논·문

Petri Net을 사용한 임베디드 시스템의 스케쥴링
(Scheduling of Embedded System Using a Petri Net)

이태용
한국전자통신연구원 이동통신연구단 무선시스템연구그룹
휴대인터넷 시스템 연구팀
grasshopper@etri.re.kr

요 약

Embedded systems specification usually has both data computation and control structures. Control structure can be two types. One is data dependent control and the other is real time control. The first one can be easily solved because it is involved with only one task and its behavior is very obvious. The problem is the second case. We must use some sophisticated techniques to resolve concurrent behavior of tasks.

To solve the second problem, these tasks require to be scheduled on a shared resource such as processor and memory. This scheduling is mainly based on the system specification. To simulate dynamic behaviors of system, it is very difficult to predict these behaviors because these behaviors have to be determined at compile time and decisions have to be made at run time. In other words, scheduling must be efficiently made while pleasing real time constraints and using the processor and memory resources as efficiently as possible. Therefore, Quasi-static scheduling algorithm is used to solve these problems.

1. Introduction

Software and its integration with the hardware are playing a very important role in embedded systems. So, they become one of the most important costs. Because of this reason, recent design flows help the designers start with good functional and implementation-independent specifications of the systems so that many heterogeneous systems can be used with their software. However, implementing software or integration with hardware is not that simple. Shared resources such as a processor and memory have to be precisely controlled. Here scheduling takes places to control these resources.

There are three types of scheduling to consider. The first is called a static scheduling technique. This static scheduling technique is used when a single task is running. This technique is considered to be the simplest one because the next schedules are predictable. Another one is a quasi-static scheduling technique used when the specification has only data dependent control in addition to data computation. In data dependent control, the schedules can be revealed at compile time while computing data. The third one is dynamic scheduling technique. This is used when the specification has also real time controls. Real time controls heavily depend on the external information. Therefore, it is not predictable.

The first one can be easily solved because it is involved with only one task and its behavior is very obvious. The problem is the second and third case. But only the second one will be considered in this paper. The quasi-static scheduling must use some sophisticated techniques to resolve concurrent behavior of tasks.

Therefore, I will explain clearly how to use Quasi-static scheduling (QSS) algorithm that is a proposed solution for this problem, and will explain on other possible solutions on this case. This Quasi-static scheduling algorithm uses Free Choice Petri Nets (FCPNs).

Several techniques for this problem have been proposed. Buck and Lee [3] have introduced a technique called the Boolean Data Flow (BDF) networks model and explained a new algorithm for computing a quasi-static schedule. Thoen et al. [2] proposed another technique that uses static information. Lin [4] proposed another algorithm that creates a software program from a concurrent process specification. These three will be explained in the next

2. Buck and Lee (Boolean Data Flow Networks Model)

Buck and Lee explained about an analytical model of the behavior in data flow graphs using data-dependent control flow [3].

They used Tagged-token data flow machines that were created to overcome some of the problems in static data flow machines. Figure l.4 shows the static data flow machines. [3]

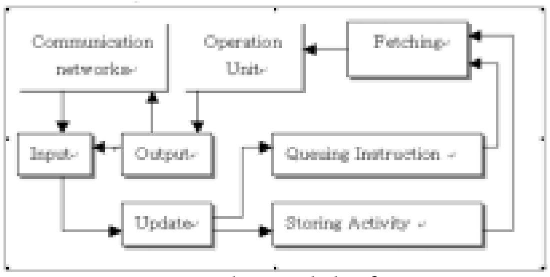


Figure 1.4 A simple model of a processing element for a static data flow machine

The goal of static data flow machines is to help the execution of loop iterations and function/procedure invocations concurrently. To make this thing work, data values are carried by tokens that include a three-part tag. The first field of the tag contains the context, according to the current procedure call. The second field of the tag contains the iteration number that is used when loop iterations are executed concurrently. The third field contains the information about activity, according to the appropriate node in the data flow graph. [3]

This scheduling technique has some shortcomings. Boolean Data Flow Networks Model with bounded memory cannot be decided. Any algorithm may fail even though the Boolean Data Flow Networks Model is schedulable. Therefore, this algorithm proposed by Buck and Lee can solve problems in special cases. [1]

3. Thoen

This technique uses static information from the specification and finds statically schedulable groups of threads from a constraint graph description of the system. The problem of this technique is that this technique does not depend on a formal model and does not describe the problems about evaluating whether or not the specification is schedulable. [2]

4. Lin

This algorithm creates a software program from a concurrent process specification using a Petri-Nets representation [4]. This algorithm is used on the assumption that the Petri-Nets is reliable. This means buffer can only store one data at a time. This algorithm guarantees algorithm termination. Also it guarantees handling multi-rate specifications.

The best aspect of this algorithm is that it always guarantees scheduling for the models. However, this approach did not include the possibility of the Petri-Nets where source and sink transitions are modeling the interaction with the environment. For this reason, this algorithm cannot specify inputs with independent rates.

5. Sgroi

From here, a new algorithm will be considered, proposed by Sgroi [1]. This algorithm takes a Petri Nets model for the inputs and produces outputs as a software implementation that has a set of software tasks. This set of software tasks is called at run-time by Real-Time Operation System (RTOS) [1].In order to find out tasks and synthesize the code for each task, it is important to consider the parts of specification with data computation and data-dependent control. And then quasi-static schedule will be computed.

6. Petri-Nets

Sgroi chose a Petri-Nets as base formal model. The reason is that Petri-Nets can express concurrency, non-deterministic choice, synchronization and causality [1]. Another reason is that most properties as well as schedulability can be determined for Petri-Nets. Data computations are represented as a type of nodes (places) and non-FIFO channels between computations units are represented as another type of nodes. Data-Dependent control is modeled by choice, which is nothing but just places, with several output exchanges, one for every possible solution for the control. Data are modeled as tokens passed by transitions through places [1].

A Petri-Net is a triple (P, T, F), where P is a non empty finite set of places, T a non-empty finite set of transitions. A Petri-Net graph is a representation of a Petri-Net as a bipartite weighted directed graph. [1]

7. Free Choice Petri-Nets

Sgroi suggested a notation of shