Bulletin of Networking, Computing, Systems, and Software – www.bncss.org, ISSN 2186–5140
Volume 1, Number 1, pages 21–30, January 2012

Hybrid Automata Theoretic Specification and Verification of CPU-DRP Embedded Systems

Ryo Yanase
Kanazawa University,
Japan
Email: ryanase@csl.ec.t.kanazawa-u.ac.jp

Shota Minami
Kanazawa University,
Japan

Satoshi Yamane
Kanazawa University,
Japan
Email: syamane@is.t.kanazawa-u.ac.jp

***Abstract*—In this paper, we propose formal modeling, specification and verification for CPU-DRP systems based on hybrid automata. First, we specify CPU and environment as real-time systems, and specify DRP as hybrid systems by using hybrid automata. Next, we verify various properties by model checking using HyTech. We have realized verification of parallel composition of CPU, DRP and environment.**

I. Introduction

Embedded systems recently begin to have various functions, but increasing the number of processors causes troubles for miniaturization and saving energy. Therefore, recently, a Dynamically Reconfigurable Processor(DRP) is paid to attention [1]. In DRP, a plural number of exclusive processings is executed in the same board by dynamically changing the circuit configuration [1], [2]. DRP is used as an accelerator of CPU, and some DRP is loosely connected with the CPU [1]. In this paper, DRP is loosely connected with the CPU. We model the embedded system which integrates CPU and DRP cooperatively. Also, we specify the embedded system using hybrid automata, and verify properties by using model checker HyTech [3].

In dynamically reconfigurable embedded systems, the deadline is available in the processing task on CPU, and CPU behaves as a real-time system. In addition, the deadline is available in the co-task on DRP. The number of executing co-tasks on DRP changes dynamically by the task creation and disappearance. Also, the operating frequency of DRP changes dynamically. Therefore, we specify the operation of dynamical reconfiguration using the hybrid automaton with the states of generation and disappearance as a static system. By the above feature, we must verify various properties with real-time and hybrid feature.

In this paper, we model the embedded system, in which CPU and DRP cooperatively behave, and then we specify the embedded system by hybrid automata. Also, we verify various properties by using model checker HyTech [3] as follows:

1) First of all, the model is respectively divided into three parts such as external environment, CPU and DRP.
2) Next, a detailed internal structure of an individual model is specified using hybrid automata. Also, dynamic behaviors of generation and disappearance of the task is specified as a static system by the states of generation and disappearance.
3) Finally, model checker HyTech [3] inputs both hybrid automata, and verifies whether hybrid automata satisfy hybrid, real-time and reactive properties or not.

A. Related Works

a) Specification language: A specification language of dynamic reconfigurable system is either reactive model, real-time model or hybrid model. Also, the style is either process algebra, automaton or Petri Net. A. Deshpande has developed SHIFT [4], and F. Kratz has developed R-Charon [5] based on hybrid automaton. SHIFT [4] and R-Charon [5] have specification power for dynamically changing the structure of the network. However, as they can not describe event trigger behaviors, then they can not describe a dynamically reconfigurable processor. Also, the Φ-calculus is a process algebra based on hybrid reconfigurable modeling language [6]. But the Φ-calculus considers continuous behavior to be a property of an explicit environment instead of being part of other embedded systems as we do. On the other hand, thought J. Teich [7] and K.Onogi [8] have studied modeling method of DRP related to this paper based on discrete event system, their method can not specify DRP, in which the operating frequency of DRP changes dynamically.

b) Verification: Wang Yi and co-workers have proposed the general schedulability checking problem for real-time tasks is a reachability problem for a decidable class of timed automata extended with subtraction [15]. Also, Cimatti and Palopoli have modeled real-time tasks by parametric timed automata [16]. Wang Yi's and Cimatti's work mean real-time properties such as schedulability can be verified by timed automata. In this paper, as we verify both real-time and hybrid properties, we specify DRP using hybrid automata.

c) Architecture: Pellizzoni and Caccamo have developed reconfigurable architecture composed of CPU and reconfigurable area (FPGA) with periodic tasks [17]. Also, H. Nakano and T. Shindo have developed dynamically reconfigurable processor LSI [2]. In this paper, our model is reconfigurable architecture composed of CPU and dynamically reconfigurable processor LSI(DRP). We have already specified reconfigurable system composed of CPU and DRP using hybrid automata. Moreover we have verified schedulability of specification using HyTech [18].

In this paper, we improve specification and verify safety and liveness with hybrid, real-time and reactive feature.

II. Model of CPU, DRP and Environment

We model the embedded system that combines CPU, DRP and environment as shown in Fig.1. The embedded system advances processing by cooperated operations of CPU and DRP. Tasks on CPU are dynamically generated by an external environment. Tasks are executed under the management of CPU-Dispatcher. The task that can be executed at the same time on CPU is one. When it is necessary to process two or more tasks, the allocation of CPU is changed according to priority (preemption). When there is a description to use DRP in a task, CPU-Dispatcher outputs the generation demand of co-task of DRP to DRP-Dispatcher. At this time, initial values such as a necessary substrate area and operating frequency in co-task information are inputted into Co-task. We explain

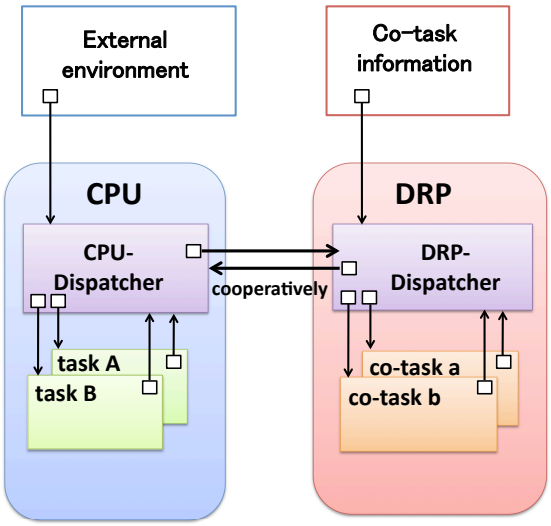

Fig. 1. Overview of dynamic reconfigurable processor

co-tasks on DRP as follows:

1) As shown in the upper part of Fig.2, two or more co-tasks are executable at the same time on DRP. It is arranged on the substrate as long as there is becoming empty in the tile in order of arrival. However, when you execute co-task a and b at the same time, it operates by the slowest value f_b in the operating frequency of co-tasks under execution as shown lower in Fig.2.
2) When the processing of a co-task is completed on DRP, the co-task is disappeared. DRP-Dispatcher informs the completion of processing to CPU-Dispatcher. Afterwards, CPU-Dispatcher restarts the processing of the task, which calls the co-task. If all the processings of the task are completed, the task is disappeared. Even when you process the same co-task, the arrangement of the tile might be different. In this case, the processing time of the co-task is assumed not to change. The embedded system that combines CPU and DRP repeats such operations.

