Formal Verification of Speed-Dependent Asynchronous Circuits Using Symbolic Model Checking of Branching Time Regular Temporal Logic [1]

Kiyoharu HAMAGUCHI†, Hiromi HIRAISHI‡ and Shuzo YAJIMA†
† Department of Information Science, Faculty of Engineering,
Kyoto University, Kyoto, 606, JAPAN.
(E-mail: hama@kuis.kyoto-u.ac.jp)
‡ Department of Information & Communication Sciences,
Kyoto Sangyo University, Kita-ku, Kyoto, 603, JAPAN.

Abstract Firstly, we show how to deal with bounded *uncertain delays* of (speed-dependent) asynchronous circuits for symbolic model checking based on temporal logic. We adopt discrete-time model. In the modeling of uncertain delays, we consider two models, i.e. static delay and dynamic delay. These models are interpreted as parameterized sequential machines and nondeterministic sequential machines respecitively.

Secondly, we show a *symbolic model checking* algorithm for the above sequential machines. As a specification description language, a temporal logic named Branching Time Regular Temporal Logic (BRTL) is employed.

A prototype of verification system based on the proposed method has been implemented and some experimental results are reported.

1 Introduction

Asynchronous parts in a logic system tend to be small, because of its difficulty in design. Their timing verification, however, has to be strict and rigorous, because subtle timing errors cause wrong behavior of the whole system.

As one of rigorous verification methods, model checking approach based on temporal logics has been widely studied and applied to verify finite state machines such as protocols or sequential circuits[1, 2]. Relating to asynchronous circuits, formal verification techniques of speed-independent/dependent circuits have been researched[3, 4].

Recently, for the purpose of verifying real-time systems, various timing models have been researched and model checking methods based on the models have been established[5, 7, 8]. The models can treat uncertain delay rigorously. In [7, 6], model checking algorithms based on *continuous-time* model have been proposed and its complexity has also been shown. Although continuous-time model is the most general, its model checking is hard to execute because of its high complexity.

[1]This research is partially supported by Japan-USA cooperative research sponsored by JSPS and NSF.

In this report, we adopt discrete-time model and introduce uncertainty of delays to the framework of model checking. Indeed, discrete-time model is not as rigorous as continuous one, but it will help circuit designers find many errors.

We deal with two types of delay models, i.e, *(uncertain) static delay* and *dynamic delay*. In the static delay model, the delay of a gate is an uncertain integer between a bounded interval, but the delay does not fluctuate at each time unit. The key idea to treat this type of delay is to regard each gate as a *parameterized sequential machine*. For example, a sequential machine $M[d]$ parameterized by a variable d over $\{2,3,4\}$ is used to model a gate with a delay value $d = 2, 3$ or 4, where $M[i]$ $(i = 1,2,3)$ corresponds to the gate with the delay value i. A parameterized sequential machine corresponding to a whole circuit can be constructed from a set of sequential machines associated with gates in the circuit.

In the other model, i.e., dynamic delay model, the delay for each gate can fluctuate at each time unit. Each gate of this type is regarded as a nondeterministic sequential machine. Assume that a delay fluctuates in the range of $\{3,4,5\}$. Then the gate is represented by a consecutive five one-bit registers. The next value for a register is chosen nondeterministically, depending on which delay value is chosen at the time unit.

The size of the sequential machine representing the whole circuits increases exponentially in the number of elements. Symbolic model checking using BDD (Binary Decision Diagram) is a recently developed model checking method and has succeeded in verifying large sequential circuits[2, 9]. In this report, a symbolic model checking algorithm for Branching Time Regular Temporal Logic (BRTL) [10] is shown.

BRTL has, as its temporal operators, deterministic finite automata whose edges are labeled by BRTL formulas. Its expressive power has been proved stronger than CTL.

When we apply model checking to parameterized sequential machines, what is obtained as a result is all value assignments to parameters such that the given specification is satisfied. This means that, in the static delay model we handle here, we can obtain all the possible combinations of delay values such that the specification is satisfied.

Furthermore, BRTL is also extended to express ambiguity of temporal properties by *parameterizing* its temporal operators. This extension means that we can describe a specification with some ambiguity.

In the following, Section 2 explains modeling of static delay and dynamic delay, and overviews formal verification of asynchronous circuits. Section 3 shows definitions of BRTL and symbolic model checking for BRTL. Section 4 reports some experimental results.

2 Verification of Asynchronous Circuits: Overview

2.1 Modeling of Static Delays

A parameterized sequential machine is regarded as a function $M : I \rightarrow \mathcal{M}$, where I is a finite interval of integers and $\mathcal{M}$ is a set of all sequential machines, i.e. $M[c]$ $(c \in I)$ represents a sequential machine. Parameterized sequential machines with two or more parameters are defined similarly.

