Modelling and Analysis of Power Consumption for Component-based Embedded Software [⋆]

Hu Jun[1,2], Li Xuandong[2], Zheng Guoliang[2], and Wang Chenghua[1]

[1] College of Information Science and Technology
Nanjing University of Aeronautics and Astronautics
Yudao Str. 29, Nanjing, 210016 China
{hujun,chwang}@nuaa.edu.cn
[2] State Key Laboratory of Novel Software Technology
Department of Computer Science and Technology
Nanjing University, Nanjing,210093 China
{lxd,zhenggl}@nju.edu.cn

Abstract. With the increasing complexity of the real-time embedded software, the power consumption is becoming a real challenge in the system designs. In this paper, for modelling the component-based embedded software, the interface automata is extended by adding time intervals on the actions and assigning energy consumption rates on the states. The extensional formalism is called *energy interface automata*. Then the system designs are modelled by energy interface automaton networks which consist of a set of energy interface automata synchronized by shared actions. Based on analyzing the integer state space of the energy interface automaton networks and its compatible reachiability graph, we develop two algorithms for the problem of the minimal energy consumption calculation and the maximal energy consumption verification respectively.

Keywords: embedded software design, power-aware computing, component-based design, real-time system, model checking, interface automata.

1 Introduction

In recent years, as the result of the increasing complexity of real-time embedded systems which also have high performance demands, the embedded software raises many challenges to current software techniques. Especially for the power consumption issues, new methodologies and tools are required to deal with it at all abstraction levels. Currently, many approaches have been proposed to reduce total power consumption, such as: the dynamic voltage scaling (DVS), dynamic power management (DPM), as well as many other studies on the energy-aware schedule schemes, and so on. However, most approaches to energy analysis of the embedded software require the system to be completely developed, depending

[⋆] Supported by the National Natural Science Foundation of China (No.60425204, No.60233020), the National Grand Fundamental Research 973 Program of China (No.2002CB312001), and by Jiangsu Province Research Foundation (No.BK2004080).

on either measurement on a hardware prototype or detailed simulation of the entire system. From the viewpoint of software engineering, typically the traditional process is too late in the design cycle to perform high-level or architectural optimizations. And it is important to model, analyze and check the power consumption characteristics at the abstraction level of the system design, without consideration of too much details of the hardware or platform.

In this paper, considering the component-based embedded software designs we extend the interface automata[1] by adding time intervals on the actions and assigning energy consumption rates on the states to describe the energy-aware behaviors of software components. The extensional formalism is called *energy interface automata (EIA)*. And the system designs are modelled by energy interface automaton networks which consist of a set of energy interface automata synchronized by shared actions. The rest of paper is organized as follows. Section 2 gives the description of the problems we concern and the basic idea of power consumption rates. In section 3, we introduce the interface automata model and the energy interface automata networks, and give the formal definitions for analysis and verification. In section 4 and 5, based on constructing and investigating the compatible reachiability graph of the integer state space of the system design models, we develop algorithms to solve the problems of minimal energy consumption calculation and maximal energy consumption verification respectively. The last section includes the related work and the conclusion.

2 Problem Statement

In this work, we mainly consider the following power consumption problems:

Minimal Energy Consumption Calculation : Given a special state of a component-based embedded software system, how much energy should be consumed at the least, while the system can operate without any mistake from the initial state to that special one?

Maximal Energy Consumption Verification : Given the time constraint, power limitation and the special state of the system, does the total amount of the energy consumption satisfy the given power limitation when the system reaches the special state within the time constraint?

Based on the observation that there are different kinds of power-aware characteristics of hardware in embedded systems (for example, most power supply devices have the busy, idle and sleep modes changing along with the system states to reduce the power consumption), we can make a reasonable assumption that *there are different power consumption rates on different states of the embedded software system, and furthermore, those rates could be any nonnegative real number, not just be several simple modes.*

3 Modelling the System Behaviors

3.1 Interface Automata(IA) Model

Interface automaton[1] is a light-weight formal language to describe the order of the interactions between the software component and its environment. The basic idea is that when the components are composed, their environment assumptions should be also composed. Accordingly, if there are *some* helpful environments satisfy both of their environment assumptions, these components are considered compatible. The formal definitions are given below.

