

Timing Extensions of STG Model and a Method to Simulate Timed STG Behavior in VHDL Environment

Michael V. Goncharov
Alexander B. Smirnov

Ilya V. Klotchkov
Nikolai A. Starodoubtsev

Institute for Analytical Instrumentation of RAS
26, Rizhsky Prospect, Saint-Petersburg 198103 Russia

Abstract

This paper includes an overview of the Signal Transition Graph (STG) model extensions that makes it possible to specify switching and signal propagation delays in an STG. The correspondence of the STG timing models to the implementation by asynchronous circuit is considered.

A method to simulate the behavior specified by consistent and bounded timed STG in VHDL environment is proposed. For illustration we present a possible use of the VHDL-based STG representation in asynchronous circuits design.

1 Introduction.

Concurrency in system design becomes more important as VLSI components scale down and their speed is increased. Asynchronous design does not require any clock signals and therefore has some potential advantages, since distributing clock across an entire chip without a skew becomes more problematic and synchronization of a number of clocks could be at least as difficult as design of a fully asynchronous system. Protagonists of asynchronous circuits also refer to a potential performance advantages (no need to wait for the clock) and lower power dissipation.

Although most of the advantages of asynchronous circuits are known for decades a typical logic designer would avoid their use in the design practice. One of the reasons is that design tools for asynchronous IC still lack a number of features provided in synchronous CAD tools. Most of the modern research in asynchronous design is devoted to logic synthesis and testing, paying less attention to the tools for high-level synthesis and simulation. In order to provide a designer of asynchronous systems with a wide variety of tools similar to those used for a synchronous design, it

is necessary to connect widely used hardware description languages like VHDL to signal transition graphs (STG) introduced in [1]. The latter are often used today for describing concurrency, causality and choice in asynchronous systems.

This problem was addressed previously in [2, 3]. In [2] a complete methodology for the design and validation of asynchronous circuits starting from STG specification is proposed. There are two major differences of our approach if it is compared with [2]. Firstly the class of STGs we consider is extended to include non-free-choice nets and delays specification. It is the major contribution of the paper and will be discussed in details in section 4 as well as other differences. Secondly in this paper we consider the application of simulation in the framework of semi-modular circuits' design (these circuits are known to operate correctly under unbounded gate delays [4]). At the same time in [2] asynchronous circuits satisfying bounded wire delays model are considered, hence, timing analysis and hazard elimination becomes the necessary part of the design process. As the result our design methodology differs from the one in [2]. In particular, semi-modular circuits are hazard-free under unbounded gate delay model and have a property being very useful for testing: these circuits halt in presence of output and some input stuck-at faults. This encourages us to pay in section 2 some attention to the usage of VHDL environment for faults' simulation. At the same time hazard elimination stage if compared with [2] can be omitted.

In this paper a method is proposed for automatic translation of an STG specification to VHDL. This methodology originates from [3] where the marked indexed VHDL (MIV) form was introduced. Arcs' markings (recalling implicit places in STGs), transition occurrence indexes and causality of events were

used there for modeling STG-like specifications in VHDL. In this paper we mostly describe the simulation of STG in VHDL environment, while [3] is mainly devoted to synthesis of asynchronous circuits using compact and easily readable VHDL-code throughout the design process.

The paper is organized as follows. In Section 2 we give an overview of the whole design methodology of semi-modular circuits based on standard VHDL environment which is typically used for synchronous circuits' design. Then we describe timed extensions of the STG model (Section 3) to provide a variety of ways to specify timed behavior. The use of VHDL for STG specifications, some examples and comments on the simulation process are given in Section 4.

2 Methodology

A brief review of the semi-modular circuits' design methodology is given to show the role of the STG behavioral simulation in the design framework. The main characteristics of the methodology are as follows.

Simulation. A circuit communicates with the environment via two types of signals: signals, which are inputs for the circuit and outputs for the environment, and vice versa, signals, which are inputs for the environment and outputs for the circuit.

Synthesis. For the remainder of the paper details of the synthesis methodology are not relevant, as synthesized circuit is only mentioned here for joint circuit-environment simulation. At the same time STG analysis (as a first stage of the synthesis) is required for identification of properties, which are essential for generation of the correct VHDL code.

Library. Library gates are assumed to be internally hazard-free.

An overview of the design methodology for semi-modular circuits is presented in Figure 1. The shaded boxes represent the data that has to be provided by a designer. The STG model is used for formal behavior specification of the circuit and the environment it operates in. STG formalism and examples are presented in Sections 3 and 4. A designer has to provide a library of internally hazard-free gates. For instance, it must be specified if an inverter can be considered as a part of a complex gate or always as a stand alone logic gate. Finally, a designer should define a fault model for the fault simulation. The issues of delay variations that are introduced by the layout design phase are out of the scope of this paper.