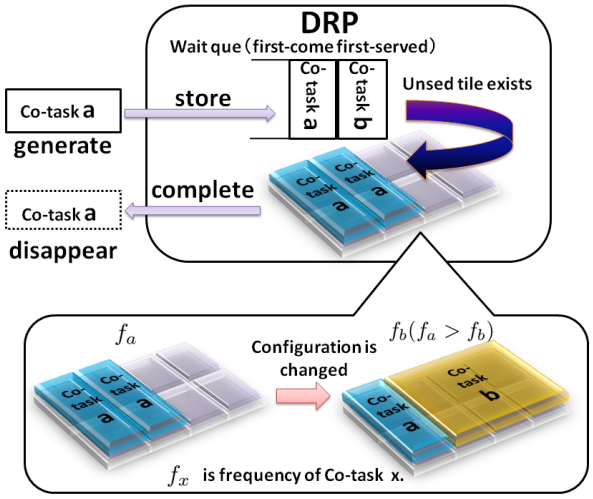

Fig. 2. Behavior of dynamic reconfigurable processor

We specify dynamic generation and disappearance by a static system. The number of tasks and co-tasks must be fixed at specification time. So, as we can specify a dynamically reconfigurable system by a static system, we can verify whether the system is schedulable or not using HyTech. According to dynamic generation and disappearance, task and co-task repeat the following behaviors.

1) Before generation of a task (or co-task), the task (or co-task) exists in a "NONE state".
2) At once when the task (or co-task) is generated, the task (or co-task) goes into a "READY state".
3) Afterwards, at once when the task (or co-task) is executed, the task (or co-task) goes into a "EXEC state".
4) Finally, at once when the task (or co-task) is finished, the task (or co-task) goes into a "NONE state".

Also, the communications between external environment, CPU-Dispatcher, task, DRP-Dispatcher and co-task in Fig.1 are expressed by parallel compositions of hybrid automata.

In general, the operating frequency of CPU uses many hundreds of MHz level, and the operating frequency of DRP is tens of MHz $\sim$ hundreds of MHz. Each input of systems is done by the task designer and LSI designer as shown in Fig.1. A dynamic switch of a configuration can change the configuration with one clock.

III. Specification Language of DRP, CPU and Environment

We define syntax and semantics of a linear hybrid automaton [3] of specification language of DRP, CPU and environment as follows. We extend a linear hybrid automaton [3] with discrete variables.

A. Syntax of a linear hybrid automaton

First, the syntax of a linear hybrid automaton is formally defined.

Definition 1: Syntax of a linear hybrid automaton
An invariable condition and a guard condition are defined as follows:

$$\phi ::= true \mid asap \mid \gamma_1 \sim \gamma_2 \mid \phi_1 \wedge \phi_2$$

, where

$$\gamma ::= x \mid d \mid c \mid \gamma_1 + \gamma_2 \mid \gamma_1 - \gamma_2$$

$\sim \in \{<, >, ==, \leq, \geq\}$, $x \in X$ is a real-valued variable, $d \in D$ is a discrete variable, c is real number. $asap$ is included only in the guard condition. The transition relation to which $asap$ attaches gives priority more than a timed transition in HyTech [3]. Let $B(X)$ be the set of invariable conditions and guard conditions.

A $flow$, which assigns a flow condition to each location, is the following predicate:

$$\alpha ::= \dot{x} = c$$

The dotted variable $\dot{x} \in \dot{X}$ refers to the first derivative of x with respect to time, i.e., dx/dt. Let $F(X)$ be the set of flow conditions. Also, arithmetic expression over a finite set $V (= X \cup D)$ is defined as follows:

$$upd ::= v := const \mid v := v + const$$

, where $const$ is real number, integer,character string. Let $UPD(V)$ be the set of arithmetic expressions.

A linear hybrid automaton LHA is $LHA = (X, D, L, inv, init, flow, E, Act)$ that consists of the following components:

- A finite set X of real-valued variables.
- A finite set D of discrete variables.
- A finite set L of locations.
- A function inv that assigns an invariant condition $\phi \in B(X)$ to each location $l \in L$.
- An initial condition $init$ consists of the set of initial locations and arithmetic expressions.
- A function $flow$ that assigns a flow condition $\alpha \in F(X)$ to each location $l \in L$.
- A finite set Act of actions, where $Act = Act_{in} \cup Act_{out} \cup \{\tau\}$. Here Act_{in} is a finite set of input actions, Act_{out} is a finite set of output actions, τ is an internal action.
- $E \subseteq L \times Act \times B(X) \times 2^{UPD(V)} \times L$ is a finite set called the transition relation. An element of E is a tuple of the form $\langle l, action, \phi, UPD(V), l' \rangle$, where action is either $a!$, $a?$, τ, and $UPD(X)$ is a finite set of arithmetic expressions, and ϕ is a guard condition.

Here a linear hybrid automaton LHA is a stopwatch automaton if a flow condition α is defined as follows:

$$\alpha ::= \dot{x} = 0 \mid \dot{x} = 1$$

■

B. Semantics of a linear hybrid automaton

First, we define a state of a linear hybrid automaton.

Definition 2: State of a linear hybrid automaton
A state of a linear hybrid automaton is a pair (l, μ, ν) consisting of a location $l \in L$, $\nu : X \to \mathbf{R}$, $\mu : D \to \mathbf{Z} \cup \mathbf{STRING}$, where $\mathbf{Z}$ is integer, $\mathbf{STRING}$ is a set of character strings.

■

Transitions of a linear hybrid automaton consist of a timed transition and two discrete transitions.

Next, we define a timed transition of a linear hybrid automaton.

Definition 3: Timed transition

$$(l, \mu, \nu) \xrightarrow{\delta} (l, \mu, \nu')$$

Here a curve of $flow$ is a differentiable function $f : [0, \delta] \to \mathbf{R^n}$, where $|X| = n$, $f(0) = \nu, f(\delta) = \nu'$.

■

Next, discrete transitions consist of an internal transition and a synchronization transition.

Definition 4: An internal transition

$$(l, \mu, \nu) \xrightarrow{\tau, guard, UPD(V)} (l', \mu', \nu')$$

ϕ is assigned to $guard$, variables are updated by $UPD(V)$. Also, if $\phi = asap$, then the discrete transition is immediately done.

■

Definition 5: Synchronization transition
When automaton 1, automaton 2, and automaton 3 change synchronously by action $a?$ and $a!$, the behaviors are formally defined as follows:

$$(l_1, \mu_1, \nu_1) \xrightarrow{a!, guard_1, UPD(V_1)} (l_1', \mu_1', \nu_1'),$$
$$(l_2, \mu_2, \nu_2) \xrightarrow{a?, guard_2, UPD(V_2)} (l_2', \mu_2', \nu_2'),$$
$$(l_3, \mu_3, \nu_3) \xrightarrow{a?, guard_3, UPD(V_3)} (l_3', \mu_3', \nu_3')$$

Automaton 1 outputs $a!$, then both automaton 2 and automaton 3 input $a?$.

■

Finally, we define a run of a linear hybrid automaton.

Definition 6: A run of a linear hybrid automaton

$$(l_0, \mu_0, \nu_0) \xrightarrow{\delta_1} (l_0, \mu_0, \nu_1) \xrightarrow{e_1} (l_1, \mu_1, \nu_2) \cdots$$

, where l_0 is an initial location, both ν_0 and μ_0 are valuations given by an initial condition, $e_1 \in E$ is a transition relation.

■

C. Parallel composition

The communications between external environment, CPU-Dispatcher, task, DRP-Dispatcher and co-task in Fig.1 are expressed by parallel compositions of hybrid automata. For given $LHA_i = (X_i, D_i, L_i, inv_i, init_i, flow_i, E_i, Act_i)$ $(i = 1, \ldots, n)$, the parallel composition $LHA_1 \times \cdots \times LHA_n$ is $LHA = (X, D, L, inv, init, flow, E, Act)$ consisting of the following components:

- A finite set $X = X_1 \cup \cdots \cup X_n$ of variables.
- A finite set $D = D_1 \cup \cdots \cup D_n$ of discrete variables.
- A finite set $L = L_1 \times \cdots \times L_n$ of locations.
- A function that assigns an invariant condition $\phi \in B(X)$ to each location $l \in L$, where $\phi = \phi_1 \wedge \cdots \wedge \phi_n$, $\phi_1 \in B(X_1)$,...,$\phi_n \in B(X_n)$.
- An initial condition is $init = (init_1, \ldots, init_n)$.
- A function that assigns a flow condition $\alpha \in F(X)$ to each location $l \in L$, where $\alpha = \alpha_1 \wedge \cdots \wedge \alpha_n$.
- A finite set $Act = \{\tau\}$ of actions. The input action synchronizes with the output action and it becomes an internal action τ.
- $E \subseteq L \times Act \times B(X) \times 2^{UPD(V)} \times L$ is a finite set called the transition relation. An element of E is a tuple

of the form $\langle l, \tau, \phi, UPD(V), l' \rangle$, where an element of E is a tuple of the form $\langle l, \tau, \phi, UPD(V), l' \rangle$, where $l, l' \in L, \tau \in Act, \phi = \phi_1 \wedge \cdots \wedge \phi_n$, $UPD(V) = UPD(V_1) \cup \cdots \cup UPD(V_n)$.

IV. Configuration of CPU, DRP and Environment

We show configuration of CPU, DRP and environment based on hybrid automata in Fig.3.

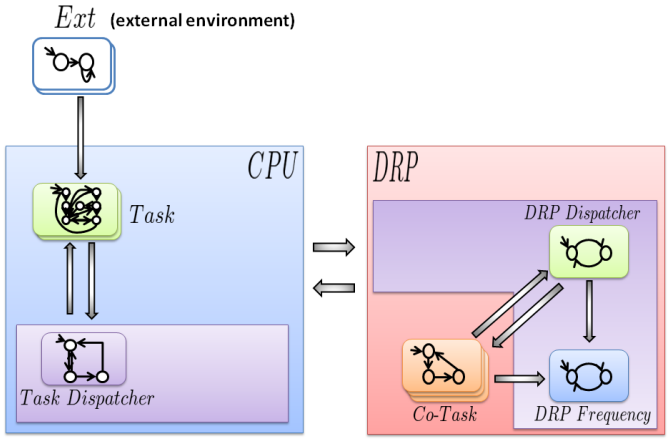

Fig. 3. Configuration of CPU, DRP and environment

1) Ext is an automaton which expresses the environment. Ext sends a start demand on $Task$.
2) $Task$ is an automaton that changes to a ready state from a none state when an activation demand is received from Ext, and a dispatch demand is sent to $Task\ Dispatcher$. It changes to an executing state when selected by $Task\ Dispatcher$ as an execution task. An executing task might send processing activities to a $Co\text{-}task$. In this case, the task changes to a waiting state, the dispatch demand is sent to the $Task\ Dispatcher$ and the processing demand of the $Co\text{-}task$ is sent to the $Co\text{-}task$. It changes to a waiting state when the end response of the processing $Co\text{-}task$ is returned. When the executing task ends processing, the dispatch demand is sent to $Task\ Dispatcher$.
3) $Task\ Dispatcher$ is an automaton for dispatching tasks. When the dispatch demand is received from a task, the task with the highest priority changes to executing state, and other tasks change to waiting states. There is also a dispatch demand from $Task$.
4) $DRP\ Dispatcher$ is an automaton for dispatching $Co\text{-}task$ in DRP. When $DRP\ Dispatcher$ receives dispatch demand from $Co\text{-}Task$, it sends execution demand to $Co\text{-}task$ with the highest priority. if there are tiles for executing a $Co\text{-}task$ of a head of waiting queue.
5) $Co\text{-}Task$ is an automaton of co-tasks executing on DRP. It changes to a ready state when a start demand of $Co\text{-}task$ is received from $task$. It changes to the executing state when the execution demand is received from $DRP\ Dispatcher$, and the processing of $Co\text{-}task$ begins. The processing end response is returned to $task$ when processing ends.
6) $DRP\ Frequency$ is an automaton that manages the frequency of DRP. When a $Co\text{-}task$ is executed with slow operating frequency, the inclination of the execution time of $Co\text{-}task$ under execution is changed.

V. Specification of CPU, DRP and Environment by Hybrid Automata

This section explains a part of specifications by a linear hybrid automata that are described in the previous section. In this paper, we assume that task consists of two tasks such as task A and task B.

A. Ext(external environment):

An external environment automaton that corresponds to Ext in Fig.3 consists of both A_E_TaskA that periodically calls task A and A_E_TaskB that periodically calls task B as shown in Fig.4. The variable "global" represents the clock variable, "Stop" is the time when the verification stops, "x_A" and "x_B" are the clock variables of each task, and "T_A" and "T_B" are periodic times.

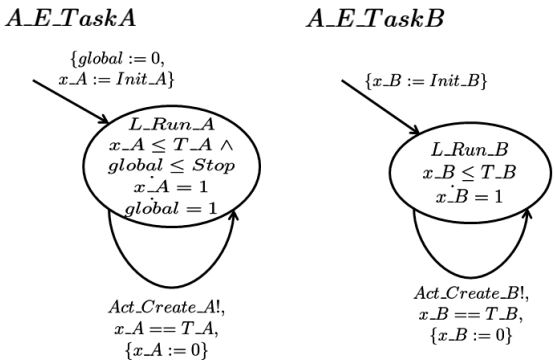

Fig. 4. Ext(environment): A_E_TaskA and A_E_TaskB

B. Task A:

Fig.5 corresponding to Task in Fig.3 is an automaton A_TaskA of task A. Here, e_A is the execution time, and $TaskA\_flag$ denotes a ready state or an executing state, and r_A denotes the elapsed time from release. log_A is a history of the execution location, $E1\_A$ and $E2\_A$ are the CPU processing times required by each execution location. The rough behaviors are as follows:

1) When the action $Act\_Create\_A?$ of the generation request of task A is input from external A_E_TaskA, A_TaskA sets the $TaskA\_flag$ and resets r_A. Then the discrete transition from L_None_A to L_Ready_A occurs. This is the dynamic generation, which is specified by a static system.
2) When the action $Act\_Exec\_A?$ of the execution request of task A is input from A_TD, according to guard

condition $log\_A == Ready\_A$, the discrete transition from L_Ready_A to L_Exec1_A occurs.