Definition 1. An *interface automaton (IA)* is a tuple $P = (V_P, v_P^{Init}, A_P^I, A_P^O, A_P^H, \mathit{\Gamma}_P)$ where:

- V_P is a finite set of states, each state $v \in V_P$;
- $v_P^{Init} \in V_P$ is the initial state;
- A_P^I, A_P^O and A_P^H are mutually disjoint sets of input, output and internal actions, and $A_P = A_P^I \cup A_P^O \cup A_P^H$ is the set of all actions;
- $\mathit{\Gamma}_P \subseteq V_P \times A_P \times V_P$ is a set of transitions. $\square$

3.2 Energy Interface Automata(EIA) and the EIA-Networks

Essentially the amount of energy consumption has a close relationship with the running time of the system. So we need to extend the interface automata with both of time and energy semantics to describe the system power-aware behaviors. Fig.1 shows an example of an energy interface automata model of component *communication*, which is an extensional energy version of the *IA* model proposed in [1]. On each action there is a time interval indicates the demand of time constraint, and on each state there is a nonnegative real number represents how much energy is consumed in 1 time units, the *power consumption rate*. For example, in state 1, the notation of time interval [2,4] indicates that the system should stay here at least 2 time units, at most 4 time units, and the real number 1.2 in state 0 denotes that the energy consumption is 1.2 in 1 time unit when system stay in there.

Definition 2. An *energy interface automaton(EIA)* is a tuple $P = (V_P,\ F_P,\ A_P,\ I_P, \mathit{\Gamma}_P)$ where:

- V_P is a finite set of states, each state $v \in V_P$; and there is an initial state v_P^{init} at most;
- $F_P : V_P \mapsto R^+$ is a function which maps each state to a nonnegative real number;
- A_P is the set of all actions, each action $a \in A_P$, and it include the A_P^I, A_P^O and A_P^H, the mutually disjoint set of input, output and internal actions respectively;
- I_P is a finite set of time intervals; each interval has the form of $[x, y]$ where x, y are nonnegative integer numbers ($x \leq y$, $y \neq \infty$);
- $\mathit{\Gamma}_P \subseteq V_P \times A_P \times I_P \times V_P$ is a set of transitions.

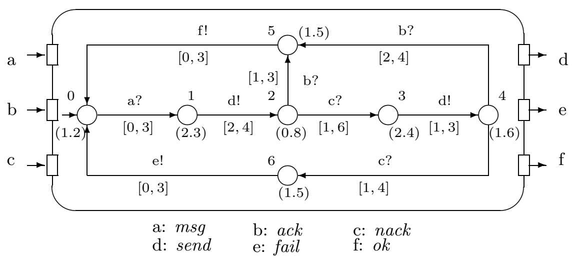


Fig. 1. The EIA model of component *communication*

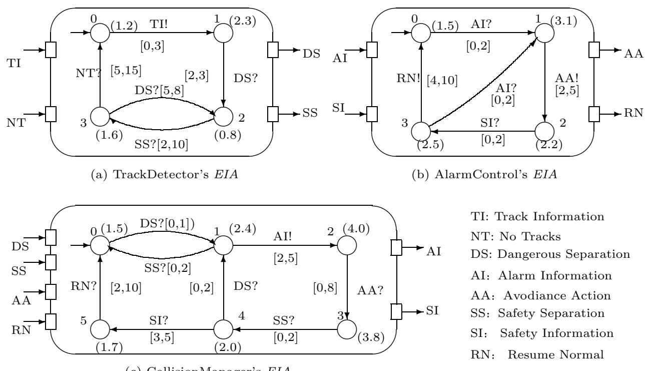
□

We use a *timed energy state sequence* with a form of $\sigma = v_0 \xrightarrow{a_0,\tau_0} v_1 \xrightarrow{a_1,\tau_1} \ldots \xrightarrow{a_{m-1},\tau_{m-1}} v_m \xrightarrow{a_m,\tau_m} v_{m+1}$ to represent a power consumption behavior starting from the initial state. Accordingly there is a power rates sequence: $e_0\,\hat{}\,e_1\,\hat{}\,\ldots\,\hat{}\,e_m\,\hat{}\,e_{m+1}$ such that each $e_i = F_P(v_i)(0 \leq i \leq m+1)$. The semantics of the behavior σ is that the system starts at state v_0, after τ_0 time units it changes to v_1 through the transition ignited by the action a_0 and staying there for τ_1 time units, and the power consumption in state v_0 and v_1 are $e_0 \times \tau_0$ and $e_1 \times \tau_1$ respectively, and so on.

