Proceedings of the 8th International
Workshop on Discrete Event Systems
Ann Arbor, Michigan, USA, July 10-12, 2006

Scheduling of Periodic Tasks on a Dynamically Reconfigurable Device Using Timed Discrete Event Systems

Toshimitsu Ushio and Kenji Onogi

***Abstract*— A dynamically reconfigurable device is a device which can change its hardware configuration arbitrarily often in order to achieve the desired performance and functions. Since several tasks are executed on the device concurrently, scheduling of both task execution and reconfiguration is an important problem. In our model, the dynamically reconfigurable device is represented by a two-level hierarchical automaton, and execution of each of periodic tasks is represented by a timed discrete event system. We propose a composition rule to get an automaton, which represents non-preemptive execution of periodic tasks on the dynamically reconfigurable device. We introduce a method to get a feasible scheduling by using state feedback control.**

I. Introduction

A dynamically reconfigurable device is an important class of reconfigurable systems. It can be reconfigured arbitrarily often and reconfiguration times are on the order of milliseconds [1], [2], [3]. Several hardware architectures of the dynamically reconfigurable device have been developed. So, real-time embedded computing platforms using such a device are expected to be applied in many engineering fields such as real-time control, communication networks, mobile systems, and multimedia applications.

When several tasks are executed in the device concurrently, it is partitioned into several blocks called process blocks and each task is assigned to one of them. Depending on tasks required to be executed in the device, we have to reconfigure it dynamically while it is working. To adopt an appropriate configuration in compliance with the current executable tasks, it is an important problem how and when to reconfigure the device. This problem is reduced to a scheduling problem under the limited hardware resource, that is, the number of process blocks. Steiger *et al.* propose an operating system with on-line scheduling for the dynamically reconfigurable device [3]. A device is modeled as a one or two dimensional resource area, which is partitioned into units. Each task is modeled as a rectangular area of units needed for its execution. An unused area in it is split into several smaller rectangular areas and assigned to tasks requiring the usage of the device. Taking into consideration the execution time of the assigned tasks, this problem is reduced to a two or three dimensional bin-packing problem since time dimension is added to the dimension of the resource area. However, the computational complexity of the bin-packing problem is NP-hard and some approximation technique will be needed from practical point of view. Noguera and Badia developed a microarchitecture of a multitasking support unit which performs dynamic scheduling for assignment of tasks to process blocks [4].

On the other hand, scheduling algorithms are key issues in real-time systems and several approaches have been considered [5], [6]. Recently, applications of discrete event system (DES) theory to scheduling have been paid attention. Real-time systems are modeled by timed automata and scheduling algorithms are regarded as controllers for the task execution [7]. Abdeddaïm *et al.* derive an optimal scheduling using the shortest path algorithms for the timed automata [8]. A schedulability checking problem is reduced to a reachability problem of the timed automata and its decidability is discussed in [9], [10]. Chen and Wonham apply the supervisory control of a timed DES with forcible events proposed by Brandin and Wonham [11] to synthesis of a scheduler for a non-preemptive task set and the controlled system by the supervisor generates all safe execution sequences which guarantee that the deadlines of all tasks are met [12].

In this paper, we consider a DES model based approach to scheduling of both the task execution and the reconfiguration in the dynamically reconfigurable device. We model the task execution by a timed DES, which will be called a *task automaton*. To model both transitions between configurations and status of each process blocks in the device, we introduce an untimed two-level hierarchical automaton. Hierarchical modeling of DESs is useful for composite systems to reduce the number of states which grows exponentially in general [13] and several studies on control of hierarchical DESs can be found in [14], [15], [16], [17]. In the two-level hierarchical automaton modeling the dynamically reconfigurable device, each state of the upper-level automaton corresponds to a configuration of the device and its transition function describes reconfigurations between possible configurations while the lower-level automaton models status of each process block in the configuration. So, each state of the upper-level automaton has a lower-level automaton and, by the transition of states of the upper-level automaton, the state of the lower-level automaton also jumps to that of the lower-level automaton corresponding to the state after the transition in the upper-level automaton. The hierarchical automaton will be called a *device automaton*. To describe all possible timed sequences when a task set is executed on the dynamically reconfigurable device, we propose a composition rule of the task automata and the device automaton and introduce a new state called a *deadline-miss state*. To obtain all safe event sequences by

This work was partially supported by a Grant-in-Aid for Scientific Research (B), No. 17360198, from the Ministry of Education, Culture, Sports Science, and Technology of Japan.

T. Ushio and K. Onogi are with Graduate School of Engineering Science, Osaka University, Toyonaka, Osaka, 560-8531, Japan `{ushio, onogi}@hopf.sys.es.osaka-u.ac.jp`

which all tasks can meet their deadlines, we apply a state feedback control [18], [19], for which a control specification is that all deadline-miss states are illegal.

II. Task model

This section describes a model of the task execution. Each task is periodic and is represented by a job shop model. Each task consists of a sequence of one or more subtasks, which are executed in a specified process block of the dynamically reconfigurable device. A global clock is assumed to be discrete time and the task execution is modeled as a timed discrete event system (TDES).

Definition 1 (Task model): Let $\{\theta_1, \theta_2, \ldots, \theta_{n_t}\}$ be a task set, where θ_i is a periodic task consisting of a sequence of n_i subtasks $\theta_{i,j}$ $(k = 1, 2, \ldots, n_i)$. Each task is described by a 4-tuple $\theta_i = (p_i, d_i, \phi_i, n_i)$, where p_i is the task period, d_i is the relative deadline, ϕ_i is the initial phase, and n_i is the number of subtasks. Each subtask $\theta_{i,j}$ is also described by 4-tuple $\theta_{i,j} = (c_{i,j}, d_{i,j}, r_{i,j}, V_{i,j})$, where $c_{i,j}$ is its execution time, $d_{i,j}$ and $r_{i,j}$ are its deadline and release time relative to the invocation of the first subtask, respectively, and $V_{i,j}$ is a process block on which the subtask executes. $\mathbf{V}_i = (V_{i,1}, V_{i,2}, \ldots, V_{i,n_i})$ will be called a *visit sequence*.

