Efficient Model Checking of Asynchronous Systems Exploiting Temporal Order-Based Reduction Method

Chikatoshi YAMADA, Yasunori NAGATA, and Zensho NAKAO

Grad. School of Engineering and Science, Univ. of the Ryukyus
1 Senbaru, Nishihara, Nakagami, Okinawa 903-0213, JAPAN
Tel. : +81(98)895-8687, Fax. : +81(98)895-8708
E-mail: {yamada@gray., ngt@, nakao@augusta.}eee.u-ryukyu.ac.jp

Abstract: Recently design verification have been played an important role in the design of large scale and complex systems. In this article, we especially focus on model checking methods. Behaviors of modeled systems are generally specified by temporal formulas of computation tree logic. However, users must know well temporal specification because the specification might be complex. We proposed method that temporal formulas are gained inductively and amounts of memory and time are reduced. Finally, we will show verification results using our proposed method.

1. Introduction

Recently design verification have been played an important role in the design of large scale and complex systems. The verification ascertains whether designed systems can exactly be executed or not. Various formal methods for the verification have been studied by researchers[1][2][3][4]. In this article, we especially focus on model checking methods. In the model checking methods, a targeted circuit or a system will be modeled and unimportant or unrelated to verification tasks are going to be eliminated from modeled structures. Users can decide a structure of hierarchical(architecture, register-transfer or gate) levels in the modeling process. The circuits(or systems) can be modeled by using signal transition graph(STG) because we particularly aim to asynchronous handshake circuits in this research. Then behaviors of modeled systems are generally specified by temporal formulas of computation tree logic(CTL). Finally checked circuits are verified whether the circuits can satisfy descriptions of specification or not. In specifying temporal formulas, however, users must know well temporal specification because the specification might be complex. In this article, we extend our proposed method[5] to specification method which can obtain temporal formulas inductively from modeling systems. Moreover, we also show that verification tasks can be executed efficiently by using our proposed method.

The rest of this article is organized as follows: In section 2, Signal Transition Graph, Computation Tree Logic and NuSMV are orver-viewed, and in section 3 our proposed method is indicated by describing procedure of specification. Moreover, we demonstrate specification example using proposed method, and in section 5 some asynchronous circuit bench marks are used for verification to compare by NuSMV[7] tool. Finally, we summarize the discussion in section 6.

2. Preliminaries

2.1 Signal Transition Graph

Signal Transition Graph(STG)[6] is a well-known method for specifying asynchronous sequential circuits. STG is a sub-class of Petri nets, which are free choice nets. The structure for STG is a tuple $M =< S, R, S_0 >$, where S is a set of signal transitions in reachable states, R is a set of transition relations on S , and S_0 is a set of initial transitions, where S_0 is a subset of S.

2.2 Computation Tree Logic

The correctness property to be verified is specified in CTL. CTL is branching-time temporal logic, extending propositional logic with temporal operators that express how propositions change their truth values over time. Here we use temporal operators: Operators **G**, **F**, and **X** mean *globally* , *sometime in the future* , and *next time* , respectively. In CTL, these operators must be preceded by a path quantifier which is either **A** (*for all computation paths*) or **E** (*for some computation path*). We consider operators **AG**, **AF**, and **AX**. The formula **AG** p holds in state s if p holds in all states along all computation paths starting from s , while the formula **AF** p holds in state s if p holds in some state along all computation paths starting from s . The formula **AX** p holds in state s if p holds in all the states that can be reached from s in exactly one step. An atomic proposition is a CTL formula. If φ and ψ are CTL formulae, then so are $\neg\varphi$, $\varphi \wedge \psi$, **AF** φ , **AG** φ , **AX** φ . Similarly, operators **EG**, **EF**, and **EX** are along some computation path, respectively.

2.3 NuSMV

We execute verification using NuSMV[7] in this research. NuSMV is designed to be a well structured, open, flexible and documented platform for model checking. NuSMV is the result of the reengineering, reimplementation and extension of SMV[8]. NuSMV can process files written in SMV language, and allows for the construction of the model with different modalities, reachability analy-

sis, fair CTL model checking, computation of quantitative characteristics of the model, and generation of counterexamples. In addition, NuSMV features an enhanced partitioning method for synchronous models, and allows for disjunctive partitioning of asynchronous models, and for the verification of invariant properties in combination with reachability analysis. Furthermore, NuSMV supports LTL model checking. The algorithm is based on the combination of a tableau constructor for the LTL formula with standard CTL model checking.