For modelling the component-based embedded software, we use the *energy interface automaton networks (EIA-Networks)* to represent the power-aware behaviors of the compositional system. Since an input action of one interface automaton may coincide with an output action of the other, these two interface automata will synchronize on such shared actions, asynchronously interleaving on other actions. Those synchronized actions between any two interface automata (P_i, P_j) are denoted by $\mathit{shared}(P_i, P_j) = A_{P_i} \cap A_{P_j} = (A^O_{P_i} \cap A^I_{P_j}) \cup (A^I_{P_i} \cap A^O_{P_j})$. More details about interface automata composition can be found in [1]. Fig.2 shows a simple version of TCAS(Traffic Collision Avoidance System) designs.

Definition 3. Let $N = (K, Z, F_N)$ be an *EIA-Networks* where $K = \{P_1, P_2, \ldots, P_n\}$, each $P_i = (V_{P_i}, F_{P_i}, A_{P_i}, I_{P_i}, \mathit{\Gamma}_{P_i})(1 \leq i \leq n)$, and $Z = \{a \in \mathit{shared}(P_i, P_j) \mid 1 \leq i, j \leq n, i \neq j\}$ is a set of all shared actions, n denotes the total number of components in system. The states and actions are defined as below:

– an *untimed energy state* $\overline{v}$ of N is in $V_{P_1} \times V_{P_2} \times \ldots \times V_{P_n}$, that is, $\overline{v} = (v_1, v_2, \ldots, v_n)(v_i \in V_{P_i}, 1 \leq i \leq n)$. The *initial untimed energy state* of N is $\overline{v}^{init}_N = (v^{init}_{P_1}, v^{init}_{P_2}, \ldots, v^{init}_{P_n})$;
– an *energy state* u of N is a pair $u = (\overline{v}, c)$ where $\overline{v} = (v_1, v_2, \ldots, v_n)$ is an untimed state of N and $c : \{v_i | 1 \leq i \leq n\} \mapsto R^+ \cup \{0\}$ is called the *clock function* which maps each v_i to a nonnegative real number that indicates for the P_i how long the system has been staying at v_i. The *initial energy state* of N is $u^{init}_N = (\overline{v}^{init}_N, c^{init})$ where $\overline{v}^{init}_N$ is the initial untimed energy state of



(a) TrackDetector's *EIA* (b) AlarmControl's *EIA* (c) CollisionManager's *EIA*

Fig. 2. An EIA-Networks of the TCAS system

N and $c^{init}(v_{P_i}^{init}) = 0$ for any $i(1 \le i \le n)$. The set of energy states of N is denoted by U_N;

– $F_N : U_N \mapsto \{E_0, E_1, \ldots, E_{|U_N|}\}$ is a power consumption function which maps each state u to a vector $E = \langle e_1, e_2, \ldots, e_n\rangle$, where each $e_i = F_{P_i}(v_i)(1 \le i \le n)$;

– the set of actions of N is $A_N = A_N^I \cup A_N^O \cup A_N^H$, where the set of input actions is $A_N^I = \left(\bigcup_{1\le i\le n} A_{P_i}^I\right)/Z$, the set of output actions is $A_N^O = \left(\bigcup_{1\le i\le n} A_{P_i}^O\right)/Z$, and the set of internal actions is $A_N^H = \left(\bigcup_{1\le i\le n} A_{P_i}^H\right) \cup Z$.