Note that $r_{i,1} = 0$, $d_{i,j_1} \leq d_{i,j_2}$ $(j_1 \leq j_2)$, and $d_{i,n_i} = d_i$. Without loss of generality, we assume that $c_{i,j} \leq d_{i,j} - r_{i,j}$.

Definition 2 (Task automaton): A task automaton G_i which describes execution of a periodic task θ_i is defined by the following automaton:

$$G_i := (Q_i \times T_i, \ \ _i, \delta_i, q^t_{i,1}),$$

where Q_i is the state set of the task, $T_i = \{(\tau^p_i, \tau^e_i) \mid \tau^p_i \text{ and } \tau^e_i \text{ are integers with } 0 \leq \tau^p_i \leq p_i \text{ and } 0 \leq \tau^e_i \leq \max\{c_{i,1}, \ldots, c_{i,n_i}\}\}$ consists of two clock variables, $\ \ _i$ is the event set, $\delta_i : \ \ _i \times Q_i \times T_i \to Q_i \times T_i$ is the transition function, and $q^t_{i,1} \in Q_i \times T_i$ is the initial state. Denoted by $Q^t_i = Q_i \times T_i$ is the set of timed states of the task.

The state set Q_i is given by

$$Q_i = \{W_{i,1}, I_{i,1}, E_{i,1}, W_{i,2}, I_{i,2}, \ldots, E_{i,n_i}, D_i\},$$

where $W_{i,j}$ is a waiting state of the subtask $\theta_{i,j}$, $I_{i,j}$ is an executable state, $E_{i,j}$ is an execution state, and D_i is a deadline-miss state. The initial state $q^t_{i,1} = (W_{i,1}, 0, 0)$. The clock variable τ^p_i is reset to 0 when the first subtask is invoked while τ^e_i is reset to 0 whenever the execution of a subtask completes. The event set $\ \ _i$ consists of the following four kinds of events: $\alpha_{i,j}$, $\beta_{i,j}$, $\gamma_{i,j}$, and $tick$. The events $\alpha_{i,j}$, $\beta_{i,j}$, and $\gamma_{i,j}$ mean the execution start, the execution completion, and the invocation of the subtask $\theta_{i,j}$, respectively. By the occurrence of the event $tick$, a global clock is advanced by one unit time. The event $\alpha_{i,j}$ is controllable and forcible while the others are uncontrollable. We use the following notation: $\ \ ^\alpha_i = \{\alpha_{i,j} | 1 \leq j \leq n_i\}$, $\ \ ^\beta_i = \{\beta_{i,j} | 1 \leq j \leq n_i\}$, $\ \ ^\gamma_i = \{\gamma_{i,j} | 1 \leq j \leq n_i\}$. Denoted by $\ \ ^c_i$ is the set of controllable events, and $\ \ ^{uc}_i$ the set of uncontrollable events except $tick$:

$$\ \ ^c_i = \ \ ^\alpha_i \quad \text{and} \quad \ \ ^{uc}_i = \ \ ^\beta_i \cup \ \ ^\gamma_i.$$

The transition function δ_i is defined as follows:

- The event $\alpha_{i,j} \in \ \ ^\alpha_i$ is enabled at states $(I_{i,j}, t^p, t^e)$ with $t^p \leq d_{i,j} - c_{i,j}$, which ensures that the deadline of the subtask $\theta_{i,j}$ can be met. By the occurrence of $\alpha_{i,j}$, the current state moves to the execution state and the clock τ^e_i is reset to 0, that is,

$$\delta_i(\alpha_{i,j}, (I_{i,j}, t^p, t^e)) = (E_{i,j}, t^p, 0).$$

- The event $\beta_{i,j} \in \ \ ^\beta_i$ occurs when the execution of the subtask $\theta_{i,j}$ completes, that is, at the state $(E_{i,j}, t^p, c_{i,j})$ for any t^p. Note that, whenever the system G_i is at the state $(E_{i,j}, t^p, c_{i,j})$, $tick$ never occurs just before $\beta_{i,j}$ occurs. By the occurrence of $\beta_{i,j}$, the current state moves to the waiting state of the next subtask: for $1 \leq j < n_i$,

$$\delta_i(\beta_{i,j}, (E_{i,j}, t^p, c_{i,j})) = (W_{i,j+1}, t^p, c_{i,j}),$$

and for $k = n_i$,

$$\delta_i(\beta_{i,n_i}, (E_{i,n_i}, t^p, c_{i,n_i})) = (W_{i,1}, t^p, c_{i,n_i}).$$

- The event $\gamma_{i,j} \in \ \ ^\gamma_i$ $(j \neq 1)$ occurs at the state $(W_{i,j}, t^p, t^e)$ if $t^p \geq r_{i,j}$. Note that $tick$ never occurs just before $\gamma_{i,j}$ occurs. By the occurrence of $\gamma_{i,j}$, the current state moves to the executable state: for any $j \neq 1$ and $t^p \geq r_{i,j}$,

$$\delta_i(\gamma_{i,j}, (W_{i,j}, t^p, t^e)) = (I_{i,j}, t^p, t^e).$$

For the first job, $\gamma_{i,1}$ occurs at the state $(W_{i,1}, \phi_i, 0)$. The current state moves to the executable state and the clock τ^p_i is reset to 0 at the same time:

$$\delta_i(\gamma_{i,1}, (W_{i,1}, \phi_i, 0)) = (I_{i,1}, 0, 0).$$

When the next job is invoked, $\gamma_{i,1}$ occurs at the state $(W_{i,1}, p_i, t^e)$:

$$\delta_i(\gamma_{i,1}, (W_{i,1}, p_i, t^e)) = (I_{i,1}, 0, t^e).$$

We suppose $(W_{i,1}, \phi_i, 0) = (W_{i,1}, p_i, t^e)$ for any t^e since $\delta_i(\gamma_{i,1}, (W_{i,1}, \phi_i, 0)) = \delta_i(\gamma_{i,1}, (W_{i,1}, p_i, t^e)) = (I_{i,1}, 0, t^e)$.