3. Proposed method

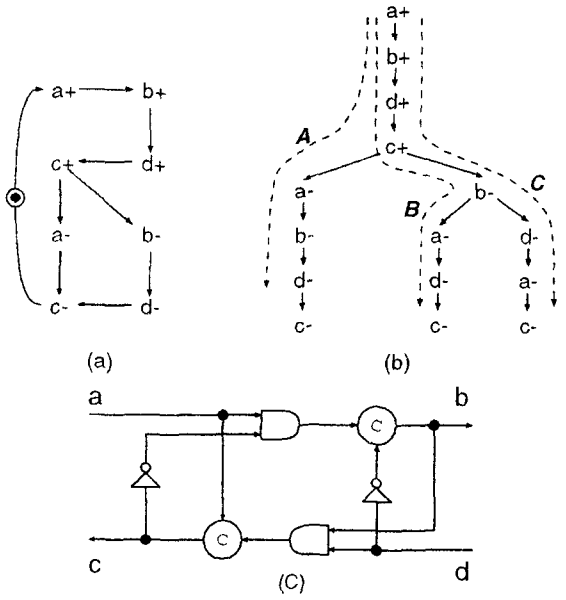

Figure 1. An asynchronous handshake circuit: (a) signal transition graph, (b) ranch expression, (c) a checked circuit, where '+' means raising edge and '-' means falling edge, respectively.

Here we show that temporal specifications can be obtained inductively. We especially take into account an asynchronous handshake circuit shown as fig.1. The procedure of specification can be indicated as follows.

[Procedure of specification]

[1.] Extracting all paths(fig.1(b)).

(A) a+ b+ d+ c+ a- b- d- c-
(B) a+ b+ d+ c+ b- a- d- c-
(C) a+ b+ d+ c+ b- d- a- c-

[2.] For each path, extracting input-output(IO) relation. The extraction can be repeated until detecting reverse value of a signal(such as from a+ to a-). If the reverse value are detected, then the extraction can start for the next signal. Initial values of all signals are "zero", i.e. "-".

(A) {(a+ , b+), (a+ , c+), (d+ , c+), (d+ , b-), (a- , b-), (a- , c-), (d- , c-), (d- , b+)}

Here we compare input a+ with d+ . Successors(outputs) of a+ are b+ and c+, and successors of d+ are c+ and b-, respectively. However, successor b- of d+ expresses the next cycle of path(shown as underline). Thus signal event a+ occurs earlier than b+. Such relation is called as *weak temporal order relation*.

(B) {(a+ , b+), (a+ , c+), (d+ , c+), (d+ , b-), (a- , c-), (d- , c-), (d- , b+)}

(C) {(a+ , b+), (a+ , c+), (d+ , c+), (d+ , b-), (a- , c-), (d- , c-), (d- , b+)}

For path (B) and (C), we can see that these paths are equivalence. Thus there are equivalent paths in IO relation. Although there is IO relation (a- , b-) in path (A), there is not (a- , b-) in (B). However, signal event b- in (B) can be occurred by (d+ , b-). The elimination of (a- , b-) shows that b- occurs earlier than a-. Thus, if there are reverse IO relation between paths, such relation is called as *strong temporal order relation*.

[3.] Let us introduce temporal operators to IO relation. In path (A), IO relation (a+ , b+) shows that b+ is immediate successor of a+ , specified as **AX**(a+ , b+). Here temporal operator **AX** can be used because there is only a transition b+ from a+. Moreover, IO relation (a+ , c+) shows that c+ is successor of a+, not immediate, specified as **AF**(a+ , c+). Similarly, temporal operators can introduce to IO relation as follows.

