Formal Modeling for Persistence Checking of Signal Transition Graph Specification with Promela

Kanut Boonroeangkaow, Arthit Thongtak and Wiwat Vatanawood

***Abstract*—Signal Transition Graph is a marked graph that represents signal flow behavior in concurrent system. In this paper, we are interesting in a Formal Verification for Signal Transition Graphs. We propose a method of formal modeling for Persistence checking of Signal Transition Graph specification with Promela. Signal Transition Graphs are modeled in Promela and run in SPIN Model Checker. Linear Temporal Logic is used for defining Persistence and Safetyness properties. Our method deals with both single-cycle and multi-cycle Signal Transition graphs.**

***Index Terms*— Formal verification, Model checking, Signal Transition Graph, SPIN, Promela.**

I. Introduction

Petri Nets are widely used for describing concurrent systems. However, regarding to the complexity of Petri Nets, Nowadays Signal Transition Graphs are interested to represent the system behaviors, such as asynchronous circuit [1] and genetic networks [2]. Signal Transition Graph is an interpreted Petri Nets. It used to represent the signal transition of concurrent systems. From Signal Transition Graph, we can visualize important properties in order to analyze or verify signal behaviors. However, automatic techniques are still needed.

Model checking is one of powerful verification technique. There are a lot of model checker software such as *NusMV*, *Blast*, *SPIN*. SPIN supports language called Promela and Linear Temporal Logic.

This paper was motivated by the interesting of formal modeling and verification of Signal Transition Graph. The proposed method can reduce the time on verification and run in automatic by using SPIN.

There are a few papers [3] use model checking to verify properties of Signal Transition Graph. In this paper, we propose the method for Signal Transition Graph properties checking in Promela using SPIN. The types of Signal Transition Graph that we focus are single-cycle and multi-cycle. First, a model of Signal Transition Graph is written in Promela. Then, properties on the system behavior are specified by Linear Temporal Logic. Linear Temporal Logic is written in description to define properties condition. Finally, SPIN runs to check, if the model doesn't meet properties, SPIN will provide a counter example.

The rest of this paper are organized as follows. Section 2 we present preliminaries on Signal Transition Graph, Model checking, Promela and SPIN. Section 3, we present related work. Section 4, we present Signal Transition Graph specification. In section 5, we describe our persistence checking scheme. In section 6, we present how to write each properties in Linear Temporal Logic (LTL). Section 7, we conclude the paper and give direction to the future work.

II. PRELIMINARIES

A. Signal Transition Graph

To specify behavior of system [3][1], we can use form of the graph called Signal Transition Graph (STG). STG shows the rising and falling transition of signals. If we specify asynchronous system by using STG, there are three types of signal: input, output and internal. It represents signal and the transition by the signal name and transition like *a*+, *a*-. *a*+ means rising transition that signal *a* changes from 0 to 1. *a*- means falling transition that signal *a* changes from 1 to 0. Input signal name will be underlined. STG consists of 3-tuples that formally define as $\sum = <T,F,M>$. T is a finite set of signal transition. F is a set of flow relations (arc). And M is the set of marking or tokens. If the transition is enabled, we call the transition is fired. Fig.1 adapted from [3] which shows the example of STG of C-element.

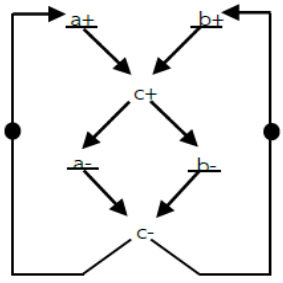

Fig.1 Signal Transition Graph of C-element adapted from [3]

In this example, signal *a* and *b* are the inputs of the system. *a*+ and *b*+ are the rising transition of signal *a* and *b*. *a*- and

Kanut Boonroeangkaow is a graduate student of department of Computer Engineering, Faculty of Engineering, Chulalongkorn University. His research interest is Software Engineering (e-mail: Kanut.B@student.chula.ac.th).

Arthit Thongtak is currently an Assistant Professor of department of Computer Engineering, Faculty of Engineering, Chulalongkorn University. His research interests include Digital System Engineering, Digital Systems Testing, Fault Tolerant Computing, Asynchronous System Design (e-mail: arthit.t@chula.ac.th), IAENG member.

Wiwat Vatanawood is currently an Associate Professor of department of Computer Engineering, Faculty of Engineering, Chulalongkorn University. His research interests include Formal Specification, Formal Verification, Software Architecture (e-mail: wiwat@chula.ac.th).