The STG is used as a primary behavior specification, because it is general enough for handling concurrency, conditional behavior and can be loaded with

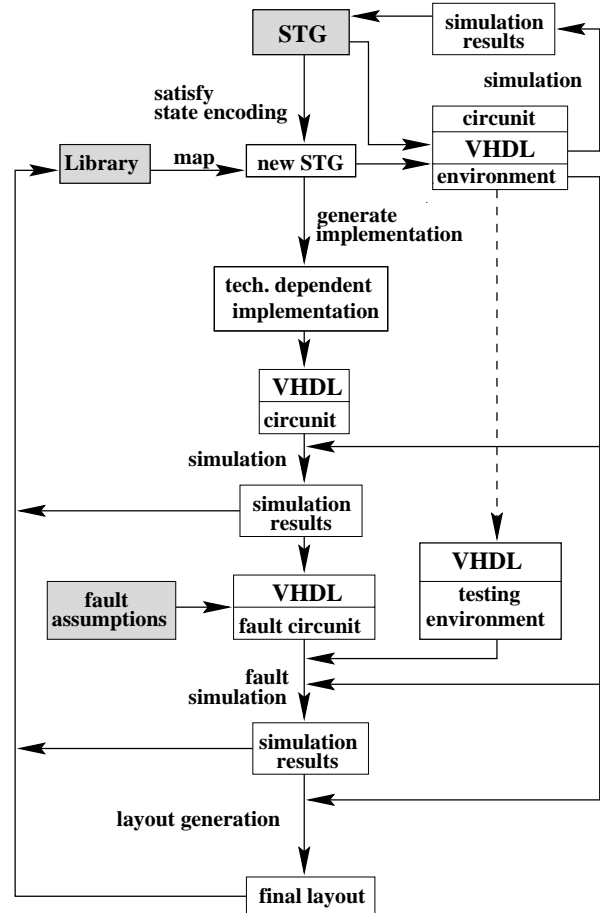


Figure 1: An overview of the design methodology for semi-modular circuits.

timing information. If necessary other specification models can be translated into STG. If the initial STG does not satisfy some of the implementability conditions such as consistency or complete state coding [5, 6], then it could be transformed to a new one. This could be done by introducing internal state signals into the STG, for instance in a way used in [3]. Similar changes can be used in order to transform initial STG to the one implementable within a given gate library. Both of these transformations are represented in Figure 1 by arcs entering the box called “new STG”.

From the initial STG a VHDL-code is generated. Its simulation convinces the designer that the initial specification is correctly represented by STG. The VHDL-code describes both the circuit and the environment as it is discussed in more details in Section 4. VHDL code can be produced for a new STG after its

transformation.

Based on the information in the new STG a technology dependent implementation can be obtained and then represented by a VHDL-code carrying some structural information. Having the previous VHDL-code for the environment and the new VHDL-code for the synthesized circuit we jointly simulate their behavior.

If the circuit simulation is successful one can modify the circuit VHDL-code for representing possible stuck-at faults according to the fault model being in use. A fault simulation can be used to validation that faults are testable. Either hypothetical delay values or real ones obtained on the next stage of design process can be used for simulation.

Finally the layout is generated and layout information is mapped back to the circuit VHDL-code. Again simulation is used for validation of the circuit behavior under real delays. A new layout is generated if the validation fails. If this process fails to converge the designer is enforced to re-iterate starting from a modified version of STG.

3 STG delay extensions.

In this Section we consider circuit delay models, STG model timing extensions and the correspondence between the circuit and the STG delay models. The delay information in STG can be used both for logic synthesis and for simulation of the specified behavior. Here we only consider the circuit simulation.

An STG, extended with delay information later on is called *Timed STG* (TSTG).

The following define Petri Nets and STG more formally.

- *Petri net* (PN) is a 4-tuple $N = (P, T, F, m_0)$, where P is a finite set of places, T is a finite set of transitions, $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation, and m_0 is the initial marking; $V = P \cup T$ stands for a set of all vertices of net N ;
- *STG* is a 4-tuple $\mathcal{G} = (N, X, Z, \Delta)$ where N is a PN, X and Z are disjoint sets of input and output signals respectively and $\Delta : T \rightarrow (X \cup Z) \times \{+, -\}$ labels transitions of N with a signal transition; transitions unlabeled with signal transitions are called *dummy*; $Y = X \cup Z$ denotes a set of all signals in STG;
- $t\bullet$ and $\bullet t$ denote the sets of output and input places for transition t , correspondingly;
- $p\bullet$ and $\bullet p$ denote the sets of output and input transitions for place p , correspondingly;

- for every arc $f_{ij} \in F$ (arc from v_i to v_j) let us denote $\bullet f_{ij} = v_i$ and $f_{ij}\bullet = v_j$ ($v_i, v_j \in V$);
- STG \mathcal{G} is called *autonomous* if for $\forall v_i \in V : |\bullet v_i| > 0$ i.e. if there are no vertices without predecessors in the STG;

3.1 Previous work on STG timing issues.

By definition Petri net underlies every STG. Thus, certain timing issues worked out for Petri nets may be adopted for STG. Let us start to review the previous efforts in the area of time STG from the work on timed Petri nets.

As an introductory work on timed Petri nets the reader may be referred to [7] where the concept of timed PN was discussed and the analysis of this model was presented.

Timing semantics for STG were defined in [8] and [9]. There some analysis and logic synthesis issues have been also considered. The latter work define timed STG binding delay intervals to arcs (places) to constrain the transitions' firing times. Absolute firing times are assigned to transitions based on the constraints specified. Only acyclic deterministic STGs are considered there.