3) In location L_Exec1_A, the following two behaviors can be done:
   - When the action $Act\_Exec\_A?$ is input, the discrete transition from L_Exec1_A to L_Exec1_A occurs.
   - When e_A reaches $E1\_A$ by flow condition $e\_A = 1$, invariable condition $e\_A \leq E1\_A$ and guard condition $e\_A == E1\_A$, the discrete transition from L_Exec1_A to L_Wait_A occurs. At this time, processing demand action $Act\_Ready\_a0!$ to co-task a is output to $A\_DRP\_Dispatcher$.
4) When action $Act\_Finish\_a0?$ is input from $A\_Co\_Task\_a0$ in location L_Wait_A, the discrete transition from location L_Wait_A to L_Ready_A occurs. Also, when action $Act\_Finish\_b0?$ is input from $A\_Co\_Task\_a0$ in location L_Wait_A, the discrete transition from location L_Wait_A to L_Fin_A occurs.

C. Task B:

Fig.6 corresponding to Task in Fig.4 is an automaton A_TaskB of task B. Here, e_B is the execution time, and $TaskB\_flag$ denotes a ready state or an executing state, and r_B denotes the elapsed time from release. log_B is a history of the execution location, $E1\_B$ and $E2\_B$ are the CPU processing times required by each execution location. The rough behaviors are as follows:

1) In location L_Ready_B, the following behaviors can be done:
   - When action $Act\_TD\_Ready\_B?$ is input from A_TD, the discrete transition from L_Ready_B to L_Ready_B occurs.
   - When action $Act\_Exec\_B?$ is input from A_TD according to guard condition $log\_B == Ready\_B$, the discrete transition from L_Ready_B to $L\_Pre\_Exec\_B$ occurs.
   - When action $Act\_Exec\_B?$ is input from A_TD according to guard condition $log\_B == Exec\_B$ or $log\_B == Exec\_a\_B$, the discrete transition from L_Ready_B to L_Exec_B occurs.
2) In location $L\_Pre\_Exec\_B$, the following two behaviors can be done:
   - When action $Act\_TD\_Ready\_B?$ is input, the discrete transition from $L\_Pre\_Exec\_B$ to L_Ready_B occurs.
   - When action $Act\_Ready\_a1!$ outputs to $A\_Co\_Task\_a1$ by $wait\_count$, the discrete transition from $L\_Pre\_Exec\_B$ to L_Wait_B occurs.

D. Task Dispatcher:

Fig.7 that corresponds to Task Dispatcher in Fig.3 is an automaton A_TD of Task Dispatcher. Here, variables $TaskA\_flag$ and $TaskB\_flag$ show whether each task executes, waits or runs. $execref$ shows the currently executing task. As flow conditions in all the locations does not exist, the flow condition in the location is not described in this Fig.7.

1) A_TD behaves from an initial location L_Idle_TD. the action $Act\_Create\_A?$, $Act\_Create\_B?$, $Act\_Ready\_a0?$, $Act\_Ready\_a1?$, $Act\_Ready\_b0?$, $Act\_Finish\_A?$, $Act\_Finish\_B?$, $Act\_Finish\_a0?$, $Act\_Finish\_a1?$, or $Act\_Finish\_b0?$ is input. Then the discrete transition from L_Idle_TD to L_Pri3 occurs.
2) In location L_Pri3, the following three behaviors can be done:
   - $execref$ is set to TaskA according to guard condition $asap \wedge TaskA\_flag == 1 \wedge TaskB\_flag == 0$. Then $Act\_Exec\_A!$ is output to A_TaskA, and the discrete transition from L_Pri3 to L_Idle_TD occurs.
   - $execref$ is set to TaskA according to guard condition $asap \wedge TaskA\_flag == 1 \wedge TaskB\_flag == 1$, and action $Act\_Exec\_A!$ is output to A_TaskA. Then the discrete transition from L_Pri3 to L_Pri2 occurs.
   - $execref$ is set to TaskB according to guard condition $asap \wedge TaskA\_flag == 0 \wedge TaskB\_flag == 1$, and the discrete transition from L_Pri3 to L_Pri2 occurs.
3) In location L_Pri2, the following two behaviors can be done:
   - Action $Act\_TD\_Ready\_B!$ is output to A_TaskB according to guard condition $asap \wedge TaskA\_flag == 1$, and the discrete transition from L_Pri2 to L_Idle_TD occurs.
   - Action $Act\_Exec\_B!$ is output to A_TaskB according to guard condition $asap \wedge TaskA\_flag == 0$, and the discrete transition from L_Pri2 to L_Idle_TD occurs.

