Low Power Hardware-Software Partitioning Algorithm for Heterogeneous Distributed Embedded Systems

Tianyi Ma, Jun Yang, Xinglan Wang

Computer and Information Engineering of College
Harbin University of Commerce
150001, Harbin, China
ma_tian_yi@163.com

Abstract. Hardware-software partitioning is one of the most crucial steps in the design of embedded systems, which is the process of partitioning an embedded system specification into hardware and software modules to meet performance and cost goals. A majority of former work focuses on the problem of meeting timing constraints under minimizing the amount of hardware or minimizing time under hardware area constraints. The trends towards energy-efficient design of distributed embedded systems indicate the need for low power hardware-software partitioning algorithms, which are not enough emphasized so far. In this paper, we design tabu search on a chaotic neural network to solve the low power hardware-software partitioning problem. By introducing chaotic dynamics and utilizing the refractory effects of neurons as the tabu effects, the realized tabu search gets partitioning result with lower energy consumption, when compared with genetic algorithm.

Keywords: Hardware-software co-design, hardware-software partitioning, tabu search, chaotic neural network, low power

1 Introduction and Related Work

Embedded systems have become omnipresent in wide variety of applications, such as telecommunication systems, consumer electronics, and other mass products. Modern embedded systems are often implemented as heterogeneous distributed systems. For completing complex embedded system under rigorous time and cost constraints, the hardware-software co-design of these, mostly, mixed software and hardware systems is an inevitable necessity [1]. One of the most crucial steps in the hardware-software co-design of embedded systems is hardware-software partitioning, that is, deciding which components of the systems should be realized in hardware and which ones in software [2]. Partitioning system specification into hardware and software, system designers have to take into account conflicting requirements on performance, power, cost, chip size, etc., and try to archive an optimal tradeoff. Hardware-software partitioning lies at one of the highest level of abstraction for the design of embedded systems, and drastically impacts the cost and performance of the whole system [3], so a good system partitioning is essential for the overall quality of embedded systems. It is also known that higher level of the design hierarchy where power is tackled, higher is the power reduction possible [4]. So low power hardware-software partitioning will consumedly reduce system power consumption.

Minimizing power consumption of embedded systems is a crucial task of modern embedded systems, but low power hardware-software partitioning algorithms, which are not enough emphasized in the past. However, the recent development of the portable-application market has intensified the interest in system-level design techniques for energy-efficient embedded systems.

Dave et al. proposed the first approach that targeted the reduction of power dissipation throughout the co-synthesis process [5]. They used a constructive algorithm to solve the classical multi-rate distributed system co-synthesis problem. This work was extended to target low power embedded systems, hence low power partitioning is only byproduct of their work.

Dick and Jha [6] reported a multi-objective genetic algorithm based co-synthesis approach. This framework simultaneously partitions and schedules task graphs for embedded systems. Their approach tries to obtain tradeoffs of different objectives, and power consumption is also one of these optimized objectives.

In [7], Henkel introduces a low-power hardware/software partitioning approach for core-based systems. Their approach is based on the idea of mapping clusters of operations/instructions to a core that yields a high utilization rate of the involved resources and thus minimizing power consumption. Their approach is low power hardware-software partitioning of embedded core-based systems, however, our work is low power partitioning problem of distributed embedded systems. And moreover, theirs is based on a fine-grained (instruction/operation-level), whereas, our approach is coarse-grained (task/procedure-level).

Peter et al. [8] proposed a simplified hardware-software partitioning formal model. Software implementation of a task is only associated with a software cost, which may be the running time, and hardware implementation of a task is associated with a hardware cost, which can be for instance area, energy consumption etc. The authors try to abandon details of the partitioning problem so as to solve the partitioning problem of large systems. They first took into account the partitioning problem of cost-constrained systems using integer linear programming, and subsequently dealt with the same problem using genetic algorithm. However the model is too simplified, so it may be difficult to be used for solving actual partitioning problem. In our work, we propose the applied formal model of partitioning problem under enlightening of their model, and use tabu search to look for the energy consumption minimum of hardware-software partitioning problem under system execution time constraint, hardware components' area constraints and software processors' memory constraints. By introducing chaotic dynamics and utilizing the refractory effects of neurons as the tabu effects, we realize the tabu search on a chaotic neural network. It can be seen from results of experiments that through reasonably designing the producing methods of candidate solutions, our algorithm is clearly superior to genetic algorithms.