3.2 Circuit assumptions.

In this paper we assume that the circuit with its environment is an autonomous system, i.e. the whole system has no inputs. Nevertheless, further on we use the terms *output signal* and *input signal* for circuit outputs and inputs, correspondingly. The system is a set of *functional elements* (FE) interconnected with *wires*. Every FE's input is connected to exactly one output of some other FE in a circuit or environment. A FE output may be connected to some inputs of FE. No two FEs are connected with their outputs. Each FE is an atomic gate implementing a function of m variables $z_i = Z(z_1, \dots, z_m)$. A FE function may be unknown. This allows a simplified behavior specification.

Each FE has an internal delay, i.e. a delay between changes at the inputs and the corresponding change at the output of the FE. In practice a delay of a FE can vary for rising and falling transitions. Hence, we consider a FE to have *rising* and *falling* delays.

Each *wire* module has one input and one or more outputs, in the latter case it corresponds to a wire fork. Every wire module is associated a set of delays, one delay per output of the wire module. Each delay corresponds to a propagation delay of a signal traveling from the input of the wire module to a particular output.

3.3 Circuit delay models.

In general there are three primary circuit delay models.

Wire delay model: FEs have zero or negligible delays. Examples: *delay insensitive (DI) model* assumes circuit wires to have unbounded delays; *bounded wire delay model* assumes circuit wires to have bounded delays.

Gate delay model: a delay is associated with every FE. Wires are supposed to have zero or negligible delays. Examples: *speed independent (SI) model* (including semi-modular circuits) assumes gates to have unbounded delays and *bounded gate delay model* - gates are assumed to have bounded delays. This model can be reduced to the *wire delay model* by assigning every gate delay value to its output wire and assuming gates to have zero delays.

Wire - gate delay model.

The first two models are simple and irredundant. On the contrary, the wire - gate delay model is more general and redundant. Indeed, since the gate delay model can be reduced to the wire delay model, the wire - gate delay model can be reduced to it as well. However, the wire - gate delay model can be more convenient and natural for practical use.

3.4 STG delay models.

STG delay models can be inherited from the corresponding delay models that are used in the PN theory: *place timed PN* and *transition timed PN*. *Place timed PNs* have delay values assigned to the places and the *transition timed PNs* have delay values assigned to transitions [7]. Correspondingly in Sections 3.4.1 and 3.4.2 we define *transition timed STG* and *place timed STG*. In Section 3.4.3 we introduce the *arc timed STG* and some redundant STG delay models that can be obtained by combining the primary models. The latter are introduced for simplifying behavior specifications. All STG delay models are compared with the circuit delay models.

Let us give a few definitions describing TSTG.

- A *path* R_{ij} from v_i to v_j ($v_i, v_j \in V$) in STG \mathcal{G} is defined as a two-tuple $R_{ij} = (F', V') : F' \subset F, V' \subset V$ so that 1) $v_i \notin V' \ \& \ v_j \notin V'$ 2) $\forall f_{ij} \in F' : f_{ij} \bullet \in (V' \cup \{v_j\}) \ \& \ \bullet f_{ij} \in (V' \cup \{v_i\})$; 3) $\forall v_k \in V' : (|v_k \bullet \cap (V' \cup \{v_j\})| = 1) \ \& \ (|\bullet v_k \cap (V' \cup \{v_i\})| = 1)$;
- Let $D(f_{ij})$ ($D(v_i)$) be a delay associated with STG arc $f_{ij} \in F$ (vertex $v_i \in V$);
- A path $R_{ij} = (V', F')$ in a TSTG is said to have a *path delay* $D(R_{ij}) = \sum_{k=0}^{r-1} D(v_k) + \sum_{k=0}^{p-1} D(f_k)$

where $\forall v_k \in V', \forall f_k \in F', r = |V'|, p = |F'|$, i.e. equal to a total delay of all elements in R_{ij} depending on the STG delay model certain elements will have zero delays;

- A transition t_j *directly depends* on another transition t_i ($t_i \longrightarrow t_j$) if $\exists R_{ij} = (F', V')$, for $\forall t_k \in V'$ and $t_k \in T$ t_k is a *dummy* transition, i.e. there exists a path R_{ij} from t_i to t_j in the STG containing no signal transitions;
- A path R_{ij} from some transition t_i to another transition t_j in an STG is said to be a *direct path* if $t_i \longrightarrow t_j$, i.e. t_j directly depends on t_i ;
- A direct path R_{ij} from transition t_i to another transition t_j in an STG is said to be a *redundant dependency* if 1) $\exists R'_{ij} \neq R_{ij}$ such as R'_{ij} is not a direct path, i.e. there exists another path from t_i to t_j that is not a direct path;
- A place in an STG is called an *implicit place* if it has one input and one output arc. Implicit places are usually not shown in the STG drawings, but are represented by the corresponding arc between two transitions.
- An STG $\mathcal{G} = (P, T, F, m_0)$ is called *free-choice* if $\forall p_i \in P$ such that $|p_i \bullet| > 1$ for $\forall t_j \in p_i \bullet : | \bullet t_j | = 1$, i.e. for every place with multiple outputs all directly succeeding transitions have a single input.