- If the system is in the executable state $(I_{i,j}, t^p, t^e)$ where $t^p \geq d_{i,j} - c_{i,j}$, then the task θ_i does not meet its deadline. So, the occurrence of $tick$ leads to the deadline-miss state when $t^p \geq d_{i,j} - c_{i,j}$:

$$\delta_i(tick, (I_{i,j}, t^p, t^e)) = (D_i, 0, 0).$$

If $t^p < d_{i,j} - c_{i,j}$ holds, then the occurrence of $tick$ advances the global clock by one unit time:

$$\delta_i(tick, (I_{i,j}, t^p, t^e)) = (I_{i,j}, t^p + 1, t^e).$$

The occurrence of $tick$ is also defined at the waiting states $(W_{i,j}, t^p, t^e)$ if $t^p < r_{i,j}$, and at the execution states $(E_{i,j}, t^p, t^e)$ if $t^e < c_{i,j}$:

$$\delta_i(tick, (W_{i,j}, t^p, t^e)) = (W_{i,j}, t^p + 1, t^e),$$
$$\delta_i(tick, (E_{i,j}, t^p, t^e)) = (E_{i,j}, t^p + 1, t^e + 1).$$

Example 1: Figure 1 shows an automaton which represents the execution of a periodic task θ_i: the number of subtasks

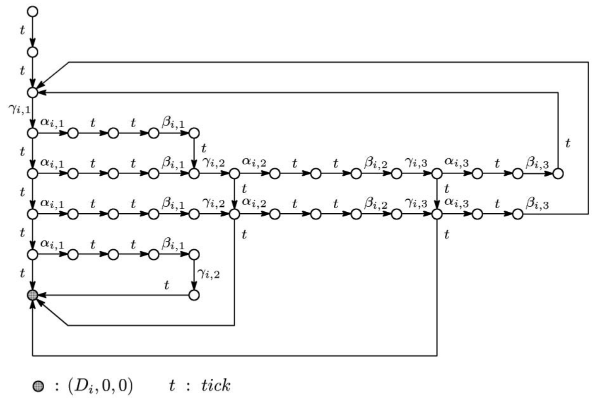

Fig. 1. Task automaton of Example 1.

$n_i = 3$, the initial period $p_i = 7$, the deadline $d_i = 7$, and the phase $\phi_i = 2$. The execution time, the release time, and the deadline of subtasks are $(c_{i,1}, c_{i,2}, c_{i,3}) = (2, 2, 1)$, $(r_{i,1}, r_{i,2}, r_{i,3}) = (0, 3, 5)$, and $(d_{i,1}, d_{i,2}, d_{i,3}) = (5, 6, 7)$, respectively.

III. Dynamically reconfigurable device

A. Two-level hierarchical automaton model

A dynamically reconfigurable device can form several configurations and change to an appropriate one dynamically in compliance with the current request of the execution of subtasks. Each configuration is composed of one or more process blocks. Let $\{b_1, b_2, \ldots, b_L\}$ be the set of all possible process blocks realized in configurations of the device. Each process block is used for the execution of a certain subtask. A process block can process just one subtask at a time. For each l $(1 \leq l \leq L)$, let $\eta(C, b_l)$ be the number of the process blocks b_l in the configuration C. So, in the configuration C, $\eta(C, b_l)$ subtasks can execute using the process block b_l concurrently.

The dynamically reconfigurable device is modeled by an untimed two-level hierarchical automaton called a *device automaton*. The upper-level automaton represents transition relations between possible configurations of the device while a state of the lower-level automaton represents the status of the process blocks.

Definition 3 (Upper-level automaton): An upper-level automaton G^U is defined as follows:

$$G^U := (Q^U, \quad, \xi, C_1),$$

where $Q^U = \{C_1, C_2, \ldots, C_{n_c}\}$ is the set of all possible configurations of the device, is the set of events corresponding to the change of configurations, $\xi : \quad \times Q^U \to Q^U$ is a partial function which defines transitions between the configurations, which correspond to reconfigurations, and $C_1 \in Q^U$ is the initial configuration. Denoted by $\lambda_{k,k'} \in$ is the event by which the configuration of the device changes from C_k to $C_{k'}$, that is, $\xi(\lambda_{k,k'}, C_k) = C_{k'}$ if it is defined.

Definition 4 (Lower-level automaton): A lower-level automaton describes status of the process blocks in each configuration. The lower-level automaton assigned to each state $C_k \in Q^U$ of the upper-level automaton is defined as follows:

$$G_k^L := (Q_k^L, \quad ^L, \delta_k^L, x_{k,1}) \qquad (k = 1, 2, \ldots, n_c),$$

where Q_k^L is the set of states, and ^{L} is the set of events, $\delta_k^L :$ $^L \times Q_k^L \to Q_k^L$ is the state transition function, and $x_{k,1} \in Q_k^L$ is the initial state. Each state $x_{k,h} \in Q_k^L$ represents the status of the process blocks in the configuration C_k. We extend η in such a way that $\eta(x_{k,h}, b_l)$ $(l = 1, 2, \ldots, L)$ represents the number of the process blocks b_l which is working at the state $x_{k,h} \in Q_k^L$. Note that $0 \leq \eta(x_{k,h}, b_l) \leq \eta(C_k, b_l)$ and that $\eta(x_{k,h}, b_l) = 0$ if $\eta(C_k, b_l) = 0$. We suppose $\eta(x_{k,1}, b_l) = 0$ $(1 \leq l \leq L)$ for each $C_k \in Q^U$. The event set ^{L} is shared with all lower-level automata and consists of two kinds of events α_l and β_l: by the occurrence of α_l, a subtask is dispatched to the process block b_l and β_l occurs when a subtask executing on b_l completes. The transition function δ_k^L is defined as follows:

$$\begin{aligned}
&\eta(\delta_k^L(\alpha_l, x_{k,h}), b_l) = \eta(x_{k,h}, b_l) + 1 \leq \eta(C_k, b_l),\\
&\eta(\delta_k^L(\alpha_l, x_{k,h}), b_m) = \eta(x_{k,h}, b_m) \quad \text{if } m \neq l,\\
&\eta(\delta_k^L(\beta_l, x_{k,h}), b_l) = \eta(x_{k,h}, b_l) - 1 \geq 0,\\
&\eta(\delta_k^L(\beta_l, x_{k,h}), b_m) = \eta(x_{k,h}, b_m) \quad \text{if } m \neq l.
\end{aligned}$$

Other transitions are undefined.