Since symbolic model checking based on manipulation of logic functions is introduced in Section 3, we describe parameterized sequential machines by transition relation functions, which represent sets of its transition edges, where a distinguished Boolean vector is assigned to each of $c \in I$.

In the following, Boolean variables $s_1, s_2, \cdots$ and $s'_1, s'_2, \cdots$ are used to express initial nodes and terminal nodes of transition edges respectively. i is an input variable.

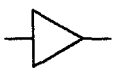


Figure 1: Delay Element

Static delay model is used in [11][2] and it has been shown that the model achieves higher accuracy in logic simulation, comparing with *ambiguous delay model* [12]. Let us consider a delay element shown in Figure 1. $d[1,4]$ means that its delay value is either of 1,2,3 or 4. In this model, it is assumed that the delay value does not change at each time unit, that is, the delay value of a gate is uncertain in its domain but it does not fluctuate.

The transition relation function of the element in Figure 1 is the conjunction of the following logic functions (1) - (4) . By introducing Boolean variables d_1 and d_0, we associate $d = 1, 2, 3$ and 4 with $\neg d_1 \wedge \neg d_0$, $\neg d_1 \wedge d_0$, $d_1 \wedge \neg d_0$ and $d_1 \wedge d_0$ respectively,

(1) $(\neg d_1 \wedge \neg d_0 \wedge (s'_4 \equiv i)) \vee (\neg d_1 \wedge d_0 \wedge (s'_4 \equiv s_1)) \vee (d_1 \wedge \neg d_0 \wedge (s'_4 \equiv s_2)) \vee (d_1 \wedge d_0 \wedge (s'_4 \equiv s_3))$,

(2) $s'_3 \equiv s_2$, (3) $s'_2 \equiv s_1$, (4) $s'_1 \equiv i$.

If $d = 2$, i.e. $d_1 = 0$ and $d_0 = 1$, then the above function represents an element with delay value 2.

The output function of the element is s_4 for $d = 1, 2, 3$ or 4.

2.2 Modeling of Dynamic Delays

Dynamic delay model is another modeling of delays bounded by minimum and maximum values. In this model, $d[1,3]$ means that the delay value can fluctuate over $\{1, 2, 3\}$. This model contains a complicated problem observed in rise/fall delay model[12] as well. Assume that a pulse $0 \rightarrow 1 \rightarrow 0$ of width 2 arrives at the input of the delay element. If delay value 3 is chosen at the first transition of the signals and delay value 1 is chosen at the second transition, a signal change at the output which is caused by the first transition has to be suppressed.

$d[1,3]$ is modeled by the following function.

$$\begin{array}{ll} ((s'_3 \equiv i) \wedge (s'_2 \equiv i) \wedge (s'_1 \equiv i)) & \vee \\ ((s'_3 \equiv s_2) \wedge ((s'_2 \equiv i) \wedge (s'_1 \equiv i)) & \vee \\ ((s'_3 \equiv s_2) \wedge (s'_2 \equiv s_1) \wedge (s'_1 \equiv i)) & \end{array}$$

The first term $((s'_3 \equiv i) \wedge (s'_2 \equiv i) \wedge (s'_1 \equiv i))$ corresponds to the case that the delay value is chosen to be 1. $(s'_2 \equiv i) \wedge (s'_1 \equiv i)$ means that the values stored previously in s'_1 and s'_2 are suppressed by i.

2.3 Specification in BRTL

BRTL is a branching time temporal logic which has deterministic ω finite automata as its temporal operators. Its formal definitions are described in the next section. In this section, an example of BRTL description is shown.

[2] In [11], this type of delay is referred to as *uncertain delay*.

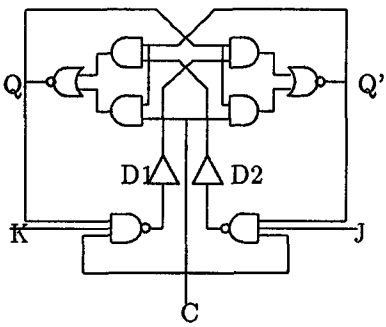


Figure 2: JK flip-flop

Figure 2 shows a design of JK flip-flop. A partial specification for the circuit is shown in Figure 3 (a). *Spec* is a BRTL formula which contains deterministic ω finite automata shown in Figure 3 (b), which are called automata connectives in the next section.

The automata accept infinite sequences composed from assignments of *true* or *false* to atomic propositions. Single circles and double circles mean rejecting states and accepting states respectively. (In BRTL, distribution of accepting or rejecting states are restricted as shown in Section 3.) If an input sequence hits some accepting states infinitely often, the sequence is accepted. Otherwise, it is rejected.