3.4.1 Transition timed STG. In the *transition timed STG* (TTSTG) model the same delay is assigned to every falling (rising) transition of the same signal. Dummy transitions, arcs and places are assigned zero delays.

The TTSTG delay model defined this way does not allow to specify the behavior where two different rising (or falling) transitions of the same signal actually have different delays (other TSTG model can be used to overcome this limitation). On the other hand, this simplified model corresponds to a simplified model of a circuit gate delay when every FE has only one rising and one falling delay.

TTSTG can be reduced to the place timed STG described below by assigning a transition delay $D(t_i)$ to every output place of a transition ($\forall p_k \in t_i \bullet : D(p_k) = D(t_i)$) and assigning zero delays to transitions ($D(t_i) = 0$). For the transitions with different delays sharing output places dummy transitions may be inserted right after these transitions, as shown in Figure 2).

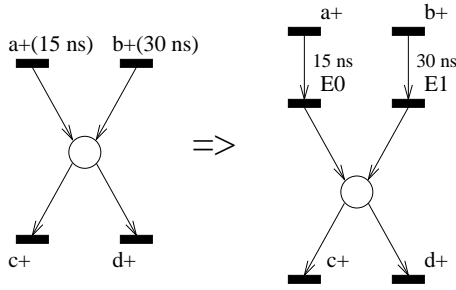


Figure 2: Reduction of a transition timed STG to PTSTG. $E0$, $E1$ are dummy transitions. Delays can be assigned to implicit places.

3.4.2 Place timed STG. In a *place timed STG model* (PTSTG) delays are assigned to places while transitions are said to fire immediately.

For illustrating specification of environment dependencies and delays by means of PTSTG let us consider an example shown in Figure 3. Signals b and c (Figure 3a) are implemented in the environment and their functions are unknown. Let us assume that it is known that rising transitions of these signals fire only after $a+$ fires and the transitions propagation delays between a and b , and a and c are different and known. Figure 3,b illustrates how those dependencies can be specified by means of STG and Figure 3,c shows how wire delays can be specified by the PTSTG model.

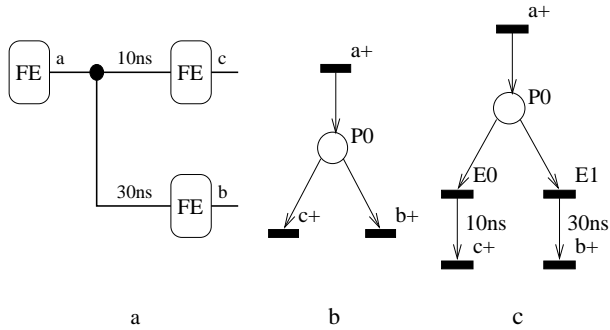


Figure 3: Specifying known environment dependencies and delays (a) by means of PTSTG (b,c). $E0$ and $E1$ are dummy transitions. Places between $E0$ and $c+$, $E1$ and $b+$ are implicit.

The PTSTG model for free-choice nets corresponds to the circuit wire delay model. For the limitations see Section 3.5. Every direct path R_{ij} in a PTSTG between signal transitions t_i and t_j corresponds to the

circuit wire between output of the FE implementing signal $\Delta(t_i)$ and input of the FE implementing signal $\Delta(t_j)$ if it is not a redundant dependency.

The correspondence between PTSTG and the wire delay model of a circuit is defined only for free-choice STGs, because different delays cannot be specified for two direct paths R_{ij} and R_{ik} in PTSTG such that $|\bullet t_j \cap \bullet t_k| > 1$ and $\exists p \in P : p \in \bullet t_j \cap \bullet t_k$, but the corresponding wires will present in the environment.

An example shown in Figure 4(c) illustrates the difficulties that can occur for non-free-choice STG in case two input wire delays for a FE have different values. A simple insertion of *dummy* transitions in the arcs $p0 \rightarrow d+$ and $p1 \rightarrow d+$ (like in Figure 3) can produce an unsafe STG (we assume that the fragment of the STG presented in the figure is a part of a cyclic behavior).

In such cases it is more convenient to use non-free-choice STG for behavior specification. (Consider example shown in Figure 4b.)

The place timed STG can be reduced to the arc timed STG defined below by assigning a place delay ($D(p_i)$) to the output arcs of p_i ($\forall f_{ij}(\bullet f_{ij} = p_i) : D(f_{ij}) = D(p_i)$) and assigning zero delays to all places ($D(p_i) = 0$).

3.4.3 Arc timed STG. In *arc timed STG* (ATSTG) delays are assigned to arcs, while places and transitions are assumed to have zero delays. ATSTG allows us a simpler way of specifying known delay information for the environment. With this model it is easy to specify the delay information even for non-free-choice STGs.

- A *ready token* in a *place timed PN* is a token that has entered a place p_i and the delay associated to the place has elapsed (so that this token is available for $p_i \bullet$).
- Let us denote with $C_{p_i}^R$ a moment in time when the number of ready tokens in place p_i becomes greater than zero.
- Let us denote with $C_{t_i}^E$ a moment in time when transition t_i becomes enabled.