– the set of time intervals of N is $I_N = \bigcup_{1\le i\le n} I_{P_i}$. □

The transitions of N is defined as follows:

Definition 4. Let $N = (K, Z, F_N)$ be an *EIA-Networks* where $K = \{P_1, P_2, \ldots, P_n\}$, each $P_i = (V_{P_i}, F_{P_i}, A_{P_i}, I_{P_i}, \varGamma_{P_i})(1 \le i \le n)$, and $u = (\overline{v}, c)$ and $u' = (\overline{v}', c')$ be its states where $\overline{v} = (v_1, v_2, \ldots, v_n)$ and $\overline{v}' = (v_1', v_2', \ldots, v_n')$, the corresponding power consumption vectors are $E = \langle e_1, e_2, \ldots, e_n\rangle$, $E' = \langle e_1', e_2', \ldots, e_n'\rangle$. Then the system can change from the state u to u' by an transition within a delay d(denoted by $u \xrightarrow{a,d} u'$), if one of the following conditions holds:

– for an action $a \notin Z$, there is a transition $(v_k, a, [x_k, y_k], v_k') \in \varGamma_{P_k}$ $(1 \le k \le n)$ satisfying $x_k \le c(v_k) + d \le y_k$ and $c'(v_k') = 0$, $e_k' = F_{P_k}(v_k')$. At the same time $v_i = v_i'$, $c'(v_i) = c(v_i) + d$ and $e_i' = e_i$ for any i $(i \ne k, 1 \le i \le n)$; or

– for an action $a \in$ *shared*$(P_i, P_j)(1 \le i, j \le n, i \ne j)$, there are transitions $(v_i, a!, [x_i, y_i], v_i') \in \varGamma_{P_i}$ ($a!$ denote a is an output action) and $(v_j, a?, [x_j, y_j], v_j') \in \varGamma_{P_j}$ ($a?$ denote a is an input action) satisfying $x_i \le c(v_i) + d \le y_i$, $x_j \le c(v_j) + d \le y_j$, $c'(v_i') = 0$, $c'(v_j') = 0$ and $e_i' = F_{P_i}(v_i')$, $e_j' = F_{P_j}(v_j')$. At the same time, $v_k = v_k'$, $c'(v_k) = c(v_k) + d$ and $e_k' = e_k$ for any $k(k \ne i, j, 1 \le k \le n)$. □

Then a behavior of N can be denoted as a timed energy state sequence: $\sigma = u_0 \xrightarrow{a_0,d_0} u_1 \xrightarrow{a_1,d_1} \ldots \xrightarrow{a_{m-1},d_{m-1}} u_n \xrightarrow{a_m,d_m} u_{m+1}$ where u_0 is the initial state of N. And for each behavior there is a corresponding vector sequence $E_0\hat{\ }E_1\hat{\ }\ldots\hat{\ }E_{m+1}$, where each $E_i = \langle e_{i1}, e_{i2}, \ldots, e_{in} \rangle$ has the same size of n. We use the $delay(\sigma)$ to denote the total time delay of a behavior, that is, $delay(\sigma) = \sum_{i=0}^{m} d_i$, and the $cost(\sigma)$ to represent the total power consumption of the behavior, that is, $cost(\sigma) = \sum_{i=0}^{m}(d_i \times \sum_{j=1}^{n} e_{ij})$.

4 Constructing the Integer Reachiability Graph

Since the delay d could be any nonnegative real number, it is clear that the state space of *EIA-Networks* N is infinite. We need to deal with the power consumption problems in a finite way. Let $\sigma = u_0 \xrightarrow{a_0,d_0} u_1 \xrightarrow{a_1,d_1} \ldots \xrightarrow{a_{m-1},d_{m-1}} u_m \xrightarrow{a_m,d_m} u_{m+1}$ be a behavior of N. If d_i is an integer for any i $(0 \leq i \leq m)$, then the σ is called an *integer behavior*. It follows that any state $((v_1, v_2, \ldots, v_m), c)$ occurring in an integer behavior satisfies $c(v_j)$ is an integer for any j $(1 \leq j \leq m)$, which is called an *integer state*.