From Definition 3, $\delta_k^L(\alpha_l, x_{k,h})$ is defined if $\eta(x_{k,h}, b_l) + 1 \leq \eta(C_k, b_l)$, otherwise it is undefined. Note that $Q_k^L \cap Q_{k'}^L = \emptyset$ if $k \neq k'$, $^L \cap (\cup_i \quad_i) = \emptyset$, and $\cap \quad ^L = \emptyset$.

When a state transition occurs in the upper-level automaton, which means that the configuration of the device changes, the state of the lower-level automaton in the new configuration is specified by the reset function $\mathcal{F}_{k,k'} : Q_k^L \to Q_{k'}^L$, which is a partial function. For example, $x_{k',h'} = \mathcal{F}_{k,k'}(x_{k,h})$ if $C_{k'} = \xi(\lambda_{k,k'}, C_k)$, $\eta(x_{k,h}, b_l) = \eta(x_{k'h'}, b_l) \leq \eta(C_{k'}, b_l)$. Enablingness of the event $\lambda_{k,k'}$ depends on the current state of the lower-level automaton G_k^L, that is, it is enabled at C_k with the state $x_{k,h}$ of G_k^L if $\mathcal{F}_{k,k'}(x_{k,h})$ is defined. So, the reset function is also a guard of $\lambda_{k,h}$. The two-level hierarchical automaton defined by $(G^U, \{G_k^L\}, \{\mathcal{F}_{k,k'}\})$ will be called a *device automaton*.

From Definitions 3 and 4, the initial state of the device automaton is the initial state $x_{1,1}$ of the initial configuration C_1.

Note that the event α_l does not specify which subtask dispatches a job to the process block b_l. Since this information is given from the task automata, we define a mapping $\mathcal{P} : \cup_i(\ _i^\alpha \cup \ _i^\beta) \to \ ^L$ as follows: if $V_{i,j} = b_l$, then $\mathcal{P}(\alpha_{i,j}) = \alpha_l$ and $\mathcal{P}(\beta_{i,j}) = \beta_l$.

Example 2: Figure 2 shows an example of a device automaton of a dynamically reconfigurable device, where $Q^U = \{C_1, C_2\}$, $\xi(\lambda_{1,2}, C_1) = C_2$, $\xi(\lambda_{2,1}, C_2) = C_1$, $(\eta(C_1, b_1), \eta(C_1, b_2), \eta(C_2, b_1), \eta(C_2, b_2)) = (2, 1, 1, 2)$, $\eta(x_{1,1}, b_1) = \eta(x_{1,1}, b_2) = \eta(x_{2,1}, b_1) = \eta(x_{2,1}, b_2) = 0$, which means that no process block is used at the states $x_{1,1}$ and $x_{2,1}$. We suppose that the reset function is defined if all working process blocks before the reconfiguration exist after

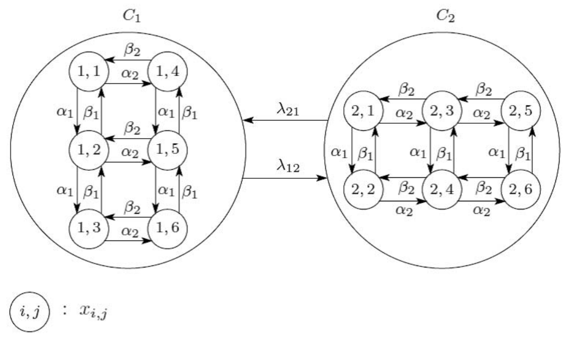

Fig. 2. Device automaton of Example 2.

it. Thus, we have $\mathcal{F}_{1,2}(x_{1,1}) = x_{2,1}$, $\mathcal{F}_{2,1}(x_{2,1}) = x_{1,1}$, $\mathcal{F}_{1,2}(x_{1,2}) = x_{2,2}$, $\mathcal{F}_{1,2}(x_{1,4}) = x_{2,3}$, $\mathcal{F}_{1,2}(x_{1,5}) = x_{2,4}$, $\mathcal{F}_{2,1}(x_{2,2}) = x_{1,2}$, $\mathcal{F}_{2,1}(x_{2,3}) = x_{1,4}$, and $\mathcal{F}_{2,1}(x_{2,4}) = x_{1,5}$.

B. Subtask execution

If the event $\alpha_{i,j}$, which makes the execution of the subtask $\theta_{i,j}$ $(V_{i,j} = b_l)$ start, occurs in the automaton G_i, then $\alpha_l = \mathcal{P}(\alpha_{i,j})$ also occurs in the lower-level automaton G_k^L synchronously. Suppose that the configuration of the upper-level automaton is C_k. Then, the occurrence of the event α_l at the state $x_{k,h}$ requires the condition $\eta(x_{k,h}, b_l) < \eta(C_k, b_l)$ since $\eta(\delta_k^L(\alpha_l, x_{k,h}), b_l) = \eta(x_{k,h}, b_l) + 1$, which means that the number of the working process blocks is increased by 1 at the state after the occurrence. If there is not a process block b_l, or all process blocks are working at the configuration C_k, then $\eta(x_{k,h}, b_l) = \eta(C_k, b_l)$ so that α_l is disabled. When the current state of the lower-level automaton is such a state $x_{k,h}$, for each $i \in \{1, \ldots, n_t\}$, $\alpha_{i,j}$ such that $\mathcal{P}(\alpha_{i,j}) = \alpha_l$ is disabled at any state q_i^t of the task automaton G_i.