In the specification shown here, signal names in the circuit are also used as atomic propositions of BRTL. "Signal j = 1" corresponds to "j is true". Pulse(x) of Figure 3 expresses that x is true during the first four time units and x stays false after the fifth time unit. Always(x) expresses that x is always true and Fall(x) expresses that after the interval of length greater than or equal to 0 where x is true, x stays false permanently. Intuitively, *Spec* expresses the timing chart shown in Figure 4.

2.4 Verification

In this section, an overview of verification procedure for a logic circuit is shown.

Firstly a parameterized Kripke structure is constructed from the circuit. Kripke structure is defined formally in Section 3. We can perform this step ***symbolically***, i.e. through manipulation of logic functions. More precisely, the conjunction of transition relation functions corresponding to gates in the circuit comes to represent the Kripke structure reflecting the behavior of the circuit.

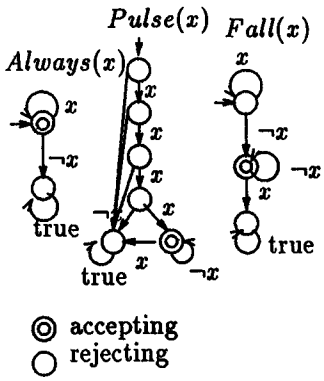
Secondly model checking is performed for the Kripke structure and a given BRTL formula. If some parameters are used in the model, we can obtain all of the assignments of integers to the parameters such that the given specification is satisfied. That is, we can obtain all combinations of delay values such that the circuit works properly. If no variable is introduced for parameterization, then the result of model checking is yes/no.

A symbolic model checking algorithm for this step is shown in Section 3.

BRTL of this paper can also express ambiguity in specification by parameterizing automata in its formulas with variables. For example, Pulse$[d](x)$ in Figure 5 means that d is the length of the first interval where x is true. The domain of d is a bounded interval of integers $\{3,4,5,6\}$. The result of model checking for this specification contains the information about d such that the specification is satisfied.

$$Spec = \forall((Pulse(c) \wedge Always(\neg j \wedge k)) \Rightarrow Fall(q)$$

(a) Specification



(b) Automata connectives

Figure 3: Specification for JK flip-flop (1)

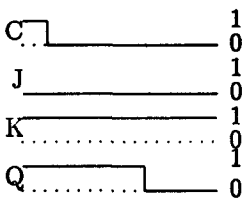
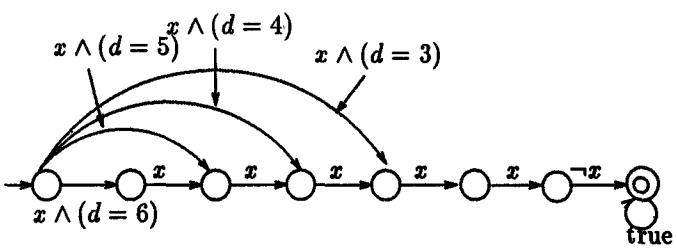


Figure 4: Timing Chart for JK flip-flop

$$Spec = \forall((Pulse[d](c) \wedge Always(\neg j \wedge k)) \Rightarrow Fall(q)$$

(a) Specification



Pulse[d](x)

(b) Automaton connective

Figure 5: Specification for JK flip-flop (2)

3 Branching Time Regular Temporal Logic and Its Symbolic Model Checking

3.1 Branching Time Regular Temporal Logic

The definition of BRTL in this report is different from that of [10]. Firstly, it is *parameterized* and, secondly, Boolean operations of automata connectives in the scope of a path quantifier are allowed.

$I_i = \{a_{i1}, a_{i2}, \cdots, a_{in_i}\}$ $(i = 1, 2, \cdots, w)$ is called a finite interval, where a_{ij} are integers and $a_{ij+1} = a_{ij} + 1$. $D = (d_1, \cdots, d_w)$ is called a variable list, where d_i $(i = 1, 2, \cdots, w)$ is a variable over I_i. D is fixed in the following section. $I \stackrel{\text{def}}{=} I_1 \times I_2 \times \cdots \times I_w$ is the domain of D. $c \in I$ is called an *environment*. V_T represents *tautology*.

Definition 1 *A deterministic ω finite automaton type 1 (dfa-1)*

$A[D] = (Q, P, Br, q_0, F)$ is defined as follows.