2 Problem Formalization

2.1 Preliminary Definitions

An embedded system application is specified as a set of communicating tasks, represented by a task graph $G_s(V,EG)$. V is the set of graph vertices where vertex $v_i \in V$, $i \in [1,n]$, is the tasks of the systems that will be partitioned. Each vertex of task graph represents a function or a process, an atomic unit of functionality to be executed. EG is the set of graph edges where each edge $eg_{ij} \in EG$ represents communication between vertex v_i and v_j.

The target architecture on which system tasks can be executed or implemented is captured using an architecture graph $G_a(PE,CL)$, where nodes set PE consists of processing components and edges set CL is composed of communication links. Every component $PE_p \in PE$, $p \in [1,k]$, is processing elements which are probably heterogeneous, like general-purpose processors, ASIPs, FPGAs, and ASICs. For distinguishing software processors from hardware components, we define $PE = PE_s \bigcup PE_h$, $PE_s = \{s_1, s_2, ..., s_a\}$ denotes software processors set, consisting of different types and numbers of general-purpose processors or ASIPs; $PE_h = \{h_1, h_2, ..., h_b\}$ denotes hardware components set, including different types and numbers of FPGAs or ASICs, and $k = a + b$.

An infrastructure of communication links, $CL = \{c_1, c_2, ..., c_w\}$, consisting of buses and point-to-point connections, connects these components.

Each task of the system specification might have multiple implementations, and therefore it can be potentially mapped to several components able to execute this task. In the task graph $G_s(V,EG)$, each vertex $v_i \in V$, $i \in [1,n]$, is assigned to six aspects properties set, $\{\{e_i^{s_m}\}, \{e_i^{h_n}\}, \{t_i^{s_m}\}, \{t_i^{h_n}\}, \{m_i^{s_m}\}, \{a_i^{h_n}\}, i \in [1,n]\}$. $e_i^{s_m}$ and $t_i^{s_m}$ respectively represents the software energy consumption and execution time for a given vertex v_i on a specified processor s_m, $m \in [1,a]$. $e_i^{h_n}$ and $t_i^{h_n}$ respectively represents the hardware energy consumption and execution time for a given vertex v_i on a specified hardware unit h_n, $n \in [1,b]$. Similarly, the software implementation of the function requires memory $m_i^{s_m}$ on the processor s_m and the hardware implementation requires area $a_i^{h_n}$ on the hardware unit h_n.

Each edge $eg_{ij} \in EG$ is associated with two aspects properties set, $\{\{e_{ij}^{c_l}\}, \{t_{ij}^{c_l}\}\}$. $t_{ij}^{c_l}$ denotes the time taken to transfer data through bus c_l, $l \in [1,w]$, if v_i and v_j are mapped to different processing elements which communicate by bus c_l, and $e_{ij}^{c_l}$ represents energy consumption for completing the data transfer. For each possible task partitioning, all these attribute values are given in a technology library. These values are either based on previous design experience or on estimation techniques.

2.2 Hardware-Software Partitioning Model

Hardware-software partitioning P can be defined as: $V : P = \{V_{s_1},...,V_{s_a},V_{h_1},...,V_{h_b}\}$, where $V_{s_1}\bigcup,...,\bigcup V_{s_a}\bigcup V_{h_1}\bigcup,...,\bigcup V_{h_b} = V$ and $V_{s_1}\bigcap,...,\bigcap V_{s_a}\bigcap V_{h_1}\bigcap,...,\bigcap V_{h_b} = \Phi$; the partitioning of graph edges are $EG : P = \{EG_{c_1},...,EG_{c_w}\}$, where $EG_{c_1}\bigcup,...,\bigcup EG_{c_w} \subseteq EG$ and $EG_{c_1}\bigcap,...,\bigcap EG_{c_w} = \Phi$.