When the subtask $\theta_{i,j}$ executing on the process block b_l of the configuration C_k completes, $\beta_{i,j}$ in G_i and $\beta_l = \mathcal{P}(\beta_{i,j})$ in G_k^L occur synchronously. Both $\gamma_{i,j}$ and $tick$ do not have any effect on the status of the process block. So, no event occurs in G_k^L synchronously with their occurrences.

C. Reconfiguration

In this paper, for simplicity, we assume that the dynamically reconfigurable device can form an arbitrary configuration if it is not used for task execution at all. So, there is a transition between every pair of configurations, C_k and $C_{k'}$ $(1 \le k, k' \le n_c,\ k \ne k')$, that is, $\xi(\lambda_{k,k'}, C_k)$ is always defined.

When there is no process block used for the execution of a subtask in the current configuration, we may have it by reconfiguration. For example, Suppose that the event $\alpha_{i,j}$ with $\mathcal{P}(\alpha_{i,j}) = \alpha_l$ is enabled at q_i^t in the task automaton G_i and that the current state of the device automaton is $(C_k, x_{k,h})$, where $C_k \in Q^U$, $x_{k,h} \in Q_k^L$, and $\eta(x_{k,h}, b_l) = \eta(C_k, b_l)$. Then, $\alpha_{i,j}$ can not occur since there is no idle process block b_l. If there is a configuration $C_{k'}$ such that $\eta(C_{k'}, b_l) \ge \eta(C_k, b_l) + 1$ and $\mathcal{F}_{k,k'}(x_{k,h}) = x_{k',h'}$, then $\alpha_{i,j}$ comes to occur after the reconfiguration to $C_{k'}$ in the device automaton. We introduce a pair of events $\omega \in$ to describe both changes of states of the tasks and the configurations of the device, where $=\ \times \cup_i\ _i$ is the set of pairs of events. In the example described above, the reconfiguration with the start of the subtask $\theta_{i,j}$ is described by the occurrence of the pair of events $\omega = (\lambda_{k,k'}, \alpha_{i,j})$. But, we may restrict the occurrence of the event $\lambda_{k,k'}$ of the upper-level automaton such that the execution of a subtask must start just after the reconfiguration is done. Thus, the pairs of events can avoid meaningless reconfigurations such as the infinite sequences of reconfigurations without the start of any task execution. Note that events related to task execution do not always occur synchronously with reconfigurations. The occurrence of an event $\sigma \in \cup_i\ _i$ without any reconfiguration is represented by the pair of events $(\varepsilon, \sigma) \in$, where ε means that no event occurs in the upper-level automaton. The following sequence is an example of a sequence of pairs of events:

$$\ldots, (\varepsilon, \beta_{i,j}), (\varepsilon, tick), (\varepsilon, \gamma_{i,j+1}), (\lambda_{k,k'}, \alpha_{i,j+1}), \ldots,$$

where the execution of $\theta_{i,j}$ has completed and the execution of $\theta_{i,j+1}$ starts after a reconfiguration.

IV. Composition of automata

To describe all possible sequences of the task set $\{\theta_1, \theta_2, \ldots, \theta_{n_t}\}$ executing on the dynamically reconfigurable device, we propose a composition rule of the set of task automata $G_i (i = 1, 2, \ldots, n_t)$ and the device automaton $(G^U, \{G_k^L\}, \{\mathcal{F}_{k,k'}\})$.

Definition 5 (Composite automaton): We consider the set of task automata G_i $(i = 1, 2, \ldots, n_t)$ modeling the task set and the device automaton $(G^U, \{G_k^L\}, \{\mathcal{F}_{k,k'}\})$. A composite automaton of these automata is defined as follows:

$$G := (\mathbf{Q},\ , \delta, \mathbf{q}_1, \mathbf{D}),$$

where $\mathbf{Q} = Q^U \times \cup_k Q_k^L \times Q_1^t \times Q_2^t \times \cdots Q_{n_t}^t$ is the state, $=\ \times \cup_i\ _i$ is the set of pairs of events, $\delta :\ \times \mathbf{Q} \to \mathbf{Q}$ is the transition function, $\mathbf{q}_1 = (C_1, x_{1,1}, q_{1,1}^t, q_{2,1}^t, \cdots, q_{n_t,1}^t)$ is the initial state, and $\mathbf{D} = \{(C_k, x_{k,h}, q_1^t, q_2^t, \ldots, q_{n_t}^t) \in \mathbf{Q} \mid \exists i\ (1 \le i \le n_t)$ such that $q_i^t = (D_i, 0, 0)\}$ is the set of deadline-miss states.

The transition function δ is defined as follows: for each $\mathbf{q} = (C_k, x_{k,h}, q_1^t, q_2^t, \ldots, q_{n_t}^t)$ and $\omega = (\lambda, \sigma)$,

- if $\lambda = \lambda_{k,k'} \in$ with $C_{k'} = \xi(\lambda_{k,k'}, C_k)$ and $\sigma \in\ _i^\alpha$ with $\mathcal{P}(\sigma) = \alpha_l$,

$$\delta(\omega, \mathbf{q}) = \begin{cases} (C_{k'}, \delta_{k'}^L(\alpha_l, \mathcal{F}_{k,k'}(x_{k,h})), q_1^t, \ldots, \\ \quad \delta_i(\sigma, q_i^t), \ldots, q_{n_t}^t) \\ \quad \text{if } \mathcal{F}_{k,k'}(x_{k,h})! \\ \qquad \wedge \eta(\mathcal{F}_{k,k'}(x_{k,h}), b_l) < \eta(C_{k'}, b_l), \\ \text{undefined} \qquad \text{otherwise;} \end{cases}$$

- if $\lambda = \varepsilon$ and $\sigma \in\ _i^\alpha \cup\ _i^\beta$,

$$\delta(\omega, \mathbf{q}) = \begin{cases} (C_k, \delta_k^L(\mathcal{P}(\sigma), x_{k,h}), q_1^t, \ldots, \\ \quad \delta_i(\sigma, q_i^t), \ldots, q_{n_t}^t) \\ \quad \text{if } \delta_k^L(\mathcal{P}(\sigma), x_{k,h})! \wedge \delta_i(\sigma, q_i^t)!, \\ \text{undefined} \qquad \text{otherwise;} \end{cases}$$

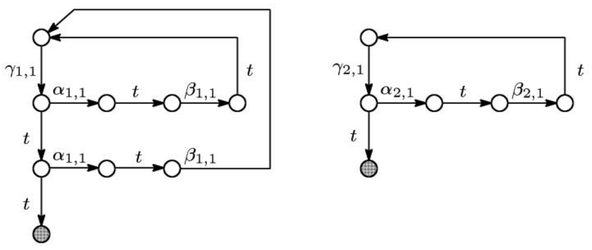

Fig. 3. Task automata of θ_1 and θ_2 of Example 3.