ISBN: 978-988-14047-3-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

b- are the falling transition of signal *a* and *b*. Signal c is the output of the system. The initial marking are at *c*-.

The properties that interested in this paper are Persistence and Safetyness.

Persistence is defined as follow [4]: Transition *s** is persistent to a non-input transition *t**. If *s* is a trigger transition of *t*. And transition *s*'* can be fired after *t** has been fired.

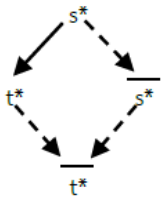

Fig.2 Example of non-persistence STG adapted from [4]

Example in Fig.2 adapted from [4] which shows the example of STG of C-element. *s** is the trigger transition of *t**. *s*'* can be fired before *t**. So The STG is non-persistence.

Safetyness is defined as follow : Not more than one token in an Arc.

B. *Model checking Techniques*

In formal verification [5], theorem proving and model checking is well known method that can be use for verification. Model checking is the method that can be exhaustively checking and confirming that the model meets the given specification. Theorem proving can check infinite space but it is hard to do automatically. Model checking can verify automatically. Because of this reason, model checking has been chosen for our method.

C. *Promela and SPIN*

We use SPIN [6][7] as a Model Checker. Language that supports in SPIN called Promela (Process Meta Language). Promela is one of verification language. It is nearly same as C language, and common for developers. The variable type is nearly same as C such as int, long, etc. This process declared by the word proctype. SPIN can also use for concurrent processes. It can make exhaustive search and simulation for formal model.

Moreover, SPIN can used as a full LTL model checking system. LTL can check logic in a linear time. This ability of LTL helps confirming STG properties. LTL description in SPIN will write *ltl* at first. Then the name of LTL will be the next. Then the temporal operator will define. The logic of the signal will be the last. The logic of the signal can define out of the description such as "*#define p1(p1==0);*". In this paper, LTL used for define properties of the STG and combined with the Promela code. The checking are done by SPIN.

III. Related Work

Signal Transition Graph was propose by Tam-An Chu [1]. STG is an interpreted Petri Nets that easy to understand. The STG properties describe in this work.

S.Park [4] proposed the generated method from STG. He generates asynchronous circuits by STG. The verification properties check by lock-relation in manual method.

W.Lawsunnee [3] proposed verification method via SPIN. His method used SPIN to simulation. Then he checks the lock-relation from the STG. Then he compares the simulation result with lock relation. The method can check only Persistence and Liveness properties. However, this method was manually checking.

Zohra SbaY, etc [8] proposed a verification technique of business processes by model checking. The model was in Petri nets. This work didn't show the persistence, liveness, safetyness properties verification. However, this model can be adapted for STG.

IV. STG Specification

STGs are categorized based on the cycle of signal transition which are single-cycle STGs and multi-cycle STGs. In this paper, we consider both STGs.

A. *Single-cycle STG*

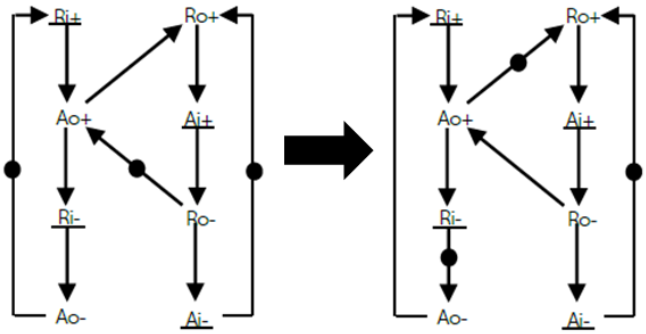

Fig.3 Example of Single-cycle STG adapted from [1]

The example of STG in Fig.3 adapted from [1] is single-cycle STG that has persistence problem. There are two input signals and two output signals represented in *Ai*, *Ri*, *Ao* and *Ro* respectively. Each signal consists of two kinds of possible value: plus (+) and minus (-). The initial marking are at *Ao*-, *Ai*- and *Ro*-. It can be seen that *Ao*+ are enabled when the tokens from *Ri*+ and *Ro*- are arriving. Then the tokens are consumed by *Ao*+ and it produces double tokens on arcs (*Ao*+ to *Ro*+ and *Ao*+ to *Ri*-). In this status both *Ro*+ and *Ri*- are enabled. If *Ri*- is fired before *Ro*+ and also *Ao*- is fired, the token on the arc of *Ao*+ to *Ro*+ is removed, since signal value of *Ao* is changed to minus.