Since for the best of our knowledge this model does not correspond to any previously considered in the literature timed PN model we define the operation rules for this model as follows.

- For every particular marking such that places $p_0 \dots p_m$ are marked transition t_i becomes enabled at time $C_{t_i}^E = \max(C_{p_0}^R + D(f_{qi}), \dots, C_{p_m}^R + D(f_{ri}))$ where $\{p_0 \dots p_m\} = \bullet t_i$ and $\forall f \bullet = t_i$.

- The difference in delays between the output arcs of a place does not affect the choice specified by the place. Therefore, a *nondeterministic choice behavior is preserved regardless of the delays assigned to the output arcs of a place.*

The use of this model is illustrated in Figure 4. Let us assume that there are five FEs in the environment that implements signals a, b, c, d and e . These signals are implemented with FEs with unknown logic functions (Figure 4a). However, signal dependencies are known, i.e. it is known that the rising transitions $c+$ and $d+$ depend on transition $a+$ of signal a and the rising transitions $d+$ and $e+$ depend on $b+$. Signal propagation delays between the FEs are assumed to be different and known as represented in Figure 4a).

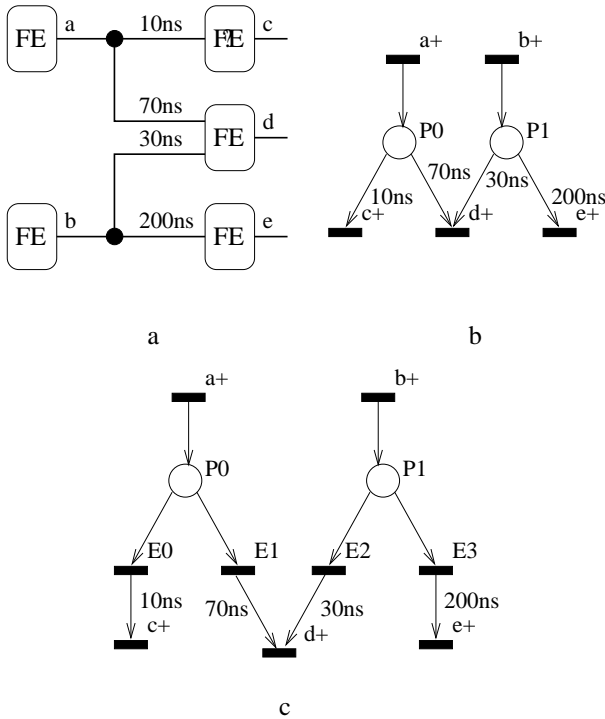


Figure 4: Example of using the arc timed STG model: (a) environment information; (b) ATSTG specification; (c) fragment of an unsafe PTSTG specification;

The ATSTG subnet specifying this behavior is shown in Figure 4b. For this example there is no easy way to specify delays by a free-choice STG (the subnet in Figure 4c is unsafe), therefore it is difficult to use a PTSTG model for specification of this behavior.

ATSTG can be reduced to PTSTG if the underlying PN is free-choice by inserting dummy transitions on

the output arcs of multiple fan-out places and on the input arcs of multiple fan-in as it is shown in Figure 5.

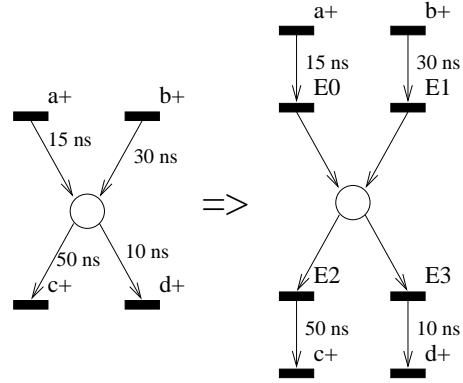


Figure 5: Free-choice ATSTG can be reduced to PT-STG by inserting dummy (E_0, \dots, E_4) transitions.

Arc timed STG corresponds to a circuit wire delay model in the following way see limitations in Section 3.5. Delay $D(R_{ij})$ of a direct path R_{ij} between signal transitions t_i and t_j corresponds to the delay of a wire between the output of the FE implementing signal $\Delta(t_i)$ and an input of the FE that implements signal $\Delta(t_j)$ if R_{ij} is not a redundant dependency.

Despite the convenience in behavior specification provided with this model ATSTG is *not a timed PN* since delays are associated with arcs. This can limit the use of ATSTG model.

3.4.4 Redundant combined models.

Transition-arc timed STG and *transition-place timed STG* can be obtained by combining the primary models. In these models delays are assigned to transitions and places, or transitions and arcs respectively. These models are redundant. Both can be reduced to ATSTG by adding delays of transitions to all of their output arcs. On the other hand, these models can be convenient for a designer.

3.5 Signal transition dependencies and wires

There is a relation between gate dependencies in a circuit and signal transition dependencies in the TSTG specifying circuit behavior. This relation, based on the notion of *trigger* and *context* signals defined below, helps to establish a correspondence between gate delay models and delay models for TSTG. Such a relation is easy to establish if a TSTG represents every gate of a circuit with a separate signal.