The energy consumption model E_P of partition P:

$$E_P = \sum_{s_m \in PE_s} \sum_{v_i \in V_{s_m}} e_i^{s_m} + \sum_{h_n \in PE_h} \sum_{v_i \in V_{h_n}} e_i^{h_n} + \sum_{c_l \in CL} \sum_{eg_{ij} \in EG_{c_l}} e_{ij}^{c_l} \qquad \textbf{(1)}$$

The energy consumption model E_P is presented as the sum of three portions, the energy dissipation of executing tasks partitioned to all software processors, the energy dissipation of executing tasks implemented on all hardware components and that of all communications involved on all buses or point to point links, respectively.

The time dissipation model T_P of partition P may be similarly constituted, so

$$T_P = \sum_{s_m \in PE_s} \sum_{v_i \in V_{s_m}} t_i^{s_m} + \sum_{h_n \in PE_h} \sum_{v_i \in V_{h_n}} t_i^{h_n} + \sum_{c_l \in CL} \sum_{eg_{ij} \in EG_{c_l}} t_{ij}^{c_l} \qquad \textbf{(2)}$$

The memory usage model $M_P^{s_m}$ of processor s_m, $m \in [1,a]$:

$$M_P^{s_m} = \sum_{v_i \in V_{s_m}} m_i^{s_m}, m \in [1,a] \qquad \textbf{(3)}$$

In the specified partition P, $M_P^{s_m}$ is the sum of memory request of all tasks mapped to the processor s_m.

The used hardware area model $A_P^{h_n}$ of hardware unit h_n, $n \in [1,b]$:

$$A_P^{h_n} = \sum_{v_i \in V_{h_n}} a_i^{h_n}, n \in [1,b] \qquad \textbf{(4)}$$

In the specified partition P, $A_P^{h_n}$ is the sum of area usage of all tasks mapped to the hardware component h_n.

2.3 Low Power Hardware-Software Partitioning Problem

Based on the above hardware-software partitioning model, the low power hardware-software partitioning problem can be defined as:

$$Min(E_P), \text{ and } T_P \le T_0,\ M_P^{s_m} \le M_0^{s_m}, m \in [1,a],\ A_P^{h_n} \le A_0^{h_n}, n \in [1,b]$$

T_0 is the time constraint of embedded system application which is taken from the interval $[0, \sum_{v_i \in V} \min(t_i^{s_1}, t_i^{s_2},...,t_i^{s_m})]$, and $M_0^{s_m}$ is memory amount dedicated by processor $s_m, m \in [1,a]$, similarly, $A_0^{h_n}$ is hardware area amount of hardware component $h_n, n \in [1,b]$.

For getting valid and low power hardware-software partitioning result, our hardware-software partitioning heuristic algorithm is guided by the following cost function, which minimizes energy consumption while simultaneously satisfying all cost constraints. It takes the following form:

$$\min f = E_P + \sum_{s_m \in PE_s} P_{s_m} \max(0, M_P^{s_m} - M_0^{s_m}) + \sum_{h_n \in PE_h} P_{h_n} \max(0, A_P^{h_n} - A_0^{h_n}) + P_t \max(0, T_P - T_0) \quad (5)$$

Where $P_{s_m}, m \in [1,a]$, $P_{h_n}, n \in [1,b]$ and P_t are adjusted coefficients, respectively corresponding to every processor memory constraint, every hardware component area constraint, and system execution time constraint.