Q is a set of finite number of states and $P = \{p_1, \cdots, p_n\}$ is a set of propositional variables. q_0 is the initial state. F is a set of accepting states and the elements of $Q - F$ are called rejecting states. Let BF be a set of all propositional formulas constructed from the elements of P. $Br : Q \times Q \times I \rightarrow BF$ is a partial function which satisfies the following three conditions.

Consider $Br(q, Q, c) = \{f | \exists q'.Br(q, q', c) = f\}$, for $q \in Q$ and $c \in I$.

(1) $f_1 \wedge f_2 = V_F$, for any $f_1, f_2 \in Br(q, Q, c)$.

(2) $\bigvee_{f \in Br(q,Q,c)} f = V_T$.

$A[D]$ accepts or rejects infinite sequences composed from the elements of $\Delta = 2^P$ under a given environment c. The transition function of $A[D]$ under c is defined to be $\delta_c : Q \times \Delta \rightarrow Q$ such that $\delta_c(q, v) = q' \Leftrightarrow Br(q, q', c)(v) = T$, where $v \in \Delta$

$A[D]$ under an environment c is described as $A[c]$. In $A[c]$, the third argument of Br is fixed to c.

For $\sigma \in \Sigma^\omega$, $\mathrm{Inf}(\sigma)$ is defined to be the set of the states through which $A[c]$ goes infinitely often. Then the set of words accepted by $A[c]$ is $\{\sigma | \mathrm{Inf}(\sigma) \cap F \text{ is not empty }\}$ and is described by $\langle A \rangle_c$.

(3) Under any environment c, there exists no path from a rejecting state q_r to q_r itself via some accepting state q_a. □

Lemma 1 For a dfa-1 $A[D] = (Q, P, Br, q_0, F)$, a dfa-1 $\overline{A[D]}$ which accepts $\Sigma^\omega - \langle A \rangle_c$ for each environment c is obtained by exchanging accepting states and rejecting states of $A[D]$.

Lemma 2 For dfa-1's $A_i[D] = (Q_i, \Sigma, P, Br_i, \delta_i, q_{i0}, F_i)$ $(i = 1, 2)$, a dfa-1 $A_1[D] | A_2[D]$ which accepts $\langle A_1 \rangle_c \cup \langle A_2 \rangle_c$ for each c is obtained as follows:

$Q = Q_1 \times Q_2 = \{(q_1, q_2) | q_1 \in Q_1 \ , \ q_2 \in Q_2\}$, $q_0 = (q_{10}, q_{20})$ and $F = \{(q_1, q_2) | q_1 \in F_1$ or $q_2 \in F_2\}$. $Br : Q \times Q \rightarrow BF$ is defined by $Br((q_1, q_2), (q_1', q_2'), c) = Br_1(q_1, q_1', c) \wedge Br_1(q_2, q_2', c)$, where $q_i, q_i' \in Q_i$ $(i = 1, 2)$ and $c = c(D)$, and both of $Br_1(q_1, q_1', c)$ and $Br_1(q_2, q_2', c)$ are defined. □

Definition 2 $S[D] = \langle \Sigma, \mathrm{As}, R, \Sigma_0 \rangle$ is called a Kripke structure, where Σ is a set of nodes, $\mathrm{As} : \Sigma \rightarrow 2^{AP}$ is an assignment function, $R \subseteq \Sigma \times \Sigma \times I$ is a *total* relation for each $c \in I$, i.e, there exists $s' \in \Sigma$ such $(s, s', c) \in R$ for any $s \in \Sigma$. $\Sigma_0 \subseteq \Sigma$ is a set of initial states.

$S[D]$ under an environment c, denoted by $S[c]$, is a Kripke structure such that R is fixed by c. □

Definition 3 Syntax and Semantics

The syntax of BRTL is as follows:

BRTL formulas:

Let p be an atomic proposition in AP, ψ and ϕ be BRTL formulas, and B be an automaton connective. Then p, $\neg\psi$ and $\psi \vee \phi$ and $\exists B$ are BRTL formulas.

Automata connectives

Let $A[D]$ be a dfa-1 and B, B_1, B_2 be automata connectives. Then $A[D](\psi_1, \psi_2, \cdots, \psi_n)$, $\neg B$ and $B_1 \vee B_2$ are also automata connectives.

$A[D](\psi_1, \psi_2, \cdots, \psi_n)$ is obtained by replacing each atomic proposition $p_i \in P$ $(1 \leq i \leq n)$ with $\psi_i \subseteq BF$ corresponding to p_i simultaneously.

The semantics of BRTL is defined on a Kripke structure $S[D] = \langle \Sigma, I, R, \Sigma_0 \rangle$ under a given environment c. $S[D], s \models_c f$ means that the BRTL formula f holds at the state s on $S[D]$ under c. In the following, $p \in AP$, ψ and ϕ are BRTL formulas and $A[D]$ is a dfa-1 and B is an automaton connective. Trans(B) represents an automaton connective obtained by applying Lemma 1 and 2 to B until all of $\neg$ and $\vee$ in B are deleted.

