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

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

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

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 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 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.

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 ~ 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 invariant condition and a guard condition are defined as follows:
\[ \phi ::= \text{true} \mid \text{asap} \mid \gamma_1 \sim \gamma_2 \mid \phi_1 \land \phi_2 \]
where
- \( \gamma ::= x \mid d \mid c \mid \gamma_1 + \gamma_2 \mid \gamma_1 - \gamma_2 \)
- \( \gamma \in \{ <, >, =, \leq, \geq \} \)
- \( x \in X \) is a real-valued variable,
- \( d \in D \) is a discrete variable,
- \( c \) is real number.
\( \text{asap} \) is included only in the guard condition.
The transition relation to which \( \text{asap} \) attaches gives priority more than a timed transition in HyTech [3]. Let \( B(X) \) be the set of invariant 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 \( \dot{x} \in X \) 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:
\[ \text{upd ::= } v ::= \text{const} \mid v := v + \text{const} \]
where \( \text{const} \) is real number, integer, character string. Let \( UPD(V) \) be the set of arithmetic expressions.

A linear hybrid automaton \( LHA \) is \( LHA = (X, D, L, \text{inv}, \text{init}, \text{flow}, E, \text{Act}) \) 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 \( \text{inv} \) that assigns an invariant condition \( \phi \in B(X) \) to each location \( l \in L \).
- An initial condition \( \text{init} \) consists of the set of initial locations and arithmetic expressions.
- A function \( \text{flow} \) that assigns a flow condition \( \alpha \in F(X) \) to each location \( l \in L \).
- A finite set \( \text{Act} \) of actions, where \( \text{Act} = \text{Act}_{\text{in}} \cup \text{Act}_{\text{out}} \cup \{ \cdot \} \).
- A function \( \text{inv} \) assigns an invariant condition \( \phi \in B(X) \) to each location \( l \in L \).
- An element of \( E \) is a tuple of the form \( (l, \text{action}, \phi, UPD(V), l') \), where \( \text{action} \) is either \( \text{in} \), \( \text{out} \), \( \tau \), and \( UPD(V) \) is a finite set of arithmetic expressions, and \( \phi \) is a guard condition.

Here a linear hybrid automaton \( LHA \) is a stopwatch automaton if a flow condition \( \alpha \) 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: A state of a linear hybrid automaton
A state of a linear hybrid automaton is a pair \((l, \mu, \nu)\) consisting of a location \( l \in L \), \( \nu : X \to R \), \( \mu : D \to Z \cup \text{STRING} \), where \( Z \) is integer, \( \text{STRING} \) 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 \( f : [0, \delta] \to R^n \), 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
\[ (l, \mu, \nu) \xrightarrow{\tau, \text{guard, UPD}(V)} (l', \mu', \nu') \]
\( \phi \) is assigned to \( \text{guard} \), variables are updated by \( \text{UPD}(V) \). Also, if \( \phi = \text{asap} \), 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:
\[ (l_1, \mu_1, \nu_1) \xrightarrow{a!, \text{guard} \_1, \text{UPD} \_1} (l_1', \mu_1', \nu_1'), \]
\[ (l_2, \mu_2, \nu_2) \xrightarrow{a\_2, \text{guard} \_2, \text{UPD} \_2} (l_2', \mu_2', \nu_2'), \]
\[ (l_3, \mu_3, \nu_3) \xrightarrow{a\_3, \text{guard} \_3, \text{UPD} \_3} (l_3', \mu_3', \nu_3') \]

