TELKOMNIKA, Vol. 11, No. 9, September 2013, pp. 5119~5125
ISSN: 2302-4046

Timed Behavioral Specification in Globally Asynchronous Locally Synchronous systems

Yu Tonglan*[1], Liu Jie[2], Zhang Juan[3], Wu Qujing[4]
School of Computer Science and Technology, University of South China
HengYang, HuNan, 421001
* e-mail: Ytonglan@163.com

Abstract

In this paper, we propose a PolGALS language for safety critical GALS(Globally Asynchronous Locally Synchronous) systems. The formal syntax and semantics are given and its compilation and implementation are defined. The language is based on timed CSP (communicating sequential process) style rendezvous between clock domains, aiming at modelling the timed behavioral of safety critical GALS systems. PolGALS is used to design timed behavioral pattern to implement timing requirements, e.g. delay, timeout, deadline, timed interrupt, etc. PolGALS provides a mechanism for implementation of timed behavioral pattern and potential for their formal verification because it is based on a PolGALS model of computation and its formal semantics.

Keywords*: PolGALS language, Semantics, Syntax, Timed behavioral, Globally Asynchronous Locally Synchronous systems*

1. Introduction

Safety critical GALS systems should have higher reliability to ensure not to be deployed into applications which have relatively rigid performance requirements, such as traffic control, health care and automotive safety systems. Further, these systems must be strengthened to meet the request of stringent fail-safe reliability to avoid economic, human or ecological catastrophes resulted in by failure in those applications. An important feature of these systems is the ability to provide continual and timely response to unpredictable changes of the state of the environment [1][2]. Safety critical GALS systems will not be operated in a controlled environment, and must be robust to unexpected conditions and adaptable to subsystem failures.

The basic problem of designing safety critical GALS system is the unentirely predictable physical world. Time is an important aspect of system specification and therefore specification language support for time is needed. By decoupling the specification from implementation and using formal mathematical models of computation for specification, GALS systems can be performed with fast simulation and efficient synthesis. The complex GALS systems can be modeled as a hierarchical composition of the simpler models of computation. Some of these simpler models of computation have finite state, such as types of finite state machines, dataflow models and synchronous/reactive models. Thus the analysis of the system can be performed at compile-time.

System develop languages are suitable for complex GALS systems, generally considered as a set of communicating processes, which have hard real-time constraints and communicate synchronously or asynchronously. System develop languages can be classified into two separate types, formal and informal. Formal languages are based on rigorous mathematical foundations, such as Esterel, CRP, SHIM and CRSM [3][4][5], which are suitable for formal verification and compilation. Informal languages are harder to compile and more difficult to be verified for lack of formal semantics, for example, SystemC [6]. However, the System develop Languages, which hides details of time constraints in the language, has failed because no widely used language expresses timing properties. High-level requirements in real-

Received January 18, 2013; Revised June 4, 2013; Accepted June 17, 2013

time systems are often stated in terms of deadline, time out and timed interrupt.

The authors' opinion is that the system-level designing language should specify time constraint. A new specification language named PolGALS is proposed which provides higher reliability and portability. The paper is organized as follows: Section 2 gives motivation PolGALS program, highlighting the main features of the language. Section 3 decribes the PolGALS program syntax. The formal semantics and MoC are presented in Section 4 and the compilation procedure is explained in Section 5. Finally, the paper ends with conclusions and future research.

2. Motivation and Related Work

To figure out the status of the current time languages and motivate the introduction of the new language, the comparison of some well-known approaches is discussed in the following. Many system level languages separate control and data-driven computations and attempt to insert timing features into system level programming languages. Their syntax and semantics are more suitable for designing digital circuits, but the timing description of capacity is limited. Lee summarized timing features in program [7]. Much earlier, Modula-2 [8] gives control over schedule of co-routines, which makes it possible for programmers to exercise some coarse control over timing. The synchronous languages have no explicit timing constructs, but their predictable and repeatable approach to concurrency can yield more predictable and repeatable timing than most alternatives [9], such as Esterel, Lustre and Signal, so that they are limited by the underlying platform. Ada can not express timing constraints. Real-Time Java augments the Java model with a few ad-hoc features that reduce variability of timing [10]. Real-time Euclid expresses process periods and absolute start times [11]. Time C introduces extensions to specify timing requirements based on events, with the objective of controlling code generation in compilers to exploit instruction level pipelining [12]. Rather than new languages, an alternative is to annotate programs written in conventional languages. Lee gives taxonomy of timing properties that must be expressible in such annotations [13]. Munzenberger gives annotations for SDL to express real-time constraints [14].