B. *Multi-cycle STG*

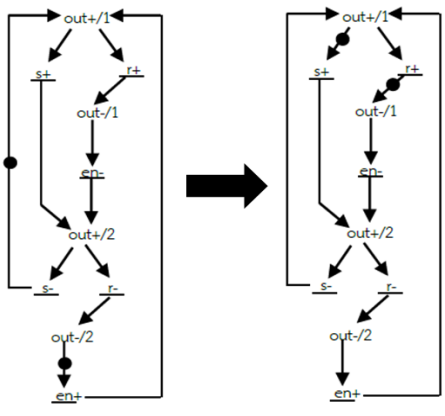

Fig.4 Example of Multi-cycle STG adapted from [1]

ISBN: 978-988-14047-3-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

The example of STG in Fig.4 adapted from [1] is multi-cycle STG that has persistence problem. There are three input signals and one output signal represented in *s*, *r* and *en* and *out* respectively. The output signal has two cycles that represented as *out/1* and *out/2*. The initial marking are at *s-* and *out-/2*. It can be seen that *out+/1* is enabled when the tokens from *s-* and *en+* are arriving. Then the tokens are consumed by *out+/1* and it produces double tokens on arcs (*out+/1* to *s+* and *out+/1* to *r+*). In this status both *s+* and *r+* are enabled. If *r+* is fired before *s+* and also *out-/1* is fired, the token on the arc of *out+/1* to *s+* will be removed, since signal value of *out* is changed to minus. The multi-cycle STGs have to check all represent signals that have more than one cycle. If out+/2 have the same situation, the value of signal *out* is change too.

V. OUR PERSISTENCE CHECKING SCHEME

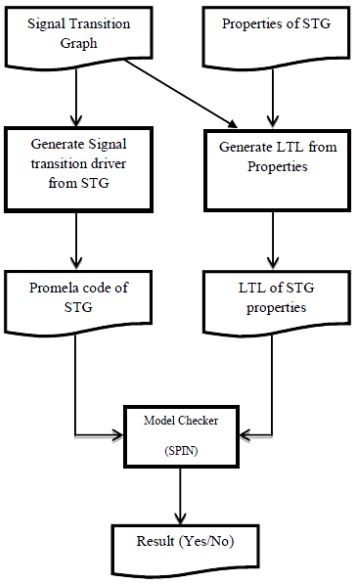

Fig.5 Our persistence checking scheme

In this paper, our Persistence checking scheme is shown in Fig.5. In the beginning, The given STG is translated (modeled) to Promela. Then properties, such as Persistence, will be specified in LTL description. The LTL description is referred from STG. Then Promela and LTL description are loaded to SPIN. Finally, the verification of the above descriptions are executed via SPIN. The result is shown as Yes or counter example.

In this section, we focus on the Promela modeling from the given STG. The structure of Promela modeling consists of 4 parts as follows.

1) Define all signal names
2) Define transition rule
3) Define initial marking
4) Define the flow relation

Since STGs is interpreted Petri-Nets by reducing places, the tokens on the place have to be located on its arc. Therefore, we define the signal name in STG including each input and output signal transition with its arc. For example in Fig.6a, the signal name is defined as *"x1px2m"*. In Fig.6b, there are 2 signal names defined as *"x1px2m"* and *"x1px3p"*.

There are three status in STG that are defined as enable status, disable status, and firing status. Enable status is the status that every input arc of signal transition have at least one token each. Disable status is the status that at least one input arc has no token. Firing status is the status that output arc of signal transition has token.

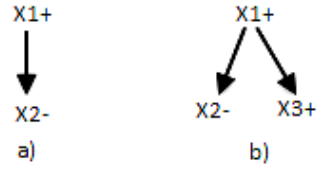

Fig.6 a) Signal name with one output b) Signal name with two outputs

The transition rule consists of enable status and its related firing status. When an enable status occurs, then firing status will occur respectively. The number of token on all inputs arc will be reduced by one, and the number of token on all outputs arc will be added by one. From this transition rule, signal persistence in every enable status will be retained until the related firing status occurs.

In Fig.7, The left side of Fig.7a and 7b shows enable status and the right side shows firing status.

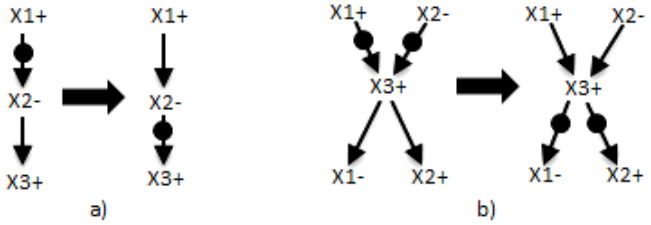

Fig.7 a) Flow relation with one input and one output b) Flow relation with two inputs and two outputs

The transition rule is defined by number of signal named in enable status and firing status. Input arc of signal transition is defined as:

"inpn(x1,…,xn) ((x1>0)&&…&&(xn>0))-> x1--,…,xn--".