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
\[ (l_0, \mu_0, \nu_0) \xrightarrow{e_1 \cdots} (l_n, \mu_n, \nu_n) \]
where \( l_0 \) is an initial location, both \( \nu_0 \) and \( \mu_0 \) 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 \( LHA_1 = (X_1, D_1, L_1, \text{inv}, \text{init}, \text{flow}, E_1, \text{Act}_1) \) (\( i = 1, \ldots , n \)), the parallel composition \( LHA_1 \times \cdots \times LHA_n \) is \( LHA = (X, D, L, \text{inv}, \text{init}, \text{flow}, E, \text{Act}) \) 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 \( \phi = \phi_1 \land \cdots \land \phi_n \), \( \phi_1 \in B(X_1) \), \( \cdots \), \( \phi_n \in B(X_n) \).
- An initial condition is \( \text{init} = \text{init}_1, \ldots , \text{init}_n \).
- A function that assigns a flow condition \( \alpha \in F(X) \) to each location \( l \), where \( \alpha = \alpha_1 \land \cdots \land \alpha_n \).
- A finite set \( \text{Act} = \{ \tau \} \) of actions. The input action synchronizes with the output action and it becomes an internal action \( \tau \).
- \( E \subseteq L \times X \times B(X) \times 2^{UPD(V)} \times L \) is a finite set called the transition relation. An element of \( E \) is a tuple
of the form \( \langle l, \tau, \phi, \text{UPD}(V), l' \rangle \), where an element of \( E \) is a tuple of the form \( \langle l, \tau, \phi, \text{UPD}(V), l' \rangle \), where \( l, l' \in L, \tau \in \text{Act}, \phi = \phi_1 \land \cdots \land \phi_n \), \( \text{UPD}(V) = \text{UPD}(V_1) \cup \cdots \cup \text{UPD}(V_n) \).

IV. CONFIGURATION OF CPU, DRP AND ENVIRONMENT

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

![Configuration of CPU, DRP and environment](image)

Fig. 3. Configuration of CPU, DRP and environment

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

An external environment automaton that corresponds to \( \text{Ext} \) in Fig.3 consists of both \( \text{A}_E \text{Task}_A \) that periodically calls task A and \( \text{A}_E \text{Task}_B \) 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.

![Specifications of CPU, DRP and environment by hybrid automata](image)

Fig. 4. \( \text{Ext(external environment): A}_E \text{Task}_A \) and \( \text{A}_E \text{Task}_B \)

B. Task A:

Fig.5 corresponding to Task in Fig.3 is an automaton \( \text{A}_\text{Task}_A \) of task A. Here, \( e_A \) is the execution time, and \( \text{Task}_A \text{flag} \) denotes a ready state or an executing state, and \( r_A \) denotes the elapsed time from release. \( \text{log}_A \) is a history of the execution location, \( E_1 \) and \( E_2 \) are the CPU processing times required by each execution location.

The rough behaviors are as follows:

1) When the action \( \text{Act}_\text{Create}_A \) of the generation request of task A is input from external \( \text{A}_E \text{Task}_A \), \( \text{A}_\text{Task}_A \) sets the \( \text{Task}_A \text{flag} \) and resets \( r_A \). Then the discrete transition from \( L_\text{None}_A \) to \( L_\text{Ready}_A \) occurs. This is the dynamic generation, which is specified by a static system.
2) When the action \( \text{Act}_\text{Exec}_A \) of the execution request of task A is input from \( A_{TD} \), according to guard
condition $\log_A == \text{Ready}_A$, the discrete transition from $L_{\text{Ready}}$ to $L_{\text{Exec1}}$ occurs.

3) In location $L_{\text{Exec1}}$, the following two behaviors can be done:
   - When the action $\text{Act\_Exec}_A$ is input, the discrete transition from $L_{\text{Exec1}}$ to $L_{\text{Exec1}}$ 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 $L_{\text{Exec1}}$ to $L_{\text{Wait}}$ occurs. At this time, processing demand action $\text{Act\_Ready}_a0$ to co-task a is output to $A_{\text{DRP\_Dispatcher}}$.

4) When action $\text{Act\_Finish}_a0$ is input from $A_{\text{Co\_Task}}a0$ in location $L_{\text{Wait}}$, the discrete transition from location $L_{\text{Wait}}$ to $L_{\text{Ready}}$ occurs. Also, when action $\text{Act\_Finish}_b0$ is input from $A_{\text{Co\_Task}}a0$ in location $L_{\text{Wait}}$, the discrete transition from location $L_{\text{Wait}}$ to $L_{\text{Fin}}$ occurs.