That these languages separate control and data-driven computations leads to low adoption rate by Software developer. New PolGALS is designed by using a syntax and design flow similar to general purpose programming languages, such as Java and C++. PolGALS extended SystemJ [15] itself with time statements. PolGALS is different from SystemJ, which can specify high-level time requirements for real-time such as delay, deadline, time out, and timed interrupt. Such tight integration of control, data-driven and time operations reduces the source code size and provides a much more powerful abstraction for describing large and complex models. PolGALS is designed for a general purpose GALS and does not target towards embedded systems alone, unlike many current system-level languages. It is capable of modeling multiple clock systems, particularly those can be modeled by a GALS MoC, and by using both pure Synchronous Reactive and GALS MOC. The compilation of PolGALS targets towards generating efficient and portable code, in this case Java code, which can be executed on different computing platforms. With Java as its compilation result, PolGALS is much more portable compared to any other current system level language. The portability of PolGALS allows designers to program PolGALS systems on a desktop and then performs automated deployment on other target architectures, such as embedded systems, without need for recompilation.

3. PolGALS Language Syntax

PolGALS not only combines features from SystemJ, timed CSP [16] with the Java programming language, but also combines the GALS reactive model of computation of SystemJ with timed CSP. Fig1 illustrates an example of PolGALS program model with two clock-domain.

A PolGALS program consists of a number of reactions, which follows the rules of synchronous reactive model of computation. Each reaction executes in lock-step with a logical clock called "tick" in the same clock domain. Reactions exchange and synchronize by using CSP style rendezvous between clock domains. Reaction interacts with its environment through a set of input and output signals and operations on these signals. Every clock domains samples inputs from the environment, reacts to these inputs instantaneously and produces the outputs

back to the environment, thereby implementing a state machine. The synchronous, asynchronous, reactions and operations on signals, clock-domains and channels and statements are together responsible for the control-flow of PolGALS programs.

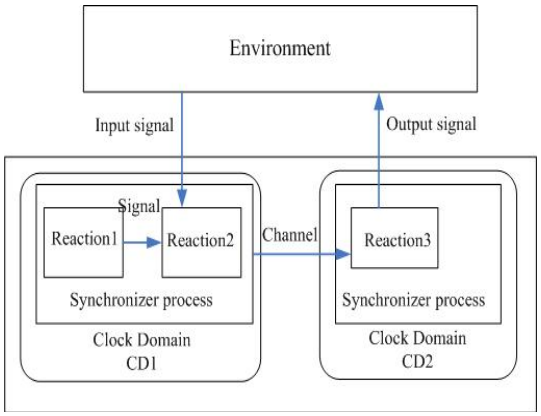

Figure 1. An example of PolGALS program model

PolGALS kernel statements consist of synchronous, asynchronous and time. Table 1 and II show the PolGALS synchronous and asynchronous kernel statements, which are similar with SystemJ [17]. The ; operator makes two synchronous reactive statements execute sequentially. Abort and suspend are based on signals, while trap is the same as Esterel's exception mechanism. Working as a watchdog, the abort statement preempts a program if the the watchdog's signal status is true in any given tick. There are Several kinds of abort statements. A weak abort statement allows the program to finish a single tick before preempting. A strong abort immediately terminates the program when signal status is true. These two kinds of abort can be further combined with an immediate signal predicate. An immediate abort starts checking the signal from the very first tick, while a non immediate abort always checks the signal after the first tick has passed. The suspend statement is similar to the abort statement. The difference is that the suspend statement pauses in the presence of signal S instead of preemption, The trap statement is a control flow based preemption statement similar to those found in modern imperative languages like Java or C++. When the exit statement is executed in the trap body, the program execution terminates immediately. The while statement means an infinite iteration statement in PolGALS program. The synchronous parallel operator || runs two or more reactions concurrently. All reactions forked by the || operator start and finish together. The synchronous statements consist of the jterm(p) statements, which are usually used for data-driven computations. Data-driven computation provides a much more powerful programming abstraction because it can be freely mixed with reactive control flow.