(A) {**AX**(a+ , b+), **AF**(a+ , c+), **AX**(d+ , c+), **AF**(d+ , c-), **AX**(a- , b-), **AF**(a- , c-), **AX**(d- , c-), **AF**(d- , c+)}

(B) {**AX**(a+ , b+), **AF**(a+ , c+), **AX**(d+ , c+), **AF**(d+ , b-), **AF**(a- , c-), **AX**(d- , c-), **AF**(d- , b+)}

(C) {**AX**(a+ , b+), **AF**(a+ , c+), **AX**(d+ , c+), **AF**(d+ , b-), **AX**(a- , c-), **AF**(d- , c-), **AF**(d- , b+)}

Although path (B) and (C) did not distinguish in procedure 2., there can be done by introducing temporal operators.

[4.] Specifying all paths using temporal formulas. In all paths, transitions which are the same temporal operator and IO relation can be extracted.

{**AX**(a+ , b+), **AF**(a+ , c+), **AX**(d+ , c+), **AF**(d+ , b-), **AF**(d- , b+)}

These relation can *globally* be satisfied on all paths. Thus temporal operator **AG** can be introduced as

follows.

AG[**AX**(a+ , b+) ∨ **AF**(a+ , c+) ∨ **AX**(d+ , c+) ∨ **AF**(d+ , b-) ∨ **AF**(d- , b+)]

Since **AF** expresses "sometime in the future for all paths", the *next* operator **AX** can be covered as **AX** *subseteq* **AF**. Thus, for IO relation (a- , c-) and (d- , c-) on the path (B) and (C), temporal formulas can be specified as follows.

AG[**AF**(a- , c-) ∨ **AF**(d- , c-)]

Therefore,

AG[**AX**(a+ , b+) ∨ **AF**(a+ , c+) ∨ **AX**(d+ , c+) ∨ **AF**(d+ , b-) ∨ **AF**(a- , c-) ∨ **AF**(d- , c-) ∨ **AF**(d- , b+)].

[5.] Here, **AX**(a+ , c+) and **AF**(d+ , c+) can be combined as **AF**(a+ ∧ d+ , c+) because transition c+ is occurred by transitions a+ and d+ at the next. Thus the formulas can be specified as follows.

AG[**AX**(a+ ∧ d- , b+) ∨ **AF**(a+ ∧ d+ , c+) ∨ **AF**(d+ , b-) ∨ **AF**(d+ , b-) ∨ **AF**(a- , c-) ∨ **AF**(d- , c-)]

This temporal specification can be expressed liveness property. As mentioned above, we can be obtained temporal formulas inductively.

4. Specification example

In this section, we demonstrate specification of an asynchronous pipeline shown as fig.2. An STG specification of the pipeline in fig.2(middle) only shows behaviors of `ctrl` modules because arbitration modules are extremely important parts for asynchronous systems. First, temporal formulas are specified without our proposed method as follows.

[Specification without proposed method]

`[ctrl1]` **AG**[**AX**(ur+ , sr+) ∨ **AF**(ur+ ∧ sr+ , ra1+) ∨ **AF**(ur+ ∧ sr+ ∧ ra1+ , cr1+ ∧ ur1-) ∨ **AF**(ra1+ , sr-) ∨ **AF**(cr1+ , ra1-) ∨ **AX**(ra1- , cr1-) ∨ **AF**(sr- ∧ cr1- , ur+)]

`[ctrl2]` **AG**[**AX**(cr1+ ∧ cr2- , rr2+) ∨ **AF**(cr1+ ∧ rr2+ , ra2+) ∨ **AX**(cr1+ ∧ rr2+ ∧ ra2+ , cr2+) ∨ **AF**(cr2+ , ra2+) ∨ **AF**(rr2+ ∧ ra2+ , cr1-) ∨ **AX**(ra2- ∧ cr1- , rr2-) ∨ **AF**(cr2- , cr1+)]

`[ctrl3]` **AG**[**AX**(cr2+ ∧ cr3- , rr3+) ∨ **AF**(cr2+ ∧ rr3+ , ra3+) ∨ **AX**(cr2+ ∧ rr3+ ∧ ra3+ , cr3+) ∨ **AF**(rr3+ ∧ ra3+ , cr2-) ∨ **AF**(ra3+ ∧ cr2- , rr3-) ∨ **AF**(cr3+ , ra3-) ∨ **AX**(ra3- , cr3-) ∨ **AF**(rr3- ∧ cr3- , cr2+)]