The output arc is defined as:

"outpn(x1,…,xn) x1=x1++,…,xn++".

The initial marking is defined with signal name and its value as:

"signal name = value".

The flow relation is defined by atomic statement and covered by loop statement in order to generate all possible paths. In the atomic statement, this input arc implys with output arc. For Example in Fig.7a, the flow relation is defined as:

"atomic{inp1(x1px2m) -> outp1(x2mx3p)}".

In Fig.7b, the flow relation is defined as:

"atomic{inp2(x1px3p,x2mx3p) -> outp2(x3px1m,x3px2p)}"

ISBN: 978-988-14047-3-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

- Single-cycle STG

Single-cycle Promela modeling is as same as the rule explained in this section.

- Multi-cycle STG

Multi-cycle STG has to remark for the signal name declared. For example in Fig.8, signal *x3* has two cycles, so the signal name will be declared including with the number of its cycle. The above signal will be declared as *"x3p1x1p"* and *"x3p1x2m"*. The below signal will be declared as *"x3p2x1m"* and *"x3p2x2p"*. Others part are the same.

The example of Promela modeling structure for single-cycle STG on Fig.3 is shown in Fig.9

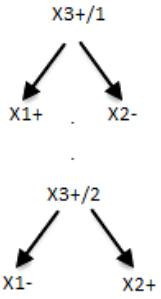

Fig.8 Example of multi-cycle signal

```
//define signal name
#define signal    byte
signal ripaop,aoprop,aoprim,rimaom,aomrip;
signal ropaip,aiprom,romaim,romaop,aimrop;

//define transition rule
#define inp1(x)      (x>0) -> x = x-1
#define inp2(x,y)    (x>0 && y>0) -> x = x-1; y=y-1
#define out1(x)      x = x+1
#define out2(x,y)    x = x+1; y = y+1


init {
//define initial marking
aomrip = 1; romaop = 1; aimrop = 1; /*initial marking*/

//define flow relation
do
    :: atomic { inp1(aomrip)    -> out1(ripaop) }
    :: atomic { inp2(ripaop,romaop) -> out2(aoprop,aoprim) }
    :: atomic { inp1(aoprim)    -> out1(rimaom)}
    :: atomic { inp1(rimaom)    -> out1(aomrip) }
    :: atomic { inp2(aimrop,aoprop) -> out1(ropaip) }
    :: atomic { inp1(ropaip)    -> out1(aiprom) }
    :: atomic { inp1(aiprom)    -> out2(romaim,romaop)}
    :: atomic { inp1(romaim)    -> out1(aimrop) }
od
}
```

Fig.9 Example of Promela modeling

VI. LTL DESCRIPTION FOR PROPERTIES AND VERIFICATION

This section, we present LTL description for persistence property checking, each type of STG and result of the verification.

A. *Persistence Property*

Persistence confirm correctness of all transition of the signal in STG. In the previous section, single-cycle and multi-cycle STG Promela modeling was shown. The LTL description rule for single-cycle and multi-cycle STG have a little bit difference.

The step to construct for the LTL description is shown as following

1) Find trigger signal name of concurrent path in STG

2) Define the truth value of the opposite trigger signal name as 1, and connect the signal name and truth value from all of this opposite trigger signal name with *AND* condition. If the STG is multi-cycle and same trigger signal name connected it with *OR* condition.

3) Define the truth value of the trigger signal name as 0, and connect signal name and truth value from all of this trigger signal name with *AND* condition for both types.

4) Specify the LTL description by implies condition and global for temporal logic.

5) If STG have more than one trigger signal names, connect LTL description by *AND* condition.

- Single-cycle STG

The example of LTL description refer from Fig.3

1) The trigger signal name are *Ao+* and *Ro-*

2) Signal name from *Ao-* and *Ro+* are defined the truth value as:

"#define op_A(aomrip == 1)" and
"#define op_R(ropaip == 1)"

3) Signal name from *Ao+ and Ro-* are defined the truth value as:

"#define A((aoprim == 0) && (aoprop == 0))" and
"#define R((romaim == 0) && (romaop==0))"

4) Specify LTL description. The LTL description defined from the definition of Persistence property. This means the trigger signal of the concurrent signal will not have any token if the opposite of trigger signal has token. So we can define as globally and check implies from op_A to A, and op_R to R. So the LTL will be:

"[](op_A->A)" and "[](op_R->R)"

5) There are more than one trigger signal, so connected it by AND condition. The LTL will be:

"[]((op_A->A)&&(op_R->R))"

- Multi-cycle STG

The example of LTL description refer from Fig.4

1) The trigger signal name are *out+/1* and *out+/2*. *out/1* and *out/2* represent the same signal *"out"*. So both represent trigger signal name *out+*.