C. Task B:

Fig.6 corresponding to Task in Fig.4 is an automaton $A_{\text{Task}}B$ of task B. Here, $e_B$ is the execution time, and $\text{Task}_B\_\text{flag}$ denotes a ready state or an executing state, and $r_B$ denotes 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 $L_{\text{Ready}}\_B$, the following behaviors can be done:
   - When action $\text{Act\_TD\_Ready}_B$ is input from $A_{\text{TD}}$, the discrete transition from $L_{\text{Ready}}\_B$ to $L_{\text{Ready}}\_B$ occurs.
   - When action $\text{Act\_Exec}_B$ is input from $A_{\text{TD}}$ according to guard condition $\log_B == \text{Ready}_B$, the discrete transition from $L_{\text{Ready}}\_B$ to $L_{\text{Pre\_Exec}}\_B$ occurs.
   - When action $\text{Act\_Exec}_B$ is input from $A_{\text{TD}}$ according to guard condition $\log_B == \text{Exec}_B$ or $\log_B == \text{Exec\_a}_B$, the discrete transition from $L_{\text{Ready}}\_B$ to $L_{\text{Exec}}\_B$ occurs.

2) In location $L_{\text{Pre\_Exec}}\_B$, the following two behaviors can be done:
   - When action $\text{Act\_TD\_Ready}_B$ is input, the discrete transition from $L_{\text{Pre\_Exec}}\_B$ to $L_{\text{Ready}}\_B$ occurs.
   - When action $\text{Act\_Ready}_a1$ outputs to $A_{\text{Co\_Task}}a1$ by $\text{wait\_count}$, the discrete transition from $L_{\text{Pre\_Exec}}\_B$ to $L_{\text{Wait}}\_B$ occurs.

D. Task Dispatcher:

Fig.7 that corresponds to Task Dispatcher in Fig.3 is an automaton $A_{\text{TD}}$ of Task Dispatcher. Here, variables $\text{Task}_A\_\text{flag}$ and $\text{Task}_B\_\text{flag}$ show whether each task executes, waits or runs. $\text{execref}$ 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) $A_{\text{TD}}$ behaves from an initial location $L_{\text{Idle}}\_\text{TD}$, the action $\text{Act\_Create}_A$, $\text{Act\_Create}_B$, $\text{Act\_Ready}_a0$, $\text{Act\_Ready}_a1$, $\text{Act\_Ready}_b0$, $\text{Act\_Finish}_A$, $\text{Act\_Finish}_B$, $\text{Act\_Finish}_a0$, $\text{Act\_Finish}_a1$, or $\text{Act\_Finish}_b0$ is input. Then the discrete transition from $L_{\text{Idle}}\_\text{TD}$ to $L_{\text{Pri3}}$ occurs.

2) In location $L_{\text{Pri3}}$, the following three behaviors can be done:
   - $\text{execref}$ is set to $A_{\text{Task}}A$ according to guard condition $\text{asap}\land \text{Task}_A\_\text{flag} == 1 \land \text{Task}_B\_\text{flag} == 0$. Then $\text{Act\_Exec}_A$ is output to $A_{\text{Task}}A$, and the discrete transition from $L_{\text{Pri3}}$ to $L_{\text{Idle}}\_\text{TD}$ occurs.
   - $\text{execref}$ is set to $A_{\text{Task}}A$ according to guard condition $\text{asap}\land \text{Task}_A\_\text{flag} == 0 \land \text{Task}_B\_\text{flag} == 1$, and action $\text{Act\_Exec}_A$ is output to $A_{\text{Task}}A$. Then the discrete transition from $L_{\text{Pri3}}$ to $L_{\text{Pri2}}$ occurs.
   - $\text{execref}$ is set to $A_{\text{Task}}B$ according to guard condition $\text{asap}\land \text{Task}_A\_\text{flag} == 0 \land \text{Task}_B\_\text{flag} == 1$, and the discrete transition from $L_{\text{Pri3}}$ to $L_{\text{Pri2}}$ occurs.

3) In location $L_{\text{Pri2}}$, the following two behaviors can be done:
   - Action $\text{Act\_TD\_Ready}_B$ is output to $A_{\text{Task}}B$ according to guard condition $\text{asap}\land \text{Task}_A\_\text{flag} == 1$, and the discrete transition from $L_{\text{Pri2}}$ to $L_{\text{Idle}}\_\text{TD}$ occurs.
   - Action $\text{Act\_Exec}_B$ is output to $A_{\text{Task}}B$ according to guard condition $\text{asap}\land \text{Task}_A\_\text{flag} == 0$, and the discrete transition from $L_{\text{Pri2}}$ to $L_{\text{Idle}}\_\text{TD}$ occurs.
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 queue, 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-task. As flow conditions do 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 \land wait_top \leq 0 \land wait_0 \leq 0 \land tile > 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 \land wait_top \leq 0 \land wait_0 \leq 0 \land tile < Tile_a, the discrete transition from L_Mapping to L_Idle_DD occurs.
   - When a waiting queue does not have a Co-Task, by guard condition asap, action Act_None! is output all Co-tasks, and the discrete transition from L_Mapping to L_Idle_DD occurs.