Lemma 1. For each behavior σ of an *EIA-Networks* N, there are integer behaviors σ' and σ'' satisfying $cost(\sigma'') \geq cost(\sigma) \geq cost(\sigma')$. □

Theorem 1. Let N be an *EIA-Networks*, u be any state of N, C be a set of behavior in which each element has the form of $u_0\hat{\ }\ldots\hat{\ }u$, and R be a real number, then for each $\sigma \in C$ satisfying $cost(\sigma) \geq R$ if and only if there is an integer behavior $\sigma' \in C$ satisfying $cost(\sigma') \geq R$. □

In the case of $cost(\sigma) \leq R$, there is a similar result. Therefore, for the problem in this paper we only need to consider the integer state space of an *EIA-Networks*.

Based on theorem 1, we can construct a reachiability graph $G = \{S, L\}$ for an *EIA-Networks* N as follows, where S is a set of nodes and L is a set of edges:

1. for the initial state u_N^{Init} of N, u_N^{Init} is in the set S, which is called *initial node* denoted by s_0;
2. for the other state u, if there is a transition $u \xrightarrow{a,d} u'$, then u' is in the set S, and there is an corresponding edge with a form of $s \xrightarrow{l} s'$, where l is the label on the edge, $l = (a, d)$.

Notice that one automaton may produce an output event that is an input event of another automaton, but it is not accepted by the latter one. It means that there are some contradictions between the environment assumptions of these two automata which have shared actions. When this happens, those corresponding states of N are called *illegal states.* An maximum fixpoint algorithm has been presented to compute the compatible composition of interface automata by removing the set of *illegal(N)* recursively[1]. As a result we can denote by $com(N)$ the *compatible EIA-Networks* which only includes the compatible states of N.

Then we can also get a compatible reachiability graph *com*(G) which only consists of those nodes corresponding to the set of compatible states. If *com*(G) is nonempty, it indicates that there is a helpful environment to make the N running normally. Based on the construction of reachiability graph, we can further conclude that each compatible integer behavior of *EIA-Networks* is corresponding to a path in the *com*(G), and vice versa.

5 Analysis and Verification of System Power Consumption

Let ρ be a path of the compatible reachiability graph *com*(G) with a form of $\rho = s_0 \xrightarrow{l_0} s_1 \xrightarrow{l_1} \ldots \xrightarrow{l_{m-1}} s_m$ in which $l_i = (a_i, d_i)(0 \leq i < m)$. Based on the discussion in the above section, several concepts are introduced as follows:

- **consumption rate vector**: For each node s_i of *com*(G), there is a corresponding vector $E_i = \langle e_{i1}, e_{i2}, \ldots, e_{in} \rangle$, and each $e_{ij}(1 \leq j \leq n)$ indicates the power consumption rate of component j in the compositional state u_i which is corresponding to the node s_i;
- **consumption matrix**: For each path ρ in *com*(G), we can construct a matrix such that $E_\rho = (E_0, E_1, \ldots, E_{m-1})^T_{m \times n}$($m$ denotes the length of the path);
- **time delay vector**: For each path ρ in *com*(G), there is a time delay vector with a form of $D_\rho = \langle d_0, d_1, \ldots, d_{m-1} \rangle$, and the total time delay of the path is $T_\rho = \sum_{i=0}^{m-1} d_i$;
- **consumption of the path**: For each path ρ in *com*(G), the power consumption is $\varepsilon_\rho = \sum_{i=1}^{n} (D_\rho \times E_\rho)_{0i}$, where $D_\rho \times E_\rho$ can be denoted as

$$(d_0, d_1, \ldots, d_m) \times \begin{pmatrix} e_{01} & e_{02} & \cdots & e_{0n} \\ e_{11} & e_{12} & \cdots & e_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ e_{(m-1)1} & e_{(m-1)2} & \cdots & e_{(m-1)n} \end{pmatrix}$$

.

In fact, the size of $D_\rho \times E_\rho$ is $1 \times n$ with a form of $C_\rho = (c_1, c_2, \ldots, c_n)$, and each c_i is exactly corresponding to the ith component's power consumption along with the path ρ.