Theorem 1. The low power hardware-software partitioning problem is *NP-hard.*

The problem is not in NP, since a given solution for the problem cannot be verified in polynomial time to be the minimal power consumption. For specifying the NP-hardness of our problem, we reference the complexity result of problem dealt with in paper [8]. Arato etc., define the hardware-software partitioning problem, which minimizes the system execution time under hardware cost constraint, and they prove that the problem is NP-hard. Comparing with their problem, we define low power hardware-software partitioning problem, which minimizes the system power consumption under system execution time constraint, memory constraint of every processor and area constraint of every hardware component. Thus, the proved *NP-hard* problem in paper [8] is a special case of our problem, and hence, the problem in our definition is also *NP-hard.*

3 Proposed Algorithm

We realize the tabu search on chaotic neural network (TS_CNN) by introducing chaotic dynamics and utilizing the refractory effects of neurons as the tabu effects. In the following, we use symbol $i, i = 1, \cdots, n$ to denote task v_i , $i \in [1,n]$ and use symbol $j, j = 1, \cdots, k$ to denote processing element $PE_j \in PE$, $j \in [1,k]$. In the tabu search, we produce candidate solutions for each current solution by changing the assignments of tasks of specified number and keeping the others unchanged. The different number of tasks to be re-assigned leads to the different size of the candidate solutions. In this paper, we specify that the number of candidate solutions equals to $n \times k$ for any size of tasks to be re-assigned. For the detail, we assume that the size of tasks to be re-assigned equals to $S+1$ ($0 \le S \le n-1$), and then we obtain $n \times k$ candidate solutions for every current solution.

3.1 Realizing tabu search by chaotic neural network

Corresponding to the above $n \times k$ candidate solutions, we construct neural network by creating $n \times k$ neurons to realize the tabu search. Our approach includes both the tabu effect and chaotic dynamics, and it is realized by the following equations with an asynchronous updating:

$$\xi_{ij}(t+1) = \beta \Delta_{ij}(t) \quad (6)$$

$$\eta_{ij}(t+1) = -W\sum_{a=1,a\neq i}^{n}\sum_{b=1,b\neq j}^{k} x_{ab}(t) + W \tag{7}$$

$$\zeta_{ij}^{1}(t+1) = -\alpha\sum_{d=0}^{t} k_r^d \{x_{p_1q_1}(t-d) + z_{p_1q_1}(t-d)\} + \theta \text{ , } \ldots \ldots, \tag{8}$$

$$\zeta_{ij}^{S}(t+1) = -\alpha\sum_{d=0}^{t} k_r^d \{x_{p_Sq_S}(t-d) + z_{p_Sq_S}(t-d)\} + \theta \qquad 0 \le S \le n-1 \tag{9}$$

$$\gamma_{ij}(t+1) = -\alpha\sum_{d=0}^{t} k_r^d \{x_{ij}(t-d) + z_{ij}(t-d)\} + \theta \tag{10}$$

$$x_{ij}(t+1) = g\{\xi_{ij}(t+1) + \eta_{ij}(t+1) + \zeta_{ij}^{1}(t+1) + \zeta_{ij}^{S}(t+1) + \gamma_{ij}(t+1)\} \tag{11}$$

$$g(y) = 1/(1+e^{-y/\varepsilon}) \tag{12}$$

When we produce candidate solutions by only changing the assignment of one task, namely, $S=0$, variables $\zeta_{ij}^{1},\cdots,\zeta_{ij}^{S}$ will be eliminated. Where β is the scaling parameter for the gain effect; K_r is the decay parameter of the tabu effect; α is the scaling parameter of the tabu effect; $\Delta_{ij}(t)$ is the gain of the objective function value and $\Delta_{ij}(t) = f_0(t) - f_{ij}(t)$; $f_0(t)$ is the objective value of the current solution at time *t* and $f_{ij}(t)$ is the value of the candidate solution at time *t* which is produced by assigning task i to component j, p_1 to q_1,… and p_S to q_S; $x_{ij}(t)$ is the output of the $(i,j)^{th}$ neuron at time t, $\xi_{ij}(t)$, $\eta_{ij}(t)$, $\zeta_{ij}^{1}(t)$,…, $\zeta_{ij}^{S}(t)$ and $\gamma_{ij}(t)$ are the internal state of the $(i,j)^{th}$ neuron at time *t* corresponding to the gain effect, the value of mutual inhibitory connections, the tabu effect of the assignment of p_1 to q_1, … that of p_S to q_S and that of i to j, respectively. W is the connection weights, and θ is the positive bias.