2) The transition from *out/1-* and *out/2-* are defined the truth value as:

"#define op_out((outm1enm==1) || (outm2enp==1))"

3) The transition from *out+/1* and *out+/2* are defined truth value as:

"#define out((outp1sp == 0) && (outp1rp == 0)
&& (outp2sm == 0) && (outp2rm == 0))"

4) The LTL description is specifying as one trigger signal name. So the LTL will be:

ISBN: 978-988-14047-3-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

"[](op_out->out)"

B. Safetyness Property

STG must have token in every transition not more than one token at all time. We will be defined as:

"#define safe(All signal name <=0)"

The LTL description for safetyness is *"globally safe"*. To check that all the time so all signal not have more than one token, therefore the LTL will be:

"[]safe"

SPIN will show an error, if the token in any signal name not meet the description.

C. Result

The example single-cycle STG in Fig.3 was run in SPIN, the result shown in Fig.10

```
(Spin Version 6.4.5 -- 1 January 2016)
    + Partial Order Reduction

Full statespace search for:
    never claim          + (p1)
    assertion violations    + (if within scope of claim)
    acceptance   cycles     + (fairness disabled)
    invalid end states   - (disabled by never claim)

State-vector 36 byte, depth reached 14, errors: 1
       8 states, stored
       0 states, matched
       8 transitions (= stored+matched)
       0 atomic steps
hash conflicts:        0 (resolved)
```

Fig.10 Result of example single-cycle STG

The result shows error, this single-cycle STG is non-persistence.

The example multi-cycle STG in Fig.4 was run in SPIN, the result shown in Fig.11

```
(Spin Version 6.4.5 -- 1 January 2016)
    + Partial Order Reduction
Full statespace search for:
    never claim          + (p1)
    assertion violations    + (if within scope of claim)
    acceptance   cycles     + (fairness disabled)
    invalid end states   - (disabled by never claim)

State-vector 36 byte, depth reached 23, errors: 1
      14 states, stored
       2 states, matched
      16 transitions (= stored+matched)
       0 atomic steps
hash conflicts:        0 (resolved)
```

Fig.11 Result of example multi-cycle STG

The result shows error, this multi-cycle STG is non-persistence.

VII. Conclusion

In this paper, we present a method of formal modeling for persistence checking of signal transition graph specification with Promela. SPIN model checker was used. As the result, we can find non-persistence properties, as well as safetyness properties for both single-cycle and multi-cycle STGs.

Future works include formal modeling for checking of others properties, and extend this work for free-choice STGs.

References

[1] Tam-Anh Chu, "Synthesis of self-timed VLSI circuits from graph-theoretic specifications", PhD thesis, Massachusetts Institute of Technology, June 1987.

[2] R. Banks, V. Khomenko, and L. J. Steggles, "A Case for Using Signal Transition Graphs for Analysing and Refining Genetic Networks," *Electron. Notes Theor. Comput. Sci.*, vol. 227, no. C, pp. 3–19, 2009.

[3] W. Lawsunnee, A. Thongtak, and W. Vatanawood, "Signal persistence checking of asynchronous system implementation using SPIN," *Lect. Notes Eng. Comput. Sci.*, vol. 2, pp. 604–609, 2015.

[4] Park,S.B, "Synthesis of Asynchronous VLSI Circuits from Signal Transition Graph Specifications. Doctoral dissertation, Department of Engineering-Computer Science, Tokyo Institute of Technology,1996.

[5] Christel Baier and Joost-Pieter Katoen, "Principles of Model Checking", The MIT Press Cambridge, Massachusetts London, England, 2008.

[6] M. Ben-Ari, *Principles of the SPIN Model Checker*. 2008.

[7] Gerard J. Holzmann, "Principles of the Spin Model Checker", Springer-Verlag London Limited, 2008.

[8] Z. Sbai, a Missaoui, K. Barkaoui, and R. Ben Ayed, "On the verification of business processes by model checking techniques," *Softw. Technol. Eng. ICSTE 2010 2nd Int. Conf.*, vol. 1, pp. V1–97, 2010.

ISBN: 978-988-14047-3-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)