- An input signal of a FE is called a *trigger signal* (also sometimes called a *drive signal*) if its transition may cause a signal transition at the output of the FE; otherwise it is called a *context signal*.
- If signal a is a trigger signal for b , then there is a direct path in the corresponding TSTG between some transition of a and some transition of b .
- If, vice versa, there is an irredundant direct path R_{ij} in the TSTG, then signal $\Delta(t_i)$ is a trigger for signal $\Delta(t_j)$.
- A redundant direct path in a TSTG may or may not correspond to a wire in the implementation.
- An irredundant input wire for signal b , corresponding to a context signal c may or may not correspond to a redundant direct path in the TSTG.

4 STG VHDL specification.

VHDL hardware description language supports specifications of hardware systems from gate level to higher behavioral levels. It supports concurrency, design decomposition and a mechanism for independent parts of a hardware system to communicate each other. Most of used VHDL statements will be briefly explained with STG VHDL specification examples.

4.1 Previous work.

Previous works on simulation of PN and STG behavior in the VHDL environment are known. PN composition and simulation are addressed in [11]. Peter Vanbekbergen et al [2] used VHDL to simulate and validate STG behavior as well as the synthesized circuit behavior. VHDL environment has been also used for interactive synthesis of asynchronous circuits [3] though the behavior specification model different from STG had been utilized there.

4.2 STG behavior VHDL specification.

The technique, we propose, allows to simulate the behavior of *bounded* and *consistent* timed STG. Boundedness means that the number of tokens at any place at any time is bounded. Consistency, in turn, requires that: 1) every rising transition of a signal is followed (not necessarily immediately) by a falling transition of this signal and vice-versa; 2) signal transition cannot be concurrent to another transition of the same signal. STG behavior is simulated in terms of signals, i.e. the simulation output is a timing diagram of signals' behavior.

The technique proposed in this paper is implemented as a part of the Taxo-Synthesis asynchronous

circuits' CAD tool. All timed STG models outlined above are supported in this tool.

We illustrate the proposed method with an example of a nondeterministic non-free-choice STG with concurrency as shown in Figure 6.

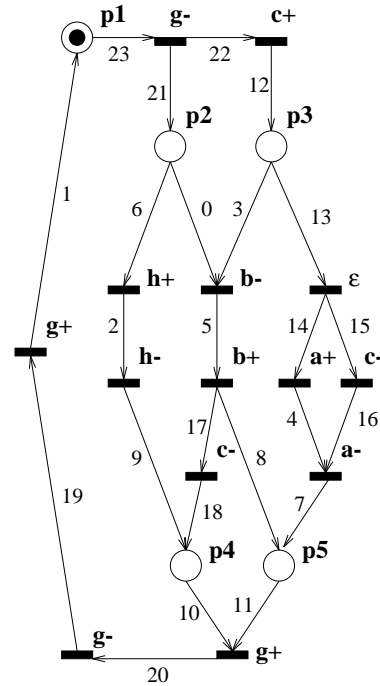


Figure 6: A simple nondeterministic non-free-choice STG with concurrency (numbers on arcs are the arcs' names).

4.2.1 High-level modularity. Let us consider autonomous STG. On a high level of abstraction the behavior, described by autonomous STG, consists of behavior to be implemented and the environment behavior specifications. Although initially both are represented by STG the parts may undergo independent alternation during the design process. Different implementations may be obtained for the first part, while the environment behavior may either remain unchanged, be generalized or specified more precisely.

Therefore, it is natural to specify behavior for these parts independently and connect them via input and output signals. This is a well-known technique among VHDL users - a separate behavior specification that may remain unchanged during the design process is used as a test bench for the circuit under design. This makes it possible to simulate different implementations as well as circuit behavior specification with the

same environment or the same implementation in different environments.

Thus, given an autonomous STG we generate three VHDL entities as follows.

- The first entity, named `CIRCUIT`, specifies the circuit part of the STG behavior - the behavior of circuit output and internal¹ signals. Circuit output signals are defined as output and circuit input signals are defined as input for this entity.
- The second entity, named `ENVIRONMENT`, specifies the environment part of the STG behavior - the behavior of environment signals. Circuit input signals are defined as output and circuit output signals are defined as input for this entity.
- The third entity, named `INTERFACE`, specifies the `CIRCUIT - ENVIRONMENT` interconnection.

4.2.2 Low-level specification. A state of STG is uniquely identified by a set of marked places. STG reachability graph may, however, grow very large for concurrent systems. This makes its use inappropriate for VHDL specification. Alternatively enabling condition that is a set of places may be specified for every transition in STG. This forms a number of VHDL blocks equal to the number of transitions in STG. Such a block consists of the condition specification and three action blocks executed successively, whenever the enabling condition is met.

Enabling condition consists of the number of places that must be marked for the transition $t \in T$ to be enabled. Action parts are: decrementing the number of tokens for the places $p \in \bullet t$; firing transition t (altering the value of $\Delta(t)$); incrementing the number of tokens for the places $p \in t\bullet$. This is the method used in [2].

Our method extends this technique in the following ways.