F. Co-Task

Fig. 9 corresponding to Co-Task in Fig. 3 is an automaton A_Co_Task_a0. Here a variable r_a0 is a elapsed time from
A. Drp Dispatcher

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 \texttt{A\_Co\_Task\_a1} and \texttt{A\_Co\_Task\_b0}, the same transitions occur, too. We do not describe \texttt{A\_Co\_Task\_a1} and \texttt{A\_Co\_Task\_b0} for the convenience of space.

G. DRP Frequency

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{Fig10.png}
\caption{DRP Frequency: A\_DRP\_Frequency}
\end{figure}

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

1) A\_DRP\_Frequency starts from the initial location \texttt{L\_Freq\_Idle}.

2) If the co-task \texttt{b0} is executed, A\_DRP\_Frequency stays in the location \texttt{L\_Freq\_b} and the operating frequency is fixed the Co-task \texttt{b0}'s one.

3) If the co-task \texttt{a0} or \texttt{a1} is executed and the Co-task \texttt{b0} is not executed, A\_DRP\_Frequency stays in the location \texttt{L\_Freq\_a} and the operating frequency is fixed the operating frequency of the Co-task \texttt{a0} and \texttt{a1}.

4) Otherwise A\_DRP\_Frequency stays in the location \texttt{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.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|}
\hline
Property & Hybrid feature & Real-time feature & Reactive feature \\
\hline
Liveness & Idle frequency & Dispatching co-tasks & Destroying Co-tasks \\
Safety & Minimum frequency & CPU schedulability & Tile control \\
\hline
\end{tabular}
\caption{Classification of properties for dynamically reconfigurable systems}
\end{table}

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 \texttt{MHL} shown in Fig.11, we verify liveness with hybrid feature. \texttt{MHL} checks whether the automaton A\_DRP\_frequency starts from location \texttt{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 $e_a^0 = 1/2, e_a^1 = 1/2$ and $e_b^0 = 1$ while the co-task $b_0$ is executing.

\[
\begin{align*}
\&e_a^0 = 0 \land e_a^1 = 0 \land e_b^0 = 0, \{ \} \\
\&e_a^0 = 0 \land e_a^1 = 0 \land e_b^0 = 0, \{ \}
\end{align*}
\]

Fig. 11. Monitor $M_{HL}$

\[
\begin{align*}
\&e_a^0 = 0 \land e_a^1 = 0 \land e_b^0 = 0, \{ \} \\
\&e_a^0 = 0 \land e_a^1 = 0 \land e_b^0 = 0, \{ \}
\end{align*}
\]

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 $a$. $M_{RTL_a0}$ checks whether $r_a^0$ does not exceed $D_A - E2_A$ until the action Act. Exec. $a0$? is input when the action Act. Ready. $a0$? is input, where $r_a^0$ is a elapsed time from start of co-task $a_0$, $D_A$ 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 $a_1$ and $b_0$.

\[
\begin{align*}
\&e_a^0 = 1 \land e_b^0 = 0, \{ \}
\end{align*}
\]

Fig. 13. Monitor $M_{RTL_a0}$

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 $a_0$. $M_{RL_a0}$ 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 $a_1$ and $b_0$.

\[
\begin{align*}
\&r_a^0 > D_A - E2_A, \{ \}
\end{align*}
\]

Fig. 14. Monitor $M_{RTS}$

\[
\begin{align*}
\&r_a^0 > D_A - E2_A, \{ \}
\end{align*}
\]

Fig. 15. Monitor $M_{RL_a0}$

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 $tile > 8$.

\[
\begin{align*}
\&tile < 0, \{ \}
\end{align*}
\]

Fig. 16. Monitor $M_{RS}$
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

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, 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