As shown in Table II ,the asynchronous statements represent the asynchronous MoC implemented in PolGALS. Signals are basic means of communication which have a status and possibly a value in PolGALS program. Signals can be either local or interface qualified with either the input or the output. The first two kernel statement is used to declare channels, including the output channel to send data across clock-domains and the input channel to receive data from a corresponding input port. These output and input ports of channels are initialized internally at channel declaration. An output port is used by the send statement to send data, while the input port is used by the receive statement to receive data. The send and receive statements are used to synchronize and transfer data over channels. Rendezvous in PolGALS is closely related to the scheduling of clock-domains and the asynchronous operator(><). PolGALS uses rendezvous for synchronization between sending and receiving reactions.

Table 1. Kernel Statements and Their Description

Synchronous Kernel Statements	Description
;	dummy statement
pause	dummy statement
[output] [input] [type] Signal S	signal declaration
emit S(exp)	signal emission
p1;p2	sequential statement
while (true) {p}	infinite loop
present (S) {p1} else {p2}	conditional statement
[weak] abort ([immediate] S) {p}	preempt watchdog
[weak] suspend ([immediate] S) {p}	halt watchdog
trap (T) {p}	trap exception mechanism
exit T	exit from trap
p1 || p2	parallel statement
Jterm(p)	Java data-driven
jtermp (p)=#s	obtaining a signal value
jtermp (p)=#c	obtaining a channel

Table 2. Asynchronous Kernel Statements and Their Description

Asynchronous Kernel Statements	Description
Output [type] channel C	sending channel declaration
Input [type] channel C	receiving channel declaration
p1><p2	asynchronous parallel
send C ([exp])	send data on channel
Receive C( )	Receive data on channel

The time statements are shown in table 3. The wait[d] is reaction p1 wait for exactly d time units. The p1 timeout[d] p2 is used to declare the first observable event of reaction p1 shall occur before d time units elapse, since the process starts. Otherwise, the reaction p2 takes over control after exactly d time units elapse. The p1 interrupt[d] p2 present reaction p1 interrupted reaction p2 behaves exactly as p1 until d time units elapse, and then p2 takes over control. The p1 deadline[d] constrains p1 to terminate before d time units.

Table 3. Time Kernel Statements and Their Description Asynchronous

Time kernel statements	Description
wait[d]	delay
p1 timeout[d] p2	timeout
p1 interrupt[d] p2	interrupt
p1 deadline[d] p2	Deadline

4. PolGALS Language Semantics

Semantic rules similar to Esterel and SystemJ are used for describing PolGALS time statement semantics. Given p and an input event E, the behavioral semantics essentially describes what happens in a single reaction of the module for the given input event. The things that can happen are emission of signals and control flow from one set of points to another. This is captured by a mathematical relation, called the transition relation,which is described as following semantic rule : $\mathrm{p}, \mathrm{data} \xrightarrow{\mathrm{K},e,t,E} p', data'$.

The rule represents the antecedent and consequent states of p, respectively, during a micro-step transition. p and p' are the statement before and after the reaction. Term data represents the state value which related with p before transition and data' represents that after transition. The variable value can be omitted if p can be executing pure control statements. Term E presents input event which is the status set of all the signals used in reaction p, but declared somewhere else. The array indexing notation to refer to signal statuses is utilized in the event set E. Term e represents the signals emitted during transition and if none are emitted then it takes the value of ⊥. Term K represents the termination code. When p is capable of generate a termination , K is an integer value. Three terms can do it. Term ; is encoded with 0, pause with 1, and exit T with an integer greater than or equal to 2. If a signal dependency has not yet been resolved. synchronous parallel reaction can produce a termination code of ∞.

Besides the above mentioned terms, K has a value of ⊥ i.e., unknown, which means p does not generate a termination code after this transition. Term t denotes a transition of t time units elapsing. In the following rules, we present the firing ones which are associated with the timed statement.

(1) The wait[d] of semantic rule as:

(I) if t ≤ d then

$$\text{Wait}[d], \text{data} \xrightarrow{\text{K},e,t,E} Wait[d-t], data'$$

This expresses that the process may be idle for any amount of time when it is less than or equal to d time units.

(II) if t ≤ d then

$$\text{Wait}[0], \text{data} \xrightarrow{\text{K},e,t,E} skip, data'$$

This expresses that the process terminates immediately after d becomes 0.