If $x_{ij}(t+1) > 0.5$, the $(i,j)^{th}$ neuron fires and the task i is assigned to component j, p_1 to q_1, …, and p_S to q_S, respectively. Because many tasks are required to re-assign to new components in one updating, all these assignments should be memorized as tabu effect to avoid the same assignment to be carried out in the range of tabu list size. Then, the tabu list consists of assignments of (i,j), (p_1,q_1), … and (p_S,q_S). Aiming at actual application, we introduce accumulated variables $z_{ij}(t)$ ($i,i=1,\cdots,n$, $j,j=1,\cdots,k$) corresponding to assignments of i to j, which are executed with firing of other neurons than the $(i,j)^{th}$ neuron even though the corresponding $(i,j)^{th}$ neuron does not fire. And then $z_{p_lq_l}(t)$ should also be prepared for memorizing the assignment of (p_l,q_l) which does not correspond to the label of a firing neuron, $l=1,\cdots,S$. In the case of the new updating iteration *t*+1, all variables $z_{ij}(t+1)$ are reset to 0. For memorizing the assignment of p_l to q_l until next updating of the $(p_l,q_l)^{th}$ neuron, the output of the $(i,j)^{th}$ neuron is added to the accumulated output of the $(p_l,q_l)^{th}$ neuron. This procedure is realized as follows: after the updating of the $(i,j)^{th}$ neuron, if the $(p_l,q_l)^{th}$ neuron is already updated on this iteration *t*, $x_{ij}(t+1)$ is added to $z_{p_lq_l}(t+1)$. Otherwise, $x_{ij}(t+1)$ is added to $z_{p_lq_l}(t)$, $l=1,\cdots,S$.

For numerical calculation, Eq. (8), (9) and (10) can be reduced as follows:

$$\zeta_{ij}^{1}(t+1) = k_r \gamma_{p_1 q_1}(t) - \alpha\{x_{p_1 q_1}(t) + z_{p_1 q_1}(t)\} + R \tag{13}$$

$$\zeta_{ij}^{S}(t+1) = k_r \gamma_{p_S q_S}(t) - \alpha\{x_{p_S q_S}(t) + z_{p_S q_S}(t)\} + R \tag{14}$$

$$\gamma_{ij}(t+1) = k_r \gamma_{ij}(t) - \alpha\{x_{ij}(t) + z_{ij}(t)\} + R \tag{15}$$

Where $R = \theta(1 - k_r)$.

4 Experiment Results

The algorithm has been implemented in C, and tested on a Pentium IV 3.0GHz PC with 256MHz memory. Since TS_CNN is a heuristic rather than an exact algorithm, we had to determine both its performance and the quality of the solutions, empirically. For this purpose, we compared it with genetic algorithm (GA) [8], which is universal for hardware-software partitioning and may be used for solving low power partitioning problem. And then the approach [8] is based on heterogeneous distributed embed systems as same as ours.

For general partitioning problem, there does not exist a widely accepted benchmark. We use a real-world GSM encoder task graph to test our approach, which consists of 53 tasks and 81 edges and is taken from [9].

The mapped target architecture in our experiments consists of one general processor and two ASICs, and these processing elements are connected through a bus, forming a heterogeneous distributed embedded system. The properties of these processing elements are listed in Table 1.

Table 1. Target architecture description.