- $S[D], s \models_c p$ iff $p \in \mathrm{As}(s)$
- $S[D], s \models_c \psi \vee \phi$ iff $S[D], s \models_c \psi$ or $S[D], s \models_c \phi$
- $S[D], s \models_c \neg\psi$ iff $S[D], s \not\models_c \psi$
- $S[D], s \models_c \exists A[D](\psi_1, \psi_2, \cdots, \psi_n)$ iff there exists an infinite sequence $\sigma = s_0 s_1 s_2 \cdots$ starting from s on $S[c]$ and a run (a sequence of states) $q_0 q_1 q_2 \cdots$ in Q such that, $S[D], s_i \models_c Br(q_i, q_{i+1}, c)$ holds and at least one state $q \in Q$ which appears infinitely in the run is in F.
- $S[D], s \models_c \exists B$ iff $S[D], s \models_c \exists(\mathrm{Trans}(B))$

If $\forall s \in \Sigma_0 . S[D], s \models_c f$, then we describe $S[D] \models_c f$. □

The Boolean operators $\wedge$, $\equiv$ and $\Rightarrow$ are also used. Besides we define $\forall B \stackrel{\mathrm{def}}{=} \neg\exists\neg B$ and $\forall\neg B \stackrel{\mathrm{def}}{=} \neg\exists B$.

3.2 Symbolic Model Checking of BRTL

The symbolic model checking technique shown in this section is an extension of the methods found in [2]. The difference is that parameters have to be handled.

Definition 4 Model checking problem

Given a Kripke structure $S[D]$ and a BRTL formula ψ, model checking problem is to obtain a set of environments c under which $S[D] \models_c \psi$ holds. □

In the following, a unique code composed from $B = \{0, 1\}$ is assigned to each node on the Kripke structure. For a node s, cd(s) represents the code assigned to s. Assume that n bit of vector is required to represent $s \in \Sigma$. $\vec{s} = s_1, s_2, \cdots, s_n$ and $\vec{s'} = s'_1, s'_2, \cdots, s'_n$ are defined to be variable vectors over B^n.

Each $c \in I$ is also encoded by B. For each $c_i \in I_i$, cd(c_i) represents the code assigned to c_i and cd(c) is the concatenation from cd(c_1) to cd(c_w). If n_i bit is required to encode c_i, then $\vec{c_i}$ is a variable vector over B^{n_i} and $\vec{c}$ is a variable vector over $B^{\sum_i n_i}$.

The Kripke structure is represented by logic functions:

1. Transition relation: $f_S(\mathrm{cd}(s), \mathrm{cd}(s'), \mathrm{cd}(c)) = 1$ iff $(s, s', c) \in R$.
2. Assignment of atomic propositions: $f_p(\mathrm{cd}(s)) = 1$ iff $p \in \mathrm{As}(s)$.

3. Initial states: $f_{init}(cd(s)) = 1$ iff $s \in \Sigma_0$

Besides the above vector variables, $\vec{q_j}$ and $\vec{q_j'}$ are used for each dfa-1 $A_j[D]$ in the BRTL formula ψ. The code for a state q_j in $A_j[D]$ is represented by $\mathrm{cd}(q_j)$.

Algorithm 1 model checking algorithm

- Input: BRTL formula ψ and the above f_S, f_p and f_{init}.
- Output: The logic function $f_{env}(\vec{c})$ such that $f_{env}(\mathrm{cd}(c)) = 1$ iff $S[D] \models_c \psi$.
- Method: Shown below.