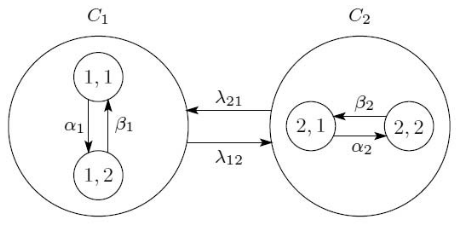

Fig. 4. Device automaton of Example 3.

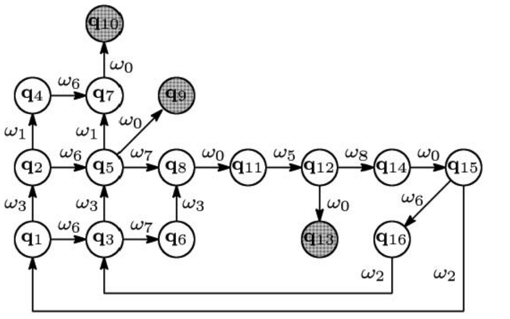

Fig. 5. Composite automaton of Example 3.

- if $\lambda = \varepsilon$ and $\sigma \in \ {}^{\gamma}_{i}$,

$$\delta(\omega, \mathbf{q}) = \begin{cases} (C_k, x_{k,h}, q_1^t, \ldots, \delta_i(\sigma, q_i^t), \ldots, q_{n_t}^t) \\ \quad \text{if } \delta_i(\sigma, q_i^t)!, \\ \text{undefined} \qquad \text{otherwise;} \end{cases}$$

- if $\lambda = \varepsilon$ and $\sigma = tick$,

$$\delta(\omega, \mathbf{q}) = \begin{cases} (C_k, x_{k,h}, \delta_1(\sigma, q_1^t), \ldots, \delta_{n_t}(\sigma, q_{n_t}^t)) \\ \quad \text{if } (1 \le \forall i \le n_t)\delta_i(\sigma, q_i^t)!, \\ \text{undefined} \qquad \text{otherwise.} \end{cases}$$

Example 3: We show a simple example of a composite automaton. We consider two tasks $\theta_1,\ \theta_2$, where $n_1 = n_2 = 1$, $p_1 = p_2 = 2$, $\phi_1 = \phi_2 = 0$, $e_{1,1} = e_{2,1} = 1$, $d_1 = d_{1,1} = 2$, and $d_2 = d_{2,1} = 1$ (see Figure 3). $V_{1,1} = b_1$ and $V_{2,1} = b_2$. The dynamically reconfigurable device has two configurations: $\eta(C_1, b_1) = 1,\ \eta(C_1, b_2) = 0,\ \eta(C_2, b_1) = 0$, and $\eta(C_2, b_2) = 1$ (see Figure 4). The reset functions are given by $\mathcal{F}_{1,2}(x_{1,1}) = x_{2,1}$ and $\mathcal{F}_{2,1}(x_{2,1}) = x_{1,1}$. Let $\mathcal{P}(\alpha_{1,1}) = \alpha_1,\ \mathcal{P}(\alpha_{2,1}) = \alpha_2,\ \mathcal{P}(\beta_{1,1}) = \beta_1$, and $\mathcal{P}(\beta_{2,1}) = \beta_2$. Shown in Figure 5 is the composite automaton of these automata. The initial state is $\mathbf{q}_1$ and the set of the deadline-miss states is $\mathbf{D} = \{\mathbf{q}_9,\ \mathbf{q}_{10},\ \mathbf{q}_{12}\}$. The list of pairs of events is as follows:

$$\begin{aligned} &\omega_0 = (\varepsilon, tick), && \omega_1 = (\varepsilon, \alpha_{1,1}), && \omega_2 = (\varepsilon, \beta_{1,1}), \\ &\omega_3 = (\varepsilon, \gamma_{1,1}), && \omega_4 = (\varepsilon, \alpha_{2,1}), && \omega_5 = (\varepsilon, \beta_{2,1}), \\ &\omega_6 = (\varepsilon, \gamma_{2,1}), && \omega_7 = (\lambda_{1,2}, \alpha_{2,1}), && \omega_8 = (\lambda_{2,1}, \alpha_{1,1}). \end{aligned}$$

V. State feedback control

When a set of periodic tasks is executed on the dynamically reconfigurable device, a scheduling which guarantees that deadlines of all tasks are met is called feasible one. We synthesize a state feedback controller to obtain all non-preemptive feasible schedulings.

Let $\ ^c = (\{\varepsilon\} \cup \quad) \times \cup_i \ {}^c_i$ and $\ ^{uc} = \{\varepsilon\} \times \cup_i \ {}^{uc}_i$ be the sets of controllable and uncontrollable pair of events, respectively. Note that all pairs of events in $\ ^c$ are forcible, that is, they are forced to occur by the controller if they are enabled in the composite automaton G. If a pair of events is forced, $(\varepsilon, tick)$ and other pairs in $\ ^c$ which are not forced is disabled. If several pairs of events are forced to occur, one of them or an enabled uncontrollable pair of event in $\ ^{uc}$ must occur.