1. Our method is developed for simulation of timed STG (i.e. time is specified with STG using one of the previously described delay models).
2. Our technique may be used to generate a VHDL code that would guarantee that every conditional path in STG is activated during simulation.
3. We limit the class of STG to be simulated to the class of consistent and bounded STG. These

¹Output (Z) signals are further divided here into the sets of output (observed by circuit and environment) and internal (observed only in the circuit).

properties must always be satisfied for a behavior that could be implemented by a finite circuit. On the contrary, it is not shown in [2] how the technique presented could handle non-free-choice STGs. In [2] the decision on the direction, the token is moved from the place, is made independently for every place with more than one output transition. That may easily lead to a simulation deadlock for a non-free-choice STG.

4. Multiple transitions of the same signal in the same direction are allowed in STG - they are resolved with an auxiliary variable stored with every such a signal. (this technique was also used in [3]).

On the other hand, the method from [3] uses VHDL environment not only for simulation but also for analysis.

Parts of the VHDL code generated for the example from Figure 6 are shown below.

The type `indexed` from [3] is used to store the transition index together with the immediate signal value.

First fragment of the VHDL code shows the falling transition of signal `g_int`² enabling condition, decrementing the number of available tokens on the input place and scheduling the transition itself with the corresponding delay. Note that `:=` is a variable assignment while `<=` is a signal assignment in VHDL. The primary difference is that signal assignment is never completed immediately but scheduled in time instead.

```
if (place8ready > 0) then
    place8ready := place8ready - 1;
    g_int <= ('0',0) after 10 ns;
end if;
```

Second fragment shows the block where the 0-th (in the case of transitions' indexes are zero based) falling transition of signal `g` firing is checked. If the firing has occurred markers are transferred to the places following this transition in STG and their "timers" are activated. These timers for place 1 and arc 22 are implemented with VHDL signals `place1` and `arc22`. Variables `place1drive` and `arc22drive` store the direction of the last transition scheduled for `place1` and `arc22`. Note the use of the type `indexed` for signal `g`: `g=('0',0)` means that `g` has logic value '0' and `index` (for instance occurrence number) is equal 0.

```
if ((g'event) and (g=('0',0))) then
    if (place1drive) then
```

²Signal `g_int` is a copy of signal `g` internal for circuit VHDL entity. It is necessary since signal output for the entity cannot be observed inside the entity.

```

        place1 <= transport '0' after 10 ns;
    else
        place1 <= transport '1' after 10 ns;
    end if;
    if (arc22drive) then
        arc22 <= transport '0' after 5 ns;
    else
        arc22 <= transport '1' after 5 ns;
    end if;
    place1drive := not place1drive;
    arc22drive := not arc22drive;
end if;

```

The following fragment shows how a token on the arc 8 becomes available as soon as the delay assigned to this arc has elapsed (event on the corresponding signal is detected). There `arc22ready` stores the number of tokens available on the arc. It is necessary when the immediate number of unavailable tokens on the place or arc is greater than one, that is often the case for n -bounded STGs with $n > 1$.

```

if ( arc22'event ) then
    arc22ready := arc22ready + 1;
end if;

```

4.2.3 Choice simulation. As soon as STG is capable of representing choice a method is necessary to simulate it. For nondeterministic behavior (with nondeterministic choice) the problem is to define a preselection policy [7] for the markings with choice.

In general, a set of transitions may be enabled to fire at some given point of time ($C_{\{t_0, \dots, t_m\}}^E$). Some of them may fire, while others can be disabled if tokens can be removed from the shared input places. This behavior may be used in the environment specification to avoid specifying very complex or unknown dependencies or to specify the circuit to be implemented behavior if arbiters are allowed in implementation.

In order to simulate nondeterministic choice we define a *place pattern* and a *transition pattern* as follows.

A *place pattern* Pat^P is a set of one or more places $p_i \in P$ such that they are concurrent, i.e. there exists a reachable marking where all places in the pattern are marked. A place pattern is uniquely identified by 1) the set of places in the pattern; 2) the number (greater than zero) of tokens at each place.

A *transition pattern* Pat^T is a set of one or more transitions $t \in T$ defined for a given place pattern in the following way: 1) every transition in the pattern must have input arcs from a place pattern; 2) all transitions from the pattern can fire simultaneously (this should be distinguished from being enabled simultaneously: for example two transitions sharing a single

input place are enabled simultaneously, but cannot fire simultaneously if the shared place is marked with one token).

A place pattern is said to be *connected* if 1) every place from the pattern share output transition with at least one other place in the pattern; 2) a graph formed by the places $p \in Pat^P$, all transitions $t \in p_i \bullet$ such that $p_i \in Pat^P$ and arcs connecting them is connected.

A place pattern is said to be *global* if it contains all marked places in STG and *local* otherwise.

A place pattern is said to be *with choice* if a number greater than one of transition patterns may be defined on the set of transitions that are output to its (place pattern's) places.