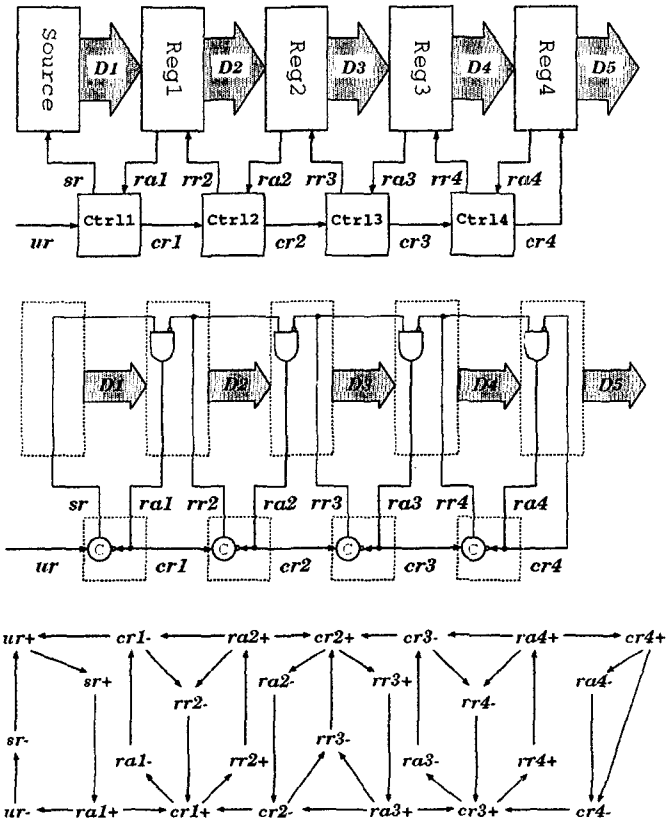

Figure 2. An asynchronous pipeline: (upper) an RTL structure, (middle) a low level construction of the pipeline, (lower) an STG of the pipeline.

`[ctrl4]` **AG**[**AX**(cr3+ ∧ cr4- , rr4+) ∨ **AF**(cr3+ ∧ rr4+ , ra4+) ∨ **AX**(cr3+ ∧ rr4+ ∧ ra4+ , cr4+) ∨ **AF**(rr4+ ∧ ra4+ , cr3-) ∨ **AF**(ra4+ ∧ cr3- , rr4-) ∨ **AF**(cr4+ , ra4-) ∨ **AX**(ra4- , cr4-) ∨ **AF**(rr4- ∧ cr4- , cr3+)]

This specification is the result of all behaviors for the `ctrl` modules in fig.2 because temporal formulas are considered not only input-output order relations but also output-input order relations, such as relation between sr+ and ra1+. Next, we indicate temporal formulas with our proposed method as follows.

[Specification with proposed method]

`[ctrl1]` **AG**[**AX**(ur+ ∧ ra1- , sr+) ∨ **AX**(ur+ ∧ ra1+ , cr1+) ∨ **AX**(ur- , sr-) ∨ **AX**(ra1- , cr1-)]

`[ctrl2]` **AG**[**AX**(cr1+ ∧ ra2- , rr2+) ∨ **AX**(cr1+ ∧ ra2+ , cr2+) ∨ **AX**(cr1- , rr2-) ∨ **AX**(ra2- , cr2-)]

`[ctrl3]` **AG**[**AX**(cr2+ ∧ ra3- , rr3+) ∨ **AX**(cr2+ ∧ ra3+ , cr3+) ∨ **AX**(cr2- , rr3-) ∨ **AX**(ra3- , cr3-)]

`[ctrl4]` **AG**[**AX**(cr3+ ∧ ra4- , rr4+) ∨ **AX**(cr3+ ∧ ra4+ , cr4+) ∨ **AX**(cr3- , rr4-) ∨ **AX**(ra4- , cr4-)]

These temporal formulas are only considered input-output order relations by our proposed method. Moreover, note that no **AF** temporal operator is used.