In order to obtain all feasible schedulings, we synthesize a state feedback controller $\ = (\pi_p, \pi_f)$ for the composite automaton G, where $\pi_p : \mathbf{Q} \to 2^{\Omega^c}$ and $\pi_f : \mathbf{Q} \to 2^{\Omega^c}$ are sets of the permitted and the forced pairs of events, respectively [18], [20]. In a state $\mathbf{q} \in \mathbf{Q}$ where $\delta(\omega, \mathbf{q})$ is undefined, we have $\omega \notin \pi_p(\mathbf{q})$ and $\omega \notin \pi_f(\mathbf{q})$.

When $\pi_f(\mathbf{q}) \neq \emptyset$ in $\mathbf{q}$, $\omega \in \pi_f(\mathbf{q})$ is forced to occur just before $(\varepsilon, tick)$ occurs. So, $(\varepsilon, tick)$ and all pairs of events which are not in $\pi_f(\mathbf{q})$ are disabled, but ones in $\pi_f(\mathbf{q}) \cup \ ^{uc}$ are enabled.

If $\pi_f(\mathbf{q}) = \emptyset$, $\delta(\omega, \mathbf{q})$ is defined, and $\omega \in \pi_p(\mathbf{q}) \cup \ ^{uc} \cup \{(\varepsilon, tick)\}$, then ω is enabled at the state $\mathbf{q}$. If ω is not in $\pi_p(\mathbf{q})$, the occurrence of the event ω is prohibited at $\mathbf{q}$ by the controller.

A state feedback controller $\ $ is said to be permissive if the reachability set of the composite automaton G controlled by $\ $ does not contain any deadline-miss state, that is, any event sequence in the controlled automaton corresponds to a feasible scheduling. We propose a method of computing the maximally permissive state feedback controller. Denoted by $\mathbf{D}^A$ is a set of states from which deadline-miss states are reachable under any control. Intuitively, the controller disables any pair of events whose occurrence leads to a state in $\mathbf{D}^A$. So, we can synthesize the controller $\ = (\pi_p, \pi_f)$ by modifying an algorithm for computation of the supremal control-invariance predicate [18], [19], [20].

[Algorithm]

step 1 As initialization, $\mathbf{D}^A = \emptyset$. For all $\mathbf{q} \in \mathbf{Q}$, $\pi_f(\mathbf{q}) = \emptyset$ and $\pi_p(\mathbf{q}) = \{\omega^c \mid \delta(\omega^c, \mathbf{q})!,\ \omega^c \in \ ^c\}$.

step 2 For all $\mathbf{q} \in \mathbf{Q} \setminus (\mathbf{D} \cup \mathbf{D}^A)$ where $\omega^{uc} \in \ ^{uc}$, which meets $\delta(\omega^{uc}, \mathbf{q}) \in \mathbf{D} \cup \mathbf{D}^A$, exists, $\mathbf{D}^A \leftarrow \mathbf{D}^A \cup \{\mathbf{q}\}$.

step 3 For all $\omega^c \in \pi_p(\mathbf{q})$ and $\mathbf{q} \in \mathbf{Q} \setminus (\mathbf{D} \cup \mathbf{D}^A)$ which meet $\delta(\omega^c, \mathbf{q}) \in \mathbf{D} \cup \mathbf{D}^A$, $\pi_p(\mathbf{q}) \leftarrow \pi_p(\mathbf{q}) \setminus \{\omega^c\}$.

step 4 For all $\mathbf{q} \in \mathbf{Q} \setminus (\mathbf{D} \cup \mathbf{D}^A)$ which meet $\delta((\varepsilon, tick), \mathbf{q}) \in \mathbf{D} \cup \mathbf{D}^A$, $\mathbf{D}^A \leftarrow \mathbf{D}^A \cup \{\mathbf{q}\}$ if $\omega^c \in \pi_p(\mathbf{q})$ which meets $\delta(\omega^c, \mathbf{q}) \notin \mathbf{D} \cup \mathbf{D}^A$ does not exist.

step 5 If there is no change in step 2, step 3, and step 4, then go to step 6. Otherwise, go to step 2.

step 6 At $\mathbf{q} \in \mathbf{Q} \setminus (\mathbf{D} \cup \mathbf{D}^A)$ which meets $\delta((\varepsilon, tick), \mathbf{q}) \in \mathbf{D} \cup \mathbf{D}^A$, for all $\omega^c \in \pi_p(\mathbf{q})$, $\pi_f(\mathbf{q}) \leftarrow \pi_f(\mathbf{q}) \cup \{\omega^c\}$.

If $\mathbf{q}_1 \in \mathbf{D}^A$, there is no feasible scheduling.

We will prove briefly that the feedback controller computed by the above algorithm is maximally permissive so that any feasible scheduling can be generated by the controlled composite automaton. Suppose that the system is at a state $\mathbf{q} \in \mathbf{Q} \setminus (\mathbf{D} \cup \mathbf{D}^A)$. Since $\mathbf{q} \notin \mathbf{D} \cup \mathbf{D}^A$, it is shown by step 2 of the algorithm that no state in $\mathbf{D} \cup \mathbf{D}^A$ is reachable from $\mathbf{q}$ by the occurrence of any pair of events. It is shown by step 3 that any transition into a state in $\mathbf{D} \cup \mathbf{D}^A$ by $\omega^c \in \ {}^c$ is disabled by the controller. A transition by $(\varepsilon, tick)$ is also disabled by the control since an event $\omega^c \in \ {}^c$ can be forced as mentioned in steps 4 and 6. Therefore, no state in $\mathbf{D} \cup \mathbf{D}^A$ is reachable from the initial state under the state feedback controller. Moreover, it is easily shown that the state feedback controller is maximally permissive. The definition of the composite rule implies that, in the controlled composite automaton, a pair of events (ε, σ) with $\sigma \in (\cup_i \ {}_i^{uc}) \cup \{tick\}$ is enabled at any state where all controllable pairs of events are disabled, and at least one uncontrollable pair of events is enabled at other states. So, deadlock does not exist. Since the event set of the composite automaton is given by the pairs of events and any task automaton forms a cycle by its definition, livelock does not exist. Therefore, if $\mathbf{q}_1 \notin \mathbf{D}^A$, we can get at least one scheduling by which all tasks meet their deadlines.

