop, P., Izosimov, V., Eles, P., Peng, Z.: Design Optimization of Time- and Cost-Constrained Fault-Tolerant Embedded Systems with Checkpointing and Replication. IEEE Transactions on VLSI Systems 17(3) (2009)
13. Punnekkat, S., Burns, A., Davis, R.: Analysis of checkpointing for real-time systems. Real-Timr J. 20(1), 83–102 (2001)
14. Kling, R.M., Banerjee, P.: ESP: Placement by Simulated Evolution. IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems 8(3), 245–256 (1989)
15. Dick, R.P., Rhodes, D.L., Wolf, W.: TGFF: task graphs for free. In: Proc. of the Sixth Intl. Workshop on Hardware/Software Codesign (CODES/CASHE 1998), Seattle, WA (1998)