Table 1. Verification results

		with our method		without our method	
Circuit name	BDD vars	Memory(KB)	Time(secs)	Memory(KB)	Time(secs)
C-element4	19	4355	0.11	4355	0.1
C-element16	217	5283	0.32	5283	0.39
queue4	55	4430	0.11	4430	0.12
pipeline4	69	4457	0.14	4507	0.19
pipeline10	141	5424	0.51	5551	0.82
pipeline20	261	15207	34.45	15358	125.42
dme1	359	6397	0.59	6430	0.68
dme2	369	8408	1.29	8445	1.35
abp4	67	5184	0.57	5811	1.12
pci	129	6634	0.98	6727	1.08

5. Verification results

We verify some asynchronous bench marks shown as table.1. All these circuits are performed on an Intel Pentium-II 450Mhz processor with 388Mb of RAM under Vine Linux 2.1. Table.1 lists the results of such a comparative circuit. For each circuit, we report the number of boolean variables necessary to represent the corresponding model("BDD vars"), and memory and time required by the systems to analyze the model. In the table, some circuits can be found in the distribution of NuSMV[7]. Here, pipeline4 is an asynchronous pipeline

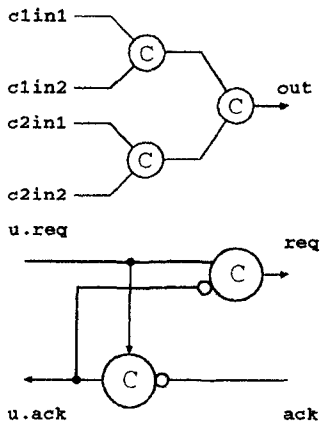

Figure 3. Verified circuits: (upper) C-element4, (lower) a queue module.

shown as fig.2. The number ♯ in pipeline♯ is the number of stages, for example, pipeline10 consists of 10 stages. C-element4 consists of 3 Muller C elements with 4 inputs and 1 output shown as fig.3(left), and queue4 consists of 4 queue modules shown as fig.3(right). For small circuits such as C-element4, queue4 and pipeline4, the memory is the same and time is no great difference between the two methods, although use of proposed method. On the other hand, as circuits are more and more large, the effects appear from the results. Especially pipeline modules are remarkable.

6. Conclusions

In this research, we proposed method that temporal formulas can be obtained inductively for specifications in model checking. Users must generally know well temporal specification because the specification might be complex. Our proposed method can be gained temporal formula specifications inductively. We aimed at input-output order relations for circuits (or systems), no considering output-input order relations. Furthermore, we defined strong/weak temporal order relations in the procedure of specification. Weak temporal order relation is to be included orders of inputs implicitly. Strong temporal order relation is to be expressed inverse input-output order relations. And the verification tasks are reduced for memory and time with our proposed method. For the future works, we will only consider strong temporal order relations, then we will check structures of complex systems with the relations.

Acknowledgement

We would like to thank the Foundation on University of the Ryukyus for taking part in ITC-CSCC2002.

References

[1] E.M. Clarke, O. Grumberg, and D. A. Peled: *Model Checking*, MIT Press, 2001.

[2] M. Huth and M. Ryan: *Logic in Computer Science*, Cambridge University Press, 2001.

[3] M. Dwyer: *Model Checking Software*, Springer, 2001.

[4] T. Kropf: *Introduction to Formal Hardware Verification*, Springer, 1999.

[5] Chikatoshi Yamada, Yasunori Nagata and Zensho Nakao: "Efficient Verification of Asynchronous Circuits Exploiting Temporal Order-Based Reduced-STG," *Proc. of 2001 International Technical Conference on Circuit/Systems, Computers and Communications* Vol. II, ITC-CSCC2001, pp.965-968, July 2001.

[6] Tam-Anh Chu, "Synthesis of self-timed VLSI circuits from graph-theoric specifications," In *Proc. International Conf. Computer Design (ICCD)*, pp.220-223. IEEE Computer Society Press, 1987.

[7] `http://sra.itc.it/tools/nusmv/` .

[8] `http://www-2.cs.cmu.edu/~modelcheck/` .