Example 4: We apply the proposed algorithm to the composite automaton shown in Figure 5, where

$$ {}^c = \{\omega_1, \omega_4, \omega_7, \omega_8\}, \qquad {}^{uc} = \{\omega_2, \omega_3, \omega_5, \omega_6\}, $$
$$ \omega_0 = (\varepsilon, tick), \qquad \mathbf{D} = \{\mathbf{q}_9, \mathbf{q}_{10}, \mathbf{q}_{13}\}. $$

Shown in Fig. 6 is the controlled composite automaton, where all edges disabled by the controller are deleted. We can get a feasible scheduling from the automaton. For example, the following event sequence is an example of a feasible scheduling.

$$ \omega_6, \omega_7, \omega_3, \omega_0, \omega_5, \omega_8, \omega_0, \omega_2, \omega_6, \ldots, $$

which corresponds to the following event sequence:

$$ \gamma_{2,1}, \lambda_{1,2}, \alpha_{2,1}, \gamma_{1,1}, tick, \beta_{2,1}, \lambda_{2,1}, \alpha_{1,1}, tick, \beta_{1,1}, \gamma_{2,1}, \ldots $$

VI. Conclusion

In this paper, we presented a method for scheduling the non-preemptive execution of a set of periodic tasks on a dynamically reconfigurable device. In this method, both the task execution and the reconfiguration are modeled as automata and we propose a composition rule of the automata. Finally, state feedback control is applied to it to get all feasible schedulings.

We assume that configurations of process blocks are given beforehand. On-line allocation of areas to process blocks, however, is important to achieve more flexible reconfiguration. So, control with the on-line allocation is future work.

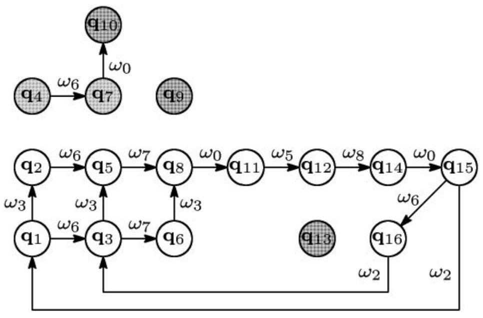

Fig. 6. Controlled automaton of Example 4.

References

[1] H. Amano, A. Jouraku, and K. Anjo, "A dynamically adaptive hardware on dynamically reconfigurable processor," *IEICE Trans.*, vol. E86-B, no. 12, pp. 3385-3391, 2003.

[2] M. Meribout and M. Motomura, "A combined approach to high-level synthesis for dynamically reconfigurable systems," *IEEE Trans. Computers*, vol. 53, no. 12, pp. 1508–1522, 2004.

[3] C. Steiger, H. Walder, and M. Platzner, "Operating systems for reconfigurable embedded platforms: online scheduling of real-time tasks," *IEEE Trans. Computers*, vol. 53, no. 11, pp. 1393–1407, 2004.

[4] J. Noguera and R. M. Badia, "Multitasking on reconfigurable architectures: microarchitecture support and dynamic scheduling," *ACM Trans. Embedded Computing Systems*, vol. 3, no. 2, pp. 385–406, 2004.

[5] G. C. Buttazzo, *Hard Real-Time Computing Systems Predictable Scheduling Algorithms and Applications*, 2nd Edition, Springer, 2005.

[6] J. W. S. Liu, *Real-Time Systems*, Prentice-Hall, 2000.

[7] K. Altisen, G. Gössler, and J. Sifakis, "Scheduler modeling based on the controller synthesis paradigm," *Real-Time Systems*, vol. 23, no. 1, pp. 55-84, 2002.

[8] Y. Abdeddaïm, E. Asarin and O. Maler, "Scheduling with timed automata," *Theoretical Computer Science*, vol. 354, pp. 272–300, 2006.

[9] E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi, "Schedulability analysis of fixed-priority systems using timed automata," *Theoretical Computer Science*, vol. 354, pp. 301–317, 2006.

[10] P. Krčál and W. Yi, "Decidable and undecidable problems in schedulability analysis using timed automata," *Proc. TACAS'04*, pp. 236–250, Springer, 2004.

[11] B. A. Brandin and W. M. Wonham,"Supervisory control of timed discrete-event systems," *IEEE Trans. Automatic Control*, vol. 39, no. 2, pp. 329–342, 1994.

[12] P. C. Y. Chen and W. M. Wonham, "Real-time supervisory control of a processor for non-preemptive execution of periodic tasks," *Real-Time Systems*, vol. 23, no. 3, pp. 183–208, 2002.

[13] D. Harel and A. Pnueli, "On the development of reactive systems," *Logics and Models of Concurrent Systems,* NATO ASI, vol. 13, pp. 477–498, 1985.

[14] Y. Brave and M. Heymann, "Control of discrete event systems modeled as hierarchical state machines," *IEEE Trans. Automatic Control*, vol. 38, no. 12, pp. 1803–1818, 1993.

[15] H. Marchand and B. Gaudin, "Supervisory control problem of hierarchical finite state machines," *Proc. the 41st IEEE Conference on Decision and Control, Las Vegas, Nevada USA*, pp. 1199–1204, 2002.

[16] B. Gaudin and H. Marchand, "Supervisory control of product and hierarchical discrete event systems," *European Journal of Control*, vol. 20, no. 2, pp. 131–145, 2004.

[17] C. Ma and W. M. Wonham, *Nonblocking Supervisory Control of State Tree Structures*, Lecture Notes in Conr. & Inf., vol. 317, Springer, 2005.

[18] P. J. Ramadge and W. M. Wonham, "Modular feedback logic for discrete event systems," *SIAM J. Control and Optimization*, vol. 25, no. 5, pp. 1208–1218, 1987.

[19] T.-J. Ho, "A new approach to synthesis problem in timed discrete-event systems," *Int. J. Control*, vol. 73, no. 6, pp. 505–519, 2000.

[20] T. Ushio and S. Takai, "Control-invariance of hybrid systems with forcible events," *Automatica*, vol. 41, no. 4, pp. 669–675, 2005.