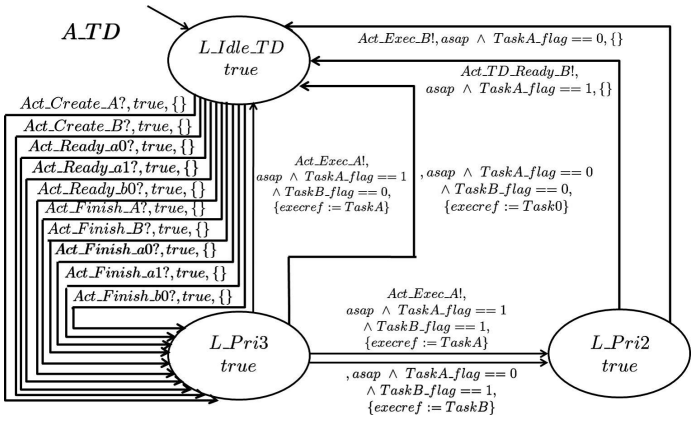

Fig. 7. Task Dispatcher: A_TD

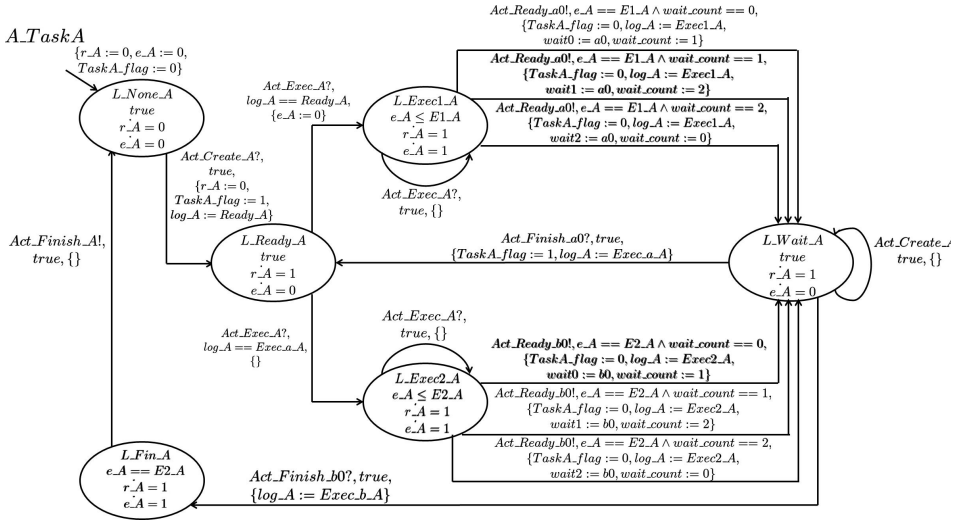

Fig. 5. Task A: A_TaskA

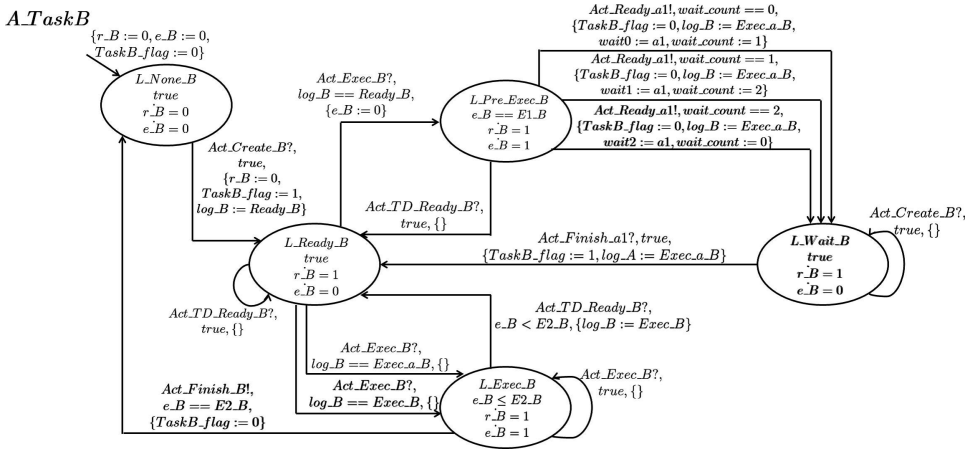

Fig. 6. Task B: A_TaskB

E. DRP Dispatcher

Fig.8 corresponding to $DRP\ Dispatcher$ in Fig.3 is an automaton $A\_DRP\_Dispatcher$. Here a variable $wait\_top$ denotes the top number of the waiting que, a variable $tile$ denotes the number of the space tiles of DRP, variables $Tile\_a$ and $Tile\_b$ show the number of the tiles which are necessary for handling each $Co\text{-}task$. As flow conditions does not exist here, the flow condition in locations is not described in the Fig.8.

1) $A\_DRP\_Dispatcher$ starts from an initial location L_Idle_DD. When either action $Act\_Ready\_a0?$, $Act\_Ready\_a1?$, or $Act\_Ready\_b0?$ is input, the discrete transition from L_Idle_DD to $L\_Mapping$ occurs. When action $Act\_Finish\_a0?$ or $Act\_Finish\_a1?$ or $Act\_Finish\_b0?$ is input from each Co-Task, the discrete transition from L_Idle_DD to $L\_Mapping$ occurs.
2) In location $L\_Mapping$, the following behaviors can be done:
   - According to guard condition $asap \wedge wait\_top == 0 \wedge wait0 == a0 \wedge tile \geq Tile\_a$, $wait\_top$ is increased by 1, and tile is decreased by $Tile\_a$, running demand action $Act\_Exec\_a0!$ is output to $A\_Co\_Task\_a0$, and the discrete transition from $L\_Mapping$ to L_Idle_DD occurs.
   - When a tile of DRP does not have a space, according to guard condition $asap \wedge wait\_top == 0 \wedge wait0 == a0 \wedge tile < Tile\_a$, the discrete transition from $L\_Mapping$ to L_Idle_DD occurs.
   - When a waiting queue does not have a $Co\text{-}task$, by guard condition $asap$, action $Act\_None!$ is output all $Co\text{-}tasks$, and the discrete transition from $L\_Mapping$ to L_Idle_DD occurs.

F. Co-Task

Fig.9 corresponding to $Co\text{-}Task$ in Fig.3 is an automaton $A\_Co\_Task\_a0$. Here a variable r_a0 is a elapsed time from