General Processor	Frequency (KHz)	Memory (byte)
	25000	2575860
HW-Component	Frequency (KHz)	Area (mm^2)
ASIC0	2500	18374
ASIC1	2500	50000
Bus	Frequency (KHz)	Average Power (μw)
	66000	4881.648
	Package Size (b)	Package Overhead (b)
	8	33

For detecting the influence for system power consumption when simultaneously change several tasks mapping schemes for producing candidate solutions of current solutions, we select 9 different values of parameter S, $S=1$, $S=2$, …, $S=9$ for experiments and compare the results. According to the producing way of candidate solutions given in above section, we can obtain 53×3 candidate solutions of each current solution in the every iteration of applying tabu search. For comparing with GA on the same condition, we set the population size of genetic algorithm equal to 159. The crossover possibility and mutation possibility are set to 0.9 and 0.5, respectively.

Through initial extensive experiments, we select a group of tabu search algorithm parameters, which are reasonable and can be combined to produce preferable solutions. The final parameters we select are the following: $k_r = 0.8$, $\alpha = 1$, $\beta = 2$, $\varepsilon = 0.001$, $W = 0.0001$, $R = 0.001$. The system time constraint is taken from the interval $[0, \sum_{v_i \in V} \min(t_i^{S_1}, t_i^{S_2}, ..., t_i^{S_m})]$, we select $0.5 \times \sum_{v_i \in V} \min(t_i^{S_1}, t_i^{S_2}, ..., t_i^{S_m})]$ as the system execution time constraints.

Based on initial experiments, we respectively run TS_CNN 100 times according to different parameter S . For impartially comparing the influence of different parameter S , the above each experiment is run on the basis of same initial values and same iteration size 200. For reviewing the performance of our approach, we also execute genetic algorithm 100 times according to above population pool size and parameters.

We compute the mean of 100 times experiments of every S , for thoroughly comparing the influence of different parameter S for objective function value. Then, we also compute the mean of 100 times random experiments of GA. As a whole, we not only compare the influence of different parameter S for TS_CNN approach, but also we compare TS_CNN with GA for detecting the performance of our approach.

Mean of energy function value for different methods are list in Figure 1. We found that when $S = 6$, that is to say, the number of tasks to be re-assigned for producing candidate solutions equals to 7, the obtained mean of energy function value is the lowest, and that becomes larger with the movement of the parameter S towards the two opposite directions. From Figure 1, it can also be seen that the mean obtained by genetic algorithm is far worse than that by the tabu search with $4 \le S \le 9$, which indicates that tabu search on chaotic neural network can find average lower power consumption solution than GA.

The distributions of energy function value when $S = 6$ are given in Figure 2, accompanying with the power consumption distribution from GA. Distribution curves display the number of solutions whose energy values f belong to the following eight open intervals, respectively, f<6, 6<f<7, 7<f<8, 8<f<9, 9<f<10, 10<f<20, 20<f<30 and f>30. As shown in Figure 2, the distribution of energy values with TS_CNN all concentrates on the former five intervals, and there are no solutions belonging to $f > 10$, whereas, there are 11 solutions of belonging to $f > 10$ with GA. The distribution characteristic illustrates that all the solutions produced by TS_CNN more approach global optimization than GA, which accord with the result obtained from mean curve. Even if in the former part of the figure 2, there are more solutions to fall in the intervals $f < 6$ and $6 < f < 7$ with TS_CNN of $S = 6$ than those of GA, which further proves that we can obtain more solutions of lower energy function values by tabu search with $S = 6$.

The performance enhancement of our approach should owe to the complex dynamics of chaotic neural network, which avoid search to be trapped in undesirable local minima. In addition, the fact that TS_CNN can produce average better solution than GA also illuminates that the tabu effect implemented by refractory effect of neurons is effective.