(2) The timeout[d] of semantic rule as:

(I) if e output by p1 then

$$\text{p1 timeou}[d], \text{data} \xrightarrow{\text{K},e,t,E} p1', data'$$

This expresses that if an output event e can be engaged by p1, then p1 timeout[d] p2 becomes p1'.

(II) if t < d then

$$\text{p1 timeout}[d]\, p2, \text{data} \xrightarrow{\text{K},e,t,E} p1' \text{timeout}[d-t]\, p2, data'$$

This expresses that if p1 may idle for less than or equal to d time units, so is the composition.

(III) if t=d then

$$\text{p1 timeout}[0]\ \ p2, \text{data} \xrightarrow{\text{K},e,t,E} p2, data'$$

This expresses that when d becomes 0, p2 takes over control by a silent transition.

(3) The interrupt[d] of semantic rule as:

(I) if t<d then

$$\text{p1 interrput}[\text{d}]\, p2, \text{data} \xrightarrow{\text{K},e,t,E} p1 \text{interrupt}[d-t]\, p2, data'$$

This expresses that if P engages in event x , P interrupt[d]Q becomes P0 interrupt[d]Q.

(II) if t=d then

$$\text{p1 interrput}[0]\, p2, \text{data} \xrightarrow{\text{K},e,t,E} p2, data'$$

This expresses that if P may idle for less than or equal to d time units, so is the composition. When d time units elapse, Q takes over by a transition.

(4) The deadline[d] of semantic rule as:

(I) if t ≤d then

$$\text{p1 deadline}[0]Q, \text{data} \xrightarrow{\text{K},e,t,E} p1', data'$$

Intuitively, these rules express that p deadline[d] behaves exactly as p except that it must terminate before d time units.

5. Compile into Java

The basic technology of PolGALS compiler is the GRC compiler approach used to compile SystemJ [17] and Esterl [18], which consists of the structural information and the operational. The structural information is preserved by the hierarchical state graph (HSG) , and the operational is explicitly represented by the control flow graph (CFG).

The HSG includes four different types of nodes called thread nodes, parallel nodes, compound nodes and boundary nodes. It outlines the structure of the synchronous reactions in terms of statements and threads of control. Thread nodes abstract the sequential composition of basic instruction inside a particular thread and parallel nodes represent concurrent threads. Meanwhile, loops, aborts and other back tracking statements are represented by compound nodes and pauses are represented by boundary nodes. The number of branches extending from a thread node indicates the number of possible states of that thread.

The CFG consists of synchronous and asynchronous nodes which are used for the code generation stage. There are seven types of synchronous nodes. The action nodes represent signal emission, enter nodes state encoding. On the other hand, test nodes represent signal tests and switch nodes indicate state selection. The fork nodes and join nodes define concurrency. Terminate nodes mark the completion code of a given thread. The sequential statements are encoded using action nodes, which are completely ignored in the HSG. The asynchronous nodes are two types. The a-fork nodes and a-join nodes express asynchronous concurrency respectively.

The AGRC intermediate format is well suited for systemJ languages, i.e., GALS programs. But it cannot express timed behavior characteristic. The AGRC format is insufficient, thereby AGRC further enhances GALS with timed behavior coupling. So we add new types nodes in order to make the new compiler suitable for compiling timed MoC, which is called TAGRC. The timer nodes represent time consumption. The resulted Timed Asynchronous GRaph Code (TAGRC) format derived directly from PolGALS semantics is the intermediate representation which PolGALS is translated into before being translated into low level Java back-end code.

The whole process of compiling PolGALS into Java code is divided into four phases as shown in Figure 2.

(1) Abstract syntax tree generation: syntactic analysis and error check includes syntactic error and front-end error are performed in this stage.

(2) Format translation: PolGALS program is translated into the TAGRC format. The structural translation rules are followed to translate each statement into one or multiple nodes of TAGRC.

(3) Back-end code Generation: The backend code generation stage is carried out on the resulting AGRC.

(4) The code optimization: some well-known algorithms like redundancy elimination and information propagation are used in this implement stage.