1. For each dfa-1 $A_j[D]$ in ψ, the following logic functions are constructed:
   - For each $(q_j, q_j', c) \in Q_j \times Q_j \times I$, $f_{E_j(\psi_k)}(\mathrm{cd}(q_j), \mathrm{cd}(q_j'), \mathrm{cd}(c)) = 1$ $\Leftrightarrow Br(q_j, q_j', c) = \psi_k$.
   - q_j is an accepting state of $A_j \Leftrightarrow f_{F_j}(\mathrm{cd}(q_j)) = 1$
2. For each subformula ϕ_i of ψ, a logic function $f_{\phi_i}(\vec{s}, \vec{c})$ which represents $S[D], s \models_c \phi_i$ is constructed in bottom up manner. Let $\theta, \theta_1, \theta_2$ are BRTL formulas.
   (a) If ϕ_i is an atomic proposition, f_{ϕ_i} is returned.
   (b) If ϕ_i is $\theta_1 \vee \theta_2$ or $\neg\theta$, then $f_{\theta_1 \vee \theta_2} = f_{\theta_1} \vee f_{\theta_2}$ and $f_{\neg\theta} = \neg f_\theta$ are returned.
   (c) If ϕ_i is $\exists B$, then the following i.– vi. are performed.
      i. For each $A_j[D](\psi_{j_1}, \psi_{j_2}, \cdots, \psi_{j_{n_j}})$ $(j = 1, 2, \cdots, m)$, vectors of variables representing the states of A_j, $\vec{q_j}$ and $\vec{q_j'}$ are introduced. Let $\vec{q} = \vec{q_1}\#\vec{q_2}\#\cdots\#\vec{q_m}$ and $\vec{q'} = \vec{q_1'}\#\vec{q_2'}\#\cdots\#\vec{q_m'}$, where $\#$ means concatenation of vectors. Let Q be defined as $Q_1 \times Q_2 \times \cdots Q_m$. $\vec{v} \stackrel{\mathrm{def}}{=} \vec{s}\#\vec{q}$ and $\vec{v'} \stackrel{\mathrm{def}}{=} \vec{s'}\#\vec{q'}$.
      ii. A logic function $f_E(\vec{v}, \vec{v'}, \vec{c})$ is constructed, which represents a set of edges of the graph G consisted of the nodes $\Sigma \times Q$.
      $f_E(\vec{v}, \vec{v'}, \vec{c}) \stackrel{\mathrm{def}}{=} \bigwedge_{j=1,2,\cdots,m}\{\bigvee_{k=1,2,\cdots,n_j}(f_{E_j(\psi_{jk})}(\vec{q_j}, \vec{q_j'}, \vec{c}) \wedge f_{\psi_{jk}}(\vec{s}, \vec{c})\} \wedge f_S(\vec{s}, \vec{s'}, \vec{c})$
      iii. A logic function $f_{F_B}(\vec{q})$ is constructed, which represents the set of accepting states of Trans(B). It is calculated recursively as follows:
         - If $B = A_j[D](\psi_{j_1}, \psi_{j_2}, \cdots, \psi_{j_{n_j}})$, then $f_{F_B}(\vec{q}) \stackrel{\mathrm{def}}{=} f_{F_j}(\vec{q_j})$.
         - If $B = \neg B_1$, then $f_{F_B}(\vec{q}) \stackrel{\mathrm{def}}{=} \neg f_{F_{B_1}}(\vec{q})$.
         - If $B = B_1 \vee B_2$, then $f_{F_B}(\vec{q}) \stackrel{\mathrm{def}}{=} f_{F_{B_1}}(\vec{q}) \vee f_{F_{B_2}}(\vec{q})$.
      iv. Let V' be a set of nodes $\{(s, q) | f_{F_B}(\mathrm{cd}(q)) = 1\}$ and G' be a restriction of G to V'. A logic function $f_{C'}(\vec{v}, \vec{c})$ is constructed, which represents the set of nodes which are in strongly connected components of G' or are reachable to the strongly connected components.
      $f_{E'}(\vec{v}, \vec{v'}, \vec{c}) \stackrel{\mathrm{def}}{=} f_{F_B}(\vec{q}) \wedge f_{F_B}(\vec{q'}) \wedge f_E(\vec{v}, \vec{v'}, \vec{c})$
      Set $f^0_{C'}(\vec{v}, \vec{c}) \stackrel{\mathrm{def}}{=} f_{F_B}(\vec{q}, \vec{c})$. $f^1_{C'}, \cdots f^k_{C'}$ are calculated until $f^{k+1}_{C'} \equiv f^k_{C'}$ holds.
      $$f^{i+1}_{C'}(\vec{v}, \vec{c}) \stackrel{\mathrm{def}}{=} (\exists \vec{v'}.(f_{E'}(\vec{v}, \vec{v'}, \vec{c}) \wedge f^i_{C'}(\vec{v'}, \vec{c}))) \wedge f^i_{C'}(\vec{v}, \vec{c})$$
      $$f_{C'}(\vec{v}, \vec{c}) \stackrel{\mathrm{def}}{=} f^k_{C'}(\vec{v}, \vec{c})$$
      v. A logic function f_R is constructed, which represents the set of nodes which are in G and reachable to $V_{C'}$. Set $f^0_R(\vec{v}, \vec{c}) \stackrel{\mathrm{def}}{=} f_{C'}(\vec{v}, \vec{c})$. $f^1_R, f^2_R, \cdots f^k_R$ are calculated until $f^{k+1}_R \equiv f^k_R$ holds.