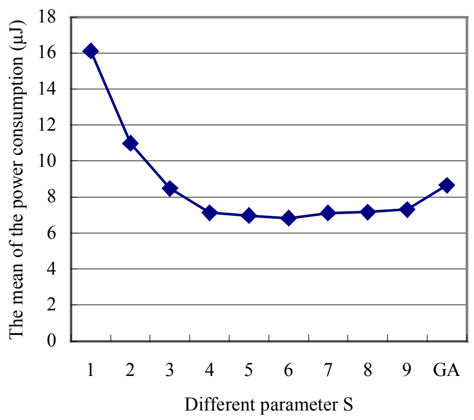

Fig. 1. The mean value of random 100 time experiments of different parameter S

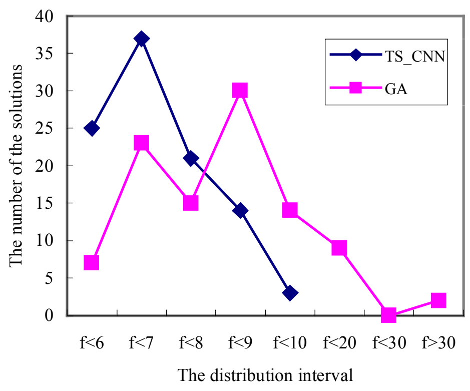

Fig. 2. Solutions distribution comparison of GA and TS_CNN with $S = 6$

5 Conclusions and Future Work

In this paper, we have introduced a new formal model for the low power hardware-software partitioning problem. This model has made it possible to investigate the optimization objective and constraints of the problem formally. Moreover, we have presented a tabu search on a chaotic neural network, which is a novel approach for the low power hardware-software partitioning of heterogeneous distributed embedded systems. In our empirical tests on a task graph of real-world GSM encoder, we find that the algorithm obviously outperforms genetic algorithm by properly producing candidate solutions of current solutions. For the specified example, when the number of tasks to be re-assigned equals to 7, the obtained mean of energy function value is the lowest. We attribute the better low power partitioning result to these facts that our formal model is valid, and the novel idea of designing tabu search on a chaotic neural network is fit to solve the problem.

Our future plans include more tests of the algorithm using other real-world examples and hypothetical examples. And we prepare to extend the algorithm and compare it with other heuristic algorithms, e.g., simulated annealing.

References

1. Wayne, W.: Hardware-software codesign of embedded systems. Proceedings of the IEEE, vol. 82, no. 7 (1994)
2. Arató, P., Mann, Z.A., Orbán, A.: Algorithmic aspects of hardware/software partitioning. ACM Transactions on Design Automation of Electronic Systems, vol. 10, no. 1, (2005) 136-156
3. Mann, Z.A., Orbán, A.: Optimization problems in system-level synthesis. Proceedings of the 3rd Hungarian-Japanese Symposium on Discrete Mathematics and Its Applications, Tokyo (Japan)(2003)
4. Jha, N.K.: Low power system scheduling and synthesis. In: Proc. Int. Conf. Computer-Aided Design, (2001) 259-263
5. Dave, B. P., Lakshminarayana, G., Jha, N. K.: COSYN: Hardware-software co-synthesis of heterogeneous distributed embedded systems. IEEE Trans. On VLSI Systems, vol. 7, 1999(92–104)
6. Dick, R. P., Jha, N. K.: MOGAC: A multiobjective genetic algorithm for the hardware-software co-synthesis of distributed embedded systems. IEEE Trans. Computer-Aided Design, vol. 17 (1998)
7. Henkel, J.: A Low Power Hardware/Software Partitioning Approach for Core-Based Embedded Systems. In: Proceedings of the 36th ACM/IEEE conference on Design automation conference, (1999) 122-127
8. Arato, P., Juhasz, S., Mann, Z.A., Orban, A., Papp, D.: Hardware/software partitioning in embedded system design. In: Proceedings of the IEEE International Symposium on Intelligent Signal Processing (2003)
9. Schmitz, M.T.: Energy Minimisation Techniques for Distributed Embedded Systems. Ph.D.dissertation, University of Southampton University, United Kingdom (2003)