5.1 The minimal consumption calculation problem

Notice that usually the given system states we concern are the untimed states, but in the *EIA-Networks* each special system state has been divided into several integer timed energy states which have the same untimed state with different time stamps. That is, for a special system state u_j, accordingly there are a set of different timed nodes in *com*(G). We use $\delta(u_j)$ to denote it.

Now given a set of finite node $\delta(u_j)$ in $com(G)$, let $\rho(s_j)$ be the set of path in which each element is a reachable path starting from the initial node s_0 to a timed node s_j. Then the problem can be described again as follows:

Given a set of finite node $\delta(u_j)$ in $com(G)$, which one in the set of path $\rho(\delta(u_j))$ satisfies $\varepsilon = min\{\varepsilon_{\rho'} | \rho' \in \rho(\delta(u_j))\}$?

We call that path the *minimal consumption path.*

Theorem 2. For each node of the compatible reachibility graph $com(G)$, its minimal consumption path is a simple path. $\square$

A simple path in $com(G)$ is a path without any loops. Based on the above theorem, we can develop an algorithm to find out which one is the minimal consumption path. The algorithm is depicted in Fig.3. The variable *min_omega* is used to store the value of minimum energy consumption, and the *min_omega_path* is a set variable used to store the minimal consumption paths. The path in *com(G)* that we have so far searched is stored in the variable *current_path.* For the given state u_j, the algorithm traverse the reachiability graph $com(G)$ by a depth first search manner to check all the simple paths between the initial state s_0 and one certain state in $\delta(u_j)$, through calculating and comparing the energy consumption of each path. At the same time, the detail information about the minimal consumption path will be recorded in the set of *mini_omega_path.* Since there is only a finite number of simple paths in $com(G)$ and the algorithm is based on depth first search framework, the complexity of the algorithm is proportional to number and length of simple paths in $com(G)$.

```
min_omega:=∞ ;ε:=0;   min_omega_path:=∅; current_path:=≺ s_0 ≻;
repeat node:= the last node of current_path;
     if node has no new successive node then delete the last node of current_path;
     else begin node:= a new successive node of node;
            if node∈δ(u_j)
            then begin
                     if the power consumption of the path current_path
                        satisfies ε < min_omega
                     then begin
                              min_omega:=ε;
                              min_omega_path:=≺ ≻;
                              min_omega_path:=current_path∪ ≺ s_j ≻;
                          end
                  end
             else append node to current_path;
           end
 until current_path=≺ ≻
```

Fig. 3. Algorithm for the minimum power consumption calculation

Once we get the minimal consumption path, it's easily to obtain some other power properties of system. Suppose that there is a minimal power consumption path ρ_{min} with a form of $\rho_{min} = s_0 \xrightarrow{l_0} s_1 \xrightarrow{l_1} \ldots \xrightarrow{l_{m-1}} s_m$, then

- *minimal power consumption rate of system*: $H_{min} = \varepsilon_{\rho_{min}} / T_{\rho_{min}}$ which can be provided as a reference for the design of system power supply.
- *peak consumption point of path*: For the path ρ_{min} if there is a s_u satisfies $\Lambda_u = \max\{d_v \cdot \sum_{i=1}^{n} e_{vi} | 0 \leq v < m\}$ then we call it the *peak consumption point* of that path.
- *minimal power consumption rate of components*: From each c_i of $C_{\rho_{min}} = D_{\rho_{min}} \times E_{\rho_{min}}$, the power consumption of each components can be easily obtained. The ith component's minimum power rate is $c_i / T_{\rho_{min}}$.

5.2 The maximum verification problem

In order to solve the maximal energy verification problem, it is necessary to check all paths from the initial node to the given node, comparing the energy consumption of each path to the given power limitation. However, due to the loops in *com*(G), the number of path could be infinite and the length of path could also be infinite. As a result, some expected special states may be unreachable in a finite time and the power consumption is infinite. So we need to consider the semantics of the loops and give some constraints to them: *For each given state, there is a deadline T_D which shows that within the limitation the system should have changed from initial state to that given state, implementing the expected functionality.* Then for a given node of *com*(G) we only consider the finite set of reachable path satisfying the corresponding T_D.