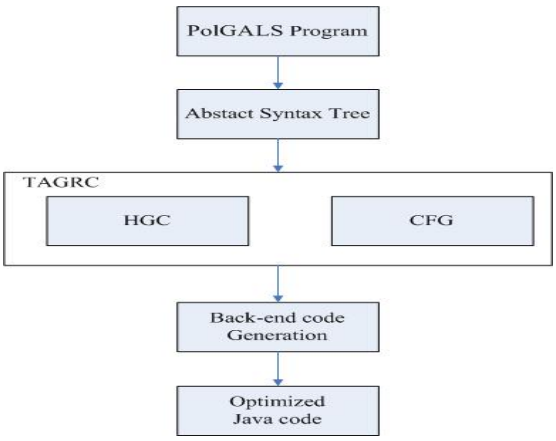

Figure 2. Compiling PolGALS

6. Conclusion

PolGALS language based on the GALS of computation combines the asynchronous features of System J with timed behavior of timed CSP and data computation capabilities of Java. We make a detailed description of MoC of PolGALS language, as well as the timed behavior semantics of the language, which are suited for compiler construction. PolGALS language utilizes timer for time consuming between coupled reactions. The Timed Asynchronous Graph Code (TAGRC) format is then proposed which is better suited for PolGALS language since it is based on a set of formal semantics, and thus is potentially easier to be verified. The TAGRC compiler provides better performance in our benchmark tests and different execution platforms, which in turn provides direct support for the active and timed behavior statements within PolGALS language.

Currently, PolGALS timed behavior model of computation is difficult to be verify because it opens a wide variety of verification and optimization techniques available in the program modeling domain. A Further study on verification techniques about timed behavior model needs to be carried on in the future work.

Acknowledgements

This work is partially supported by the projects funded by Science and technology project of Hunan Province (Grant No. 2011GK3192), Key project of Hunan Province scientific research of colleges and Universities (Grant No. 11A105).

References

[1] Junsuo, QU. Design of Time Synchronization Method for Real-Time EPON. *TELKOMNIKA Indonesian Journal of Electrical Engineering*. 2013; 11(7):
[2] Qing-Quan Liu. Coordinated Motion Control of Autonomous and Semiautonomous Mobile Agents. *TELKOMNIKA Indonesian Journal of Electrical Engineering.* 2012; 10(8):1929-1935.
[3] Berry G. *The Esterel v5 language primer version 5_9*. Ecole des Mines de Paris, CMA, INRIA. 2000.
[4] Ramesh S. *Communicating reactive state machines: design, model and implementation.* Proceedings of the IFAC Workshop on distributed computer control systems. Pergamon Press. 1998; 9:105-110.
[5] Tardieu O, Edwards SA. *Scheduling-independent threads and exceptions in SHIM.* Proceedings of the 6th ACM and IEEE international conference on embedded software. NewYork, NY. 2006: 142–151.
[6] Grötker T, Liao S, Martin G, et al. *System design with SystemC. Kluwer Academic Publishers*, USA, 2002.
[7] Edward A Lee. Computing needs time.*Communications of the ACM*. 2009; 52(5): 70-79.
[8] Wirth N, Gries D. *Programming in Modula-2*. Springer-Verlag, New York. 1983.
[9] Benveniste A, Berry G. *The synchronous approach to reactive and real-time systems.* Proceedings of the IEEE. 1991; 79(9): 1270-1282.
[10] Burns A, Wellings A. *Real-Time Systems and Programming Languages: Ada 95, Real-Time Java and Real-Time POSIX.* Addison Wesley, 3d edition. 2001.
[11] Klingerman E, Stoyenko AD. Real-time Euclid: A language for reliable real-time systems. *Software Engineering, IEEE Transactions.* 1986; 12(9):941-949.
[12] Leung A, Palem KV, Pnueli A. *TimeC: A time constraint language for ILP processor compilation*. Technical Report TR1998-764. New York University. 1998.
[13] Lee I, Davidson S, Wolfe V. *Motivating time as a first class entity*. Technical Report MS-CIS-87-54. University of Pennsylvania Aug. 1987.
[14] Münzenberger R, Dörfel M, Hofmann R, et al. A general time model for the specification and design of embedded real-time systems. *Microelectronics Journal*. 2003; 34(11): 989-1000.
[15] Malik A, Salcic Z, Roop PS, et al. SystemJ: A GALS Language for System Level Design, *Computer Languages. Systems and Structures.* 2010; 36(4): 317-344.
[16] Roscoe AW. *Understanding concurrent systems*. Springerverlag London Limited. 2010.
[17] Malik A. *Principia Lingua SystemJ*. PhD thesis, The University of Auckland New Zealand. 2010.
[18] Potop-Butucaru D. *Optimisations for faster execution of esterel programs.* PhD thesis, Ecoledes Minesde Paris. 2002.