In the example from Figure 6 the set $\{p0, p1\}$ is a place pattern with choice. The corresponding transitions patterns are $\{b-\}$ and $\{E1*, h+\}$. VHDL specification for these places pattern follows. Expression $(place0ready > 0)$ and $(place1ready > 0)$ is the places pattern guard condition; `GetDirection` is the function that calculates the next transitions pattern to activate; `FCGroup_0_Direction` is a variable that stores the number of the next transitions pattern to activate;

```

if ((place0ready>0) and (place1ready>0)) then
    if (FCGroup_0_Direction = 0) then
        b_int <= '0' after 10 ns;
        place1ready := place1ready - 1;
        place0ready := place0ready - 1;
    end if;
    if (FCGroup_0_Direction = 1) then
        if (dummy_E1 = '1') then
            dummy_E1 <= '0' after 0 ns;
        else
            dummy_E1 <= '1' after 0 ns;
        end if;
        place0ready := place0ready - 1;
        h_int <= '1' after 10 ns;
        place1ready := place1ready - 1;
    end if;
    FCGroup_0_Direction := GetDirection(
        FCGroup_0_Direction,
        FCGroup_0_MaxValue );
end if;

```

All transitions patterns are defined for every place pattern with choice and transition patterns are activated in some order whenever the place pattern is activated. The iteration order must be defined for every place pattern with choice.

This approach allows defining of either the order or the probability distribution law for the sequence of

transition patterns activation for every place pattern with choice. The use of patterns allows handling of non-free-choice STG behavior.

The number of patterns necessary for simulation is minimal if only connected place patterns with choice are used. However, this preselection policy cannot guarantee that every conditional path is activated during simulation.

It can be guaranteed if all global patterns with choice are generated. This, however, can exponentially increase the size of the VHDL code for the nets with concurrency.

4.3 Joint behavior-circuit simulation.

Despite the fact that most of the synthesis methods/tools are proved to provide correct solutions under a given set of assumptions, simulation is still widely used to check the validity of the design behavior. This convinces a designer in the correctness of the synthesized circuit on one hand and lets a designer to check if the assumptions important for the design match those of the synthesis system.

A joint simulation possibility is provided by the behavior VHDL specification modularity. A few more requirements must be met, however, to make possible a circuit-environment simulation. Those are out of the scope of this paper.

5 Conclusion.

In this paper some possible timed extensions to STG are considered. They are based on associating delays to transitions, places, arcs or some of their combinations. We also have studied the correspondence between different timed STG models and wire and gate delay models for circuits. Some limitations of the timed STG models are discussed.

The use of the VHDL for formal specification of the introduced timed STG models is the major contribution of the paper. Both parts of STG VHDL code – a circuit and its environment are considered interacting asynchronously. Concurrency, choice, synchronization, causality, and timing information are expressed using VHDL specification of timed STG.

This allows us to use the proposed approach for simulating behavior of speed-independent and semi-modular circuits and their interaction with environment. An overview of the design methodology for semi-modular circuits was given to illustrate the use of STG simulation.

The timed STG models and the algorithm for translating timed STGs to VHDL are implemented in the

Taxo-Synthesis CAD system.

6 Acknowledgements

The authors are very grateful to Michael Kishinevsky for the invaluable advises on the preparation of the final version of our paper.

References

- [1] T.A.Chu, C.K.C.Leung and T.S.Wanuga A design methodology for concurrent VLSI Systems. In *Proceedings of ICCD* pages 407-410, 1985.
- [2] P. Vanbekbergen, A. Wang, K. Keutzer. "A Design and Validation System for Asynchronous Circuits", In *Proceedings of DAC*, 1995.
- [3] N.A.Starodoubtsev, A.V.Yakovlev and S.Yu.Petrov. Use of VHDL - based environment for interactive synthesis of asynchronous circuits, *VHDL User Forum Europe: Proceedings SIG-VHDL Spring'96 Working Conference*. Dresden, Germany, May 1996, Shaker Verlag, Aachen, 1996, pp. 21-33.
- [4] R.E.Miller. Switching theory. Vol.II.Sequential circuits and machines. John Wiley & Sons Inc. 1966.
- [5] A.Kondratyev, J.Cortadella, M.Kishinevsky, E.Pastor, O.Roig and A.Yakovlev. Checking Signal Transition Graph Implementability by Symbolic BDD Traversal, In *Proceeding of EDT Conference*, Paris, March 1995, IEEE Comp. Society Press, N.Y., pp. 325 - 332.
- [6] L.Lavagno and A.Sangiovanni-Vincentelli. *Algorithms for Synthesis and Testing of Asynchronous Circuits*, Kluwer Academic Publishers, 1993.
- [7] F. Bowden "Modeling time in Petri net", In proceedings of "The second workshop on stochastic models", Gold Coast, July 1996
- [8] P. Vanbekbergen, G. Goossens, H. De Man "Specification and Analysis of Timing Constraints in Signal Transition Graphs." IMEC Laboratory, Kapeldreef 75, B-3001 Leuven, BELGIUM
- [9] P. Vanbekbergen, G. Goossens, B. Lin "Relational and Timing Semantics for a Timed Signal Transition Graph Model." IMEC Laboratory, Kapeldreef 75, B-3001 Leuven, BELGIUM
- [10] P. Vanbekbergen, C. Ykman-Couvereur, B. Lin, and H. De Man. Generalizing signal transition graphs for modelling mixed asynchronous/synchronous and arbitration behavior. In *Proceedings of International Workshop of Logic Synthesis*, May 1993
- [11] S. Mohanty. "An Integrated Design Environment for Rapid System Prototyping, Performance Modeling and Analysis using VHDL". (Thesis.) September, 1994