Based on the assumption and the theorem 1, we can develop a verification algorithm which is depicted in Fig.4. The algorithm traverses the *com*(G) in a depth first manner starting from the initial node. We use E_D to denote the predefined amount of energy consumption limitation. The *error_path* to store the illegal paths, which satisfy the deadline but violating the energy consumption limitation. The boolean variable $satisfied$ indicates whether all system behaviors satisfying deadlines without violating the given power consumption. For each new node discovered, the algorithm first check if it is such that the total time delay of the path corresponding to *current_path* less than the T_D. If not, then the algorithm backtracks. If yes, then the algorithm further check whether the new node is the given node s_j. If yes again, then the algorithm continue to compare the energy consumption of the *current_path* with the E_D. If the result of the comparison is false, then we put the path into *error_path* and assign the $satisfied$ with $false$. If the new node is not s_j, then the algorithm adds it into the *current_path* to begin a new search. Theorem 1 and the above assumption give the guarantee that the algorithm can terminate, and the complexity is proportional to nodes in *com*(G).

6 Conclusion

The related work include[2–6]. A prototype is being developed to implement the algorithms proposed in this paper, and will be integrated into the tool set of MCESD (Model checker for Component-based Embedded Software Designs)

```
$satisfied$ := **true**; $error\_path$:=$\emptyset$; $current\_path$:=$\prec s_0 \succ$ ;
repeat $node$:= the last node of $current\_path$;
    if $node$ has no new successive node then delete the last node of $current\_path$;
    else begin $node$:= a new successive node of $node$;
            if the total time delay of $current\_path\hat{}node$ satisfies $T_\rho \leq T_D$
            then begin
                    if $node = s_j$
                    then begin
                            if the power consumption of $current\_path$ satisfies $\varepsilon > E_D$
                            then put $current\_path\hat{}node$ into the set of $error\_path$;
                                    $satisfied$ := **false**;
                         end
                    else append the $node$ to $current\_path$;
                  end
         end
until $current\_path$=$\prec\ \succ$
return satisfied.
```

Fig. 4. Algorithm for the maximum power consumption verification

which is based on our preceding works [7, 8]. For future work, we will consider the other kinds of resource properties in component-based embedded software designs, and do more case studies in practical use.

References

1. L. de Alfaro and T. A. Henzinger. Interface automata. In *Proc. of the Joint 8th European Software Engineering Conference and 9th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (ESEC/FSE 01)*, pp.109-120, Austria, 2001.
2. A. Chakrabarti, L. de Alfaro, T. A. Henzinger, and M. Stoelinga. Resource interfaces. In *Proc. of the EMSOFT 2003*, LNCS 2855, pp.117–133, 2003.
3. L. de Alfaro, T. A. Henzinger, and M. Stoelinga. Timed interfaces. In *Proc. of the EMSOFT 2002*,LNCS 2491, pp. 108–122, 2002
4. I. Lee, A. Philippou, and O. Sokolsky. Process-algebraic modeling and analysis of power-aware real-time systems. In *Journal of Computing and Control Engineering*, 13:180–188, 2002.
5. M. Núñez and I. Rodrígez. Pamr: a process algebra for the management of resources in concurrent systems. In *Proc. of Formal Techniques for Networked and Distributed Systems*, Kluwer, pp. 169–184,2001.
6. T. K. Tan, A. Raghunathan, and N. K. Jha. Energy macromodeling of embedded operating systems. In *ACM Transactions on Embedded Computing Systems*, 4(1):231–254, 2005.
7. J. Hu, X. Yu, Y. Zhang, T. Zhang, X. Li, and G. Zheng. Checking component-based embedded software designs for scenario-based timing specifications. In *Proc. of 2005 IFIP International Conference on Embedded and Ubiquitous Computing* , LNCS 3824, pp. 395–404, 2005.
8. J. Hu, X. Yu, Y. Zhang, T. Zhang, L. Wang, X. Li, and G. Zheng. Scenario-based verification for component-based embedded software designs. In *Proc. of 34th International Conference on Parallel Processing Workshops*, IEEE CS, pp. 240–247,2005.