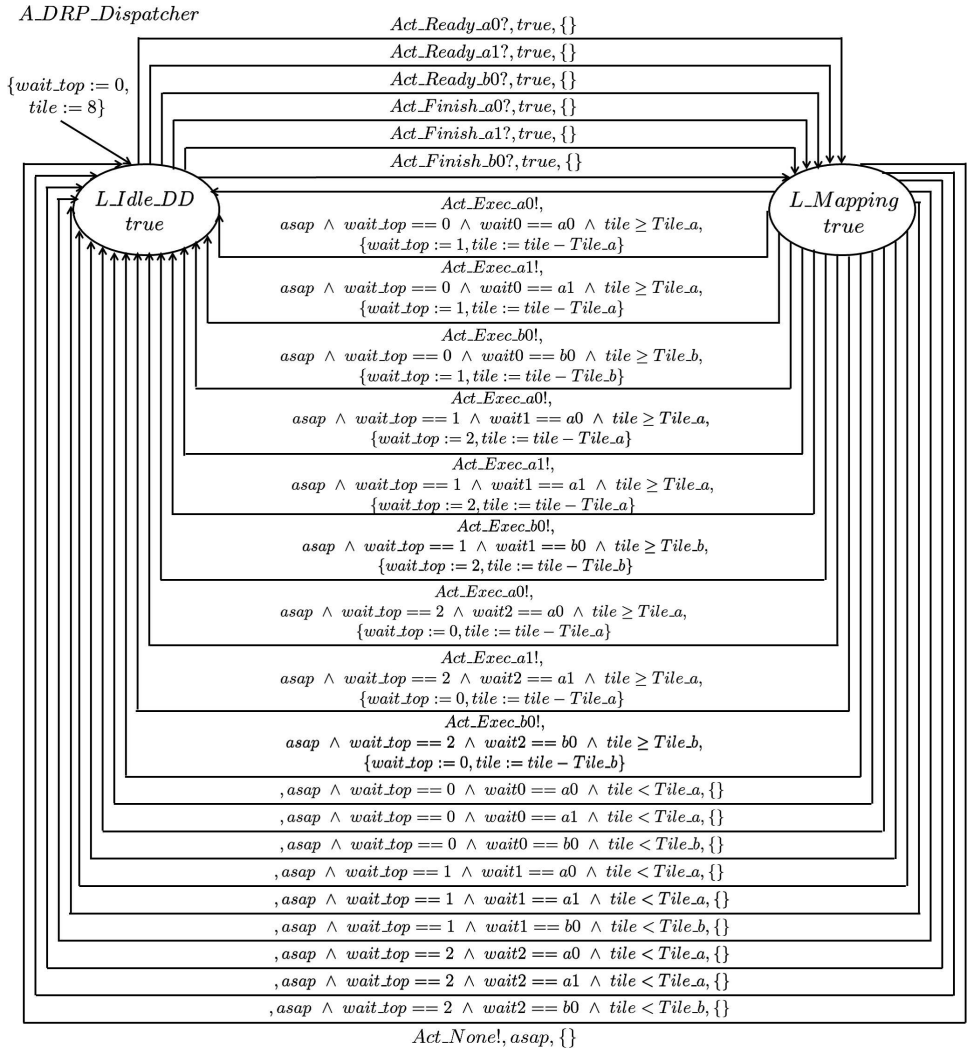

Fig. 8. DRP Dispatcher: $A\_DRP\_Dispatcher$

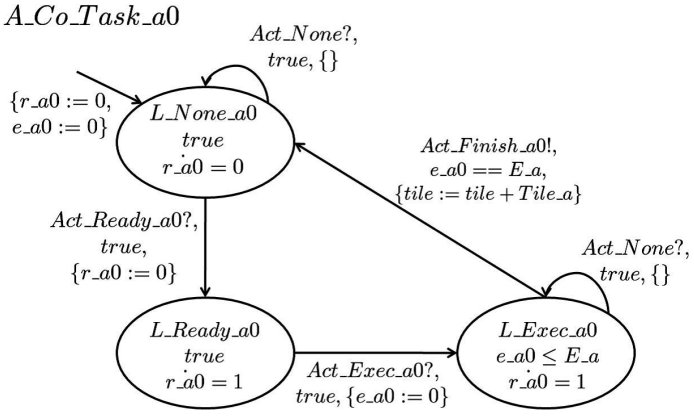

Fig. 9. Co-Task: $A\_Co\_Task\_a0$

start, and e_a0 is a execution time, and E_a is a computation time of $Co\text{-}Task\ a$.

1) $A\_Co\_Task\_a0$ starts from an initial location L_None_a0. If the action $Act\_Ready\_a0?$ is input from $A\_DRP\_Dispatcher$, the discrete transition to L_Ready_a0 occurs after r_a0 is reset. An initial location L_None_a0 means $A\_Co\_Task\_a0$ does not exist, and means $A\_Co\_Task\_a0$ is dynamically generated by the action $Act\_Ready\_a0?$. This is the dynamic generation, which is specified by a static system.
2) If the action $Act\_Exec\_a0?$ is input from $A\_DRP\_Dispatcher$ in the location L_Ready_a0, the discrete transition to L_Exec_a0 occurs after e_a0 is reset.
3) When the discrete transition from L_Exec_a0 to L_None_a0 occurs, the tile of DRP is released by $(tile := tile + Tile\_a)$, and the action $Act\_Finish\_a0!$ is outputted to A_TaskA and $A\_DRP\_Dispatcher$. The discrete transition from L_Exec_a0 to L_None_a0 means $A\_Co\_Task\_a0$ is dynamically disappeared by

$Act\_Finish\_a0!$. This is the dynamic disappearance, which is specified by a static system.

In the case of $A\_Co\_Task\_a1$ and $A\_Co\_Task\_b0$, the same transitions occur, too. We do not describe $A\_Co\_Task\_a1$ and $A\_Co\_Task\_b0$ for the convenience of space.

G. DRP Frequency

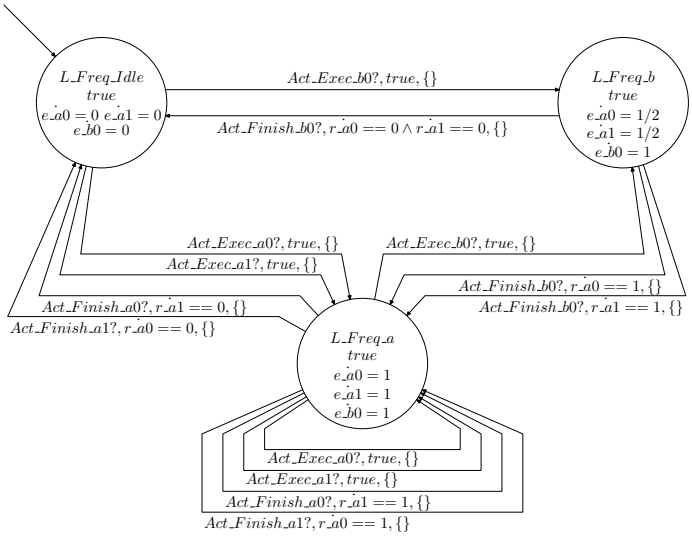

Fig. 10. DRP Frequency: $A\_DRP\_Frequency$

Fig.10 corresponding to $DRP\ Frequency$ in Fig.3 is an automaton $A\_DRP\_Frequency$.

1) $A\_DRP\_Frequency$ starts from the initial location L_Freq_Idle.
2) If the co-task $b0$ is executed, $A\_DRP\_Frequency$ stays in the location L_Freq_b and the operating frequency is fixed the Co-task $b0$'s one.
3) If the co-task $a0$ or $a1$ is executed and the Co-task $b0$ is not executed, $A\_DRP\_Frequency$ stays in the location L_Freq_a and the operating frequency is fixed the operating frequency of the Co-task $a0$ and $a1$.
4) Otherwise $A\_DRP\_Frequency$ stays in the location L_Freq_Idle.

VI. Verifying properties of the system by using HyTech

A. Verification Properties

1) Overview: Dynamically reconfigurable systems have the following three significant features that are called hybrid, real-time and reactive feature [19].

- Hybrid feature——Systems behave as a hybrid system with dynamically changing the operating frequency of DRP
- Real-time feature——Systems behave as a real-time system with the deadline in the processing task
- Reactive feature——Systems behave as a reactive system with responding to the input from the environment.

For our specification, we must verify safety and liveness [20] with the above features. Therefore, we classify the verification properties into six categories shown in TABLE I.

a) Properties with hybrid feature: In this paragraph, we explain liveness and safety conditions with hybrid feature.

Liveness with hybrid feature is a "Idling frequency" condition for the operating frequency of DRP. All executing co-tasks finish and the operating frequency of DRP becomes the idling frequency in the future. As well as liveness, safety with hybrid feature is a "Minimum frequency" condition for the operating frequency of DRP. The operating frequency of DRP is always be fixed on a minimum frequency of running co-tasks.

b) Properties with real-time feature: Next, we explain liveness and safety conditions with real-time feature.

Liveness with real-time feature is a "Dispatching co-tasks" condition for dispatching Co-tasks. If the co-task is called by a task then it is started executing within the maximum waiting time, where the maximum waiting time is the difference between the deadline and the CPU processing time.

Safety with real-time feature is a "CPU schedulability" condition for schedulability of tasks. The remaining time needed to finish processing the task is always less than remaining time until the deadline.

c) Properties with reactive feature: Finally, we explain liveness and safety conditions with reactive feature.

Liveness with reactive feature is a "Destroying co-tasks" condition for destroying Co-tasks. If the co-task is dispatched by the DRP dispatcher then it is destroyed in the future.

Safety with reactive feature is a "Tile control" condition for resource management (i.e., management of tiles). In this paper, maximum number of tiles is 8. Therefore, number of usage tiles always ranges from 0 to 8.

TABLE I
Classification of properties for dynamically reconfigurable systems

	Hybrid feature	Real-time feature	Reactive feature
Liveness	Idling frequency	Dispatching co-tasks	Destroying Co-tasks
Safety	Minimum frequency	CPU schedulability	Tile control

2) Monitor for verifying various properties: In this section, we show monitor automata [12] used to verify properties. The monitor contains special states which are only reachable by violating executions. Besides, the monitor must act strictly as an observer of the original system, without changing its behavior. Reachability analysis is performed on the parallel composition of the system(CPU, DRP and environment) and the monitor. The system correct iff no violating state in the monitor is reached [13]. As the flow condition of the variable that takes the real number value is constant here, the flow condition in each location is omitted in Fig.11–16.

a) Monitor for liveness with hybrid feature: Using the monitor automaton M_{HL} shown in Fig.11, we verify liveness with hybrid feature. M_{HL} checks whether the automaton $A\_DRP\_frequency$ starts from location L_Freq_Idle and reaches this location again.

b) Monitor for safety with hybrid feature: Using the monitor automaton M_{HS} shown in Fig.12, we verify safety with hybrid feature. In this paper, we assume that the operating frequency of co-task b is less than the operating frequency of Co-task a. So, M_{HS} checks whether the flow conditions satisfy $e\_\dot{a} = 1/2, e\_\dot{a}1 = 1/2$ and $e\_\dot{b}0 = 1$ while the co-task $b0$ is executing.

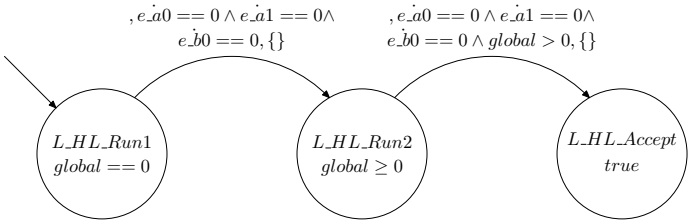

Fig. 11. Monitor M_{HL}

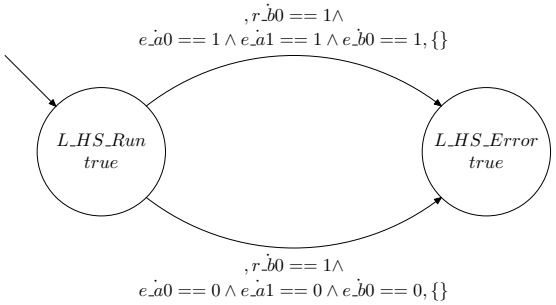

Fig. 12. Monitor M_{HS}

c) Monitor for liveness with real-time feature: Using the monitor automaton M_{RTL_a0} shown in Fig.13, we verify liveness with real-time feature for co-task $a0$. M_{RTL_a0} checks whether r_a0 does not exceed $D\_A - E2\_A$ until the action $Act\_Exec\_a0?$ is input when the action $Act\_Ready\_a0?$ is input, where r_a0 is a elapsed time from start of co-task $a0$, D_A is the deadline of task A and $E2\_A$ is the CPU processing time of task A. Monitor automata can be composed with co-tasks $a1$ and $b0$.

d) Monitor for safety with real-time feature: Using the monitor automaton M_{RTS} shown in Fig.14, we verify safety with real-time feature. M_{RTS} checks whether each remaining time $E2\_A - e\_A, E2\_B - e\_B$ for processing task exceed remaining time $D\_A - r\_A, D\_B - r\_B$ until deadline.

e) Monitor for liveness with reactive feature: Using the monitor automaton M_{RL_a0} shown in Fig.15, we verify liveness with reactive feature for co-task $a0$. M_{RL_a0} checks whether the action $Act\_Finish\_a0?$ is input when the action $Act\_Ready\_a0?$ is input. Monitor automata can be composed with co-tasks $a1$ and $b0$.

f) Monitor for safety with reactive feature: Using the monitor automaton M_{RS} shown in Fig.16, we verify safety with reactive feature. M_{RS} checks whether the number of usage tiles $tile$ satisfies $tile < 0$ or $tile > 8$.

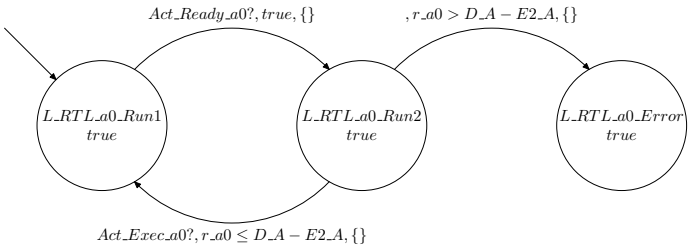

Fig. 13. Monitor M_{RTL_a0}

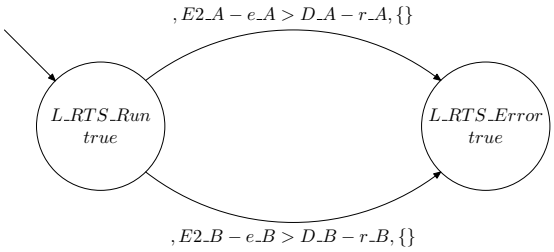

Fig. 14. Monitor M_{RTS}

B. Practical verification experiment

Using above monitor automata, we verify six properties for the dynamically reconfigurable system specified in section IV by HyTech. In this paper, we show only result of verifying safety with real-time due to limitations of space.

First of all, the contents of the processings of tasks are assumed to be TABLE II, and the parameters of co-tasks are assumed to be TABLE III.

We verify safety with real-time in the case of TABLE II–III. A period of task A is T_A, and a period of task B is T_B, where $T\_A = 70$ and $T\_B = 200$ are shown in Fig.4. Pro-

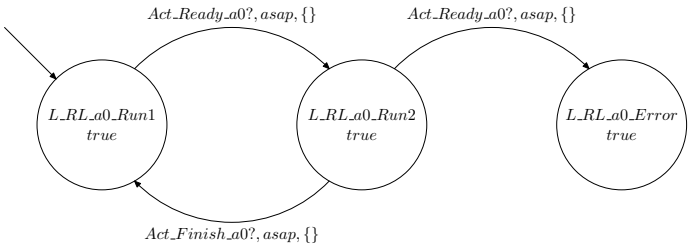

Fig. 15. Monitor M_{RL_a0}

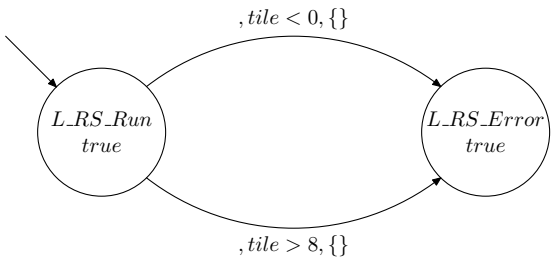

Fig. 16. Monitor M_{RS}

TABLE II
TASK PARAMETER(211MB, 7.1 SECONDS)

Task	Period	Deadline	Priority	Processing procedure
A	70	70	high	20,Co_Task_a,10,Co_Task_b
B	200	200	low	Co_Task_a,110

cessing procedure "20, Co_Task_a,10, Co_Task_b" means that "co-task a is called after CPU advances processing for 20, and co-task b is called after CPU advances processing for 10 afterwards". Also, $E1\_A$ and $E2\_A$ in Fig.5 are $E1\_A = 20$ and $E2\_A = 10$. Moreover, "Co_Task_a,110" means that $E1\_B$ and $E2\_B$ in Fig.6 are $E1\_B = 110$ and $E2\_B = 110$. As computation time of Co-Task a is 10 in TABLE III, E_a

TABLE III
CO-TASK INFORMATION

Co-Task	Computation time	Deadline	Tiles	Ratio of Frequency
Co-Task a	10	15	2	1
Co-Task b	5	10	6	1/2

in Fig.9 is $E\_a = 10$. By verification, it is not possible to schedule the system because the remainder time until the deadline became 29 at time 171 though the time of 30 is needed for CPU processing of task B. In this case, required memory and computation time are 211MB and 7.1 seconds. After the specification is corrected referring to the output error,

TABLE IV
CORRECTED TASK PARAMETER(581MB, 15.8 SECONDS)

Task	Period	Deadline	Priority	Processing procedure
A	70	70	high	20,Co_Task_a,10,Co_Task_b
B	200	200	low	Co_Task_a,97

it is verified again. Corrected task parameter is in TABLE IV. It is possible to schedule the system as a result of correcting the processing time of task B from 110 to 97. In this case, required memory and computation time were 581MB and 15.8 seconds.

VII. CONCLUSION

We model and specify the embedded system such that CPU and DRP cooperatively behave. We specify six verification properties such as safety and liveness with hybrid, real-time and reactive features. Also we verify whether the system is satisfiable or not by parallel composing CPU, DRP and environment. Especially we specify a dynamic reconfigurable processor by a static model. Therefore, we show our proposed model can be verified using an existing model verifier HYTECH [3] by a practicable verification cost. In this points, there are novelty and effectiveness that do not exist in other previous research. In this paper, assuming that co-tasks of DRP are already generated, we model, specify and verify the system. But in fact, co-tasks on DRP are generated and disappeared. For this, during modeling, and specification and verification stages, the state space explosion problems may occur.

In order to avoid the state space explosion problems, we have already developed dynamic hybrid automaton and its dynamic hybrid CEGAR(CounterExample-Guided Abstraction Refinement) [14]. Dynamic hybrid automaton can express the dynamic generation and disappearance of co-tasks. Also, dynamic hybrid CEGAR can avoid the state space explosion problems of formal verifications. We are now implementing dynamic hybrid CEGAR verifier.

REFERENCES

[1] H. Amano, A Survey on Dynamically Reconfigurable Processors, *IEICE Transactions*, 89-B(12), pp.3179-3187, 2006.

[2] H. Nakano, T. Shindo, T. Kazami and M. Motomura, Development of Dynamically Reconfigurable Processor LSI, *NEC Technical Journal*, Vol.56(4), pp.99-102, 2003.

[3] T.A.Henzinger and P.Ho, H.Wong-Toi, HyTech: A Model Checker for Hybrid Systems, *STTT*, Vol.1(1-2), pp.110-122, 1997.

[4] A.Deshp, A.Gollu and L.Semenzato, The SHIFT programming language and run-time system for dynamic networks of hybrid systems, *IEEE Transactions on Automatic Control*, Vol.43(4), pp.584-587, 1998.

[5] F.Kratz, O.Sokolsky, G.J.Papspas and I.Lee, R-Charon, a Modeling Language for Reconfigurable Hybrid Systems, *LNCS 3927*, pp.392-406, 2006.

[6] W.C. Rounds and H. Song, The Phi-Calculus: A Language for Distributed Control of Reconfigurable Embedded Systems, *LNCS 2623*, pp.435-449, 2003.

[7] J.Teich and M.Koster, (Self-)reconfigurable finite state machines: Theory and implementation, *Proc. of Design, automation and test in Europe*, pp.559–568, 2002.

[8] K.Onogi and T.Ushio, Scheduling of Periodic Tasks on a Dynamically Reconfigurable Device Using Timed Discrete Event Systems, *IEICE Transactions*, E89-A(11), pp.3227–3234, 2006.

[9] J. Goossens and P. Richard, Overview of real-time scheduling problems, *Proceedings of ninth international workshop on project management and Scheduling*, pp.13-22, 2004.

[10] P.Marwedel, Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems,2nd edition, Springer, 2010.

[11] S.Takinai and S.Yamane, Case study of Modeling, Specification and Finite Model Checking for Preemptive Embedded Software, *IEICE Transactions*, E93-D(11), pp.2403–2415, 2010. (in Japanese)

[12] M.Y. Vardi and P. Wolper, An automata-theoretic approach to automatic program verification, *Proceedings of Logic in Computer Science*, pp.322-331. IEEE Computer Society Press, 1986.

[13] T.A. Henzinger, Pei-Hsin Ho and Howard Wong-Toi, A user guide to HyTech, *LNCS 1019*, Springer, pp. 41-71, 1995.

[14] M.Sakai and S.Yamane, Dynamic hybrid hybrid automaton and Dynamic hybrid CEGAR, *RIMS Kokyuroku*, RIMS(Kyoto University) 1744, pp.15-24, 2011(in press). (in Japanese)

[15] E. Fersman, P. Pettersson, and W. Yi, Timed automata with asynchronous processes: Schedulability and decidability, *LNCS 2280*, pp.67-82, 2002.

[16] A. Cimatti, L. Palopoli and Y. Ramadian, Symbolic Computation of Schedulability Regions Using Parametric Timed Automata, *IEEE Real-Time Systems Symposium*, pp.80-89, 2008.

[17] R. Pellizzoni and M. Caccamo, Hybrid Hardware-Software Architecture for Reconfigurable Real-Time Systems, *Proceedings of the 14th IEEE RTAS*, pp.273-284, 2008

[18] S. Minami, S. Takinai, S. Sekoguchi, Y. Nakai and S. Yamane, Modeling, Specification and Model checking of dynamically reconfigurable processors, *Computer Software 28(1)*, pp.190-216, 2011.

[19] Z. Manna and A. Pnueli, Models for Reactivity, *Acta Inf. 30(7)*, pp.609-678, 1993.

[20] E. M. Clarke, Jr., O. Grumberg and D. A. Peled, Model Checking, The MIT Press, 1999.