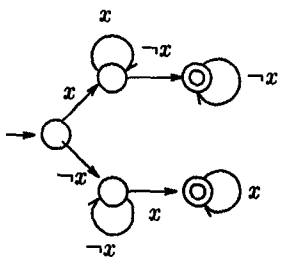


Figure 6: Automaton connective: Change(x)

$$f_R^{i+1}(\vec{v},\vec{c}) \stackrel{\text{def}}{=} (\exists \vec{v'}.(f_E(\vec{v},\vec{v'},\vec{c}) \wedge f_R^i(\vec{v'},\vec{c}))) \vee f_R^i(\vec{v},\vec{c})$$
$$f_R(\vec{v},\vec{c}) \stackrel{\text{def}}{=} f_R^k(\vec{v},\vec{c})$$

vi. $f_{\psi_i}(\vec{s},\vec{c}) \stackrel{\text{def}}{=} \exists \vec{q}.(f_R(\vec{s},\vec{q},\vec{c}) \wedge f_I(\vec{q}))$, where $f_I(\vec{q})$ represents the initial state of B.

(d) $(\bigvee_{c\in I} f_{cd}(c)) \wedge \forall \vec{s}.(f_{init}(\vec{s}) \Rightarrow f_\psi(\vec{s},\vec{c}))$ is returned, where $f_{cd}(c)$ represents a logic formula (minterm) representing $cd(c)$.

4 Experimental Results

A prototype verifier based on the above methods was implemented. The verifier is written in language C and runs on a SPARC 1+ workstation. This program utilizes the Boolean function manipulator developed by Minato et.al. [13], which uses SBDD (Shared Binary Decision Diagram) representation as its internal representation. SBDD is an improvement of the binary decision diagram[14], which shares all possible subgraphs among multiple functions. There are many advantages besides those of the BDD. For example, equivalence of two functions can be checked only by comparing the pointers.

For the purpose of efficient manipulation of SBDD, the whole transition relation functions are not generated. Each of logic functions corresponding to the gates in the circuits is handled separately as shown in [15].

Each element in the circuit is initialized to an arbitrary stable value, that is, the set of all stable states of the circuits is used as the set of initial states for the verifier. The verified conditions are f_1 for sjk4, sjk5, mjk1, mjk2 and mjk3, and f_2 for the others. ($Change(x)$ is shown in Figure 6.) The length of the consecutive x in $Pulse(C)$ is adjusted for each example. For all the examples, SBDD size is limited to 500,000.

$$f_1 = \forall((Pulse(C) \wedge Always(\neg J \wedge K)) \Rightarrow (Fall(Q))) \quad (1)$$
$$f_2 = \forall((Pulse(C) \wedge Always(\neg J \wedge K)) \Rightarrow (Fall(Q) \wedge Fall(\neg q'))) \wedge$$
$$\forall((Pulse(C) \wedge Always(J \wedge K)) \Rightarrow (Change(Q) \wedge Change(Q')) \quad (2)$$

The verification results for the circuits in Figure5 are shown in Table 1. Several combinations of the delay values were checked.

Static delay model was used for sjk's and dynamic delay model for djk's. mjk's mean that, for example, the delays [4, 12] for D1 and D2 were divided into eight dynamic delays $[4,5],[5,6],\cdots,[11,12]$, and each of the delays was associated with a Boolean vector. By introducing Boolean variables for them, the verifier checked which fluctuation among $[4,5],[5,6],\cdots,[11,12]$ satisfies the given condition.

The meaning of each column is also shown in Table 1. For example, the row sjk1

Table 1: Experimental Results

	#OP	time	#var	dl1	dl2	v/i
sjk1	34	62	21	[1,5]	[1,1]	∘
sjk1*	36	64	21	[1,5]	[1,1]	∘
sjk2	42	828	33	[7,8]	[2,2]	∘
sjk3	34	1784	27	[0,0]	[1,3]	×
sjk4	15	2118	33	[1,7]	[1,2]	∘
sjk5	23	—	43	[1,8]	[1,3]	—
djk1	34	34	19	[1,5]	[1,1]	×
djk1$^+$	38	179	24	[9,10]	[1,1]	∘
djk2	42	907	43	[11,12]	[1,2]	∘
djk3	58	5658	59	[15,16]	[2,3]	×
mjk1	17	584	43	$\langle 4,12\rangle$	[1,2]	∘
mjk2	22	8777	59	$\langle 8,16\rangle$	[2,3]	×
mjk3	26	—	67	$\langle 17,20\rangle$	[2,3]	—

#OP: Number of states in BRTL automata connectives

time CPU time (seconds)

#var: Number of input and internal Boolean variables

dl1: Delays for $D1$ and $D2$

dl2: Delays for the rest of gates

v/i: ∘ (resp., ×) means that the verifier found a (resp., no) combination such that the given condition is satisfied. If there is no parameters, ∘ (resp., ×) means yes (resp., no).

means that static delays d1[1,5] and d2[1,5] were assigned to the delay elements $D1$ and $D2$. The delays of the rest of the gates were fixed to 1. '—' in the column means that the verifier failed to finish verification under the limit of SBDD size.

In sjk1*, the length of the consecutive x in $Pulse(x)$ was parameterized. The result shows that at least four 1's is necessary for the required behavior.

The results for djk1 and djk1$^+$ show that, under dynamic delay model, the delays of D1 and D2 have to be longer than under static delay model.

The experimental results show that the cost of verification is very sensitive to the increase of the delay values. Especially for static delay model, it is hard to handle a large amount of uncertainty, though in sjk4, $7^2 \times 2^8$ possible combinations were checked.

5 Conclusion

In this report, we have considered formal verification method of speed-dependent asynchronous circuits which have, in particular, gates with uncertain delays. We have shown how to represent delays of gates by parameterized sequential machines or by nondeterministic sequential machines. When we use parameters, we can obtain all the assignments to parameters such that a given specification is satisfied.By using symbolic model checking method, some experimental results were shown.

One of interesting future works is to extend the timing model to treat continuous quality. In [7, 6], it has been shown that the model checking for TCTL on continuous-time model can be solved decidably and the problem can be reduced to a certain problem over finite graphs. This suggests that it would be possible to handle the continuous-time

model by symbolic model checking. When we try to use BDD for the verification based on continuous-time model, however, much more efficient methods will be required.

Acknowledgements
The authors would like to thank anonymous referees for helpful comments.

References

[1] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic Verification of Finite State Concurrent Systems Using Temporal Logic Specifications: A Practical Approach. In *10th ACM Symposium on Principles of Programming Languages*, pages 117–126, January 1983.

[2] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and J. Hwang. Symbolic Model Checking: 10^{20} States and Beyond. In *Proceedings of 5th IEEE Symposium on Logic in Computer Science*, June 1990.

[3] M. C. Browne, E. M. Clarke, D. L. Dill, and B. Mishra. Automatic Verification of Sequential Circuits Using Temporal Logic. *IEEE Transactions on Computers*, C-35(12):1035–1044, December 1986.

[4] D. L. Dill and E. M. Clarke. Automatic Verification of Asynchronous Circuits Using Temporal Logic. *IEE Proceedings*, 133:276–282, September 1986.

[5] R. Alur and D. Dill. Automata for Modeling Real-Time Systems. In *Proceedings of ICALP 90'*, 1990.

[6] R. Alur, C. Courcoubetis, and D. Dill. Model-Checking for Real-Time Systems. In *Proceedings of 5th IEEE Symposium on Logic in Computer Science*, pages 414–425, June 1990.

[7] H. R. Lewis. A Logic of Concrete Time Intervals. In *Proceedings of 5th IEEE Symposium on Logic in Computer Science*, pages 380–389, 1990.

[8] J.S. Ostroff. Automated Verification of Timed Transition Models. *Automated Verification Methods for Finite State Systems*, pages 247–256, 1989.

[9] O. Coudert, C. Berthet, and J-C. Madre. Verification of Sequential Machines Using Functional Vectors. *Proceedings of IMEC-IFIP International Workshop on Applied Formal Methods for Correct VLSI Design*, pages 111–128, November 1989.

[10] K. Hamaguchi and H. Hiraishi and S. Yajima. Branching Time Regular Temporal Logic for Model Checking with Linear Time Complexity. *Workshop on Computer-Aided Verification*, June 1990.

[11] Nagisa Ishiura, Yutaka Deguchi, and Shuzo Yajima. Coded Time-Symbolic Simulation Using Shared Binary Decision Diagram. *Proceedings of 27th Design Automation Conference*, pages 130–135, 1990.

[12] M. Abramovici, M. A. Breuer, and A. D. Freidman. *Digital Systems Testing and Testable Design.* Computer Science Press, 1990.

[13] Shin ichi Minato, Nagisa Ishiura, and Shuzo Yajima. Shared Binary Decision Diagram with Attributed Edges for Efficient Boolean Function Manipulation. *Proceedings of 27th Design Automation Conference*, pages 52–57, 1990.

[14] R. E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[15] J. R. Burch, E. M. Clarke, and D. E. Long. Representing Circuits More Efficiently in Symbolic Model Checking. *Proceedings of 28th Design Automation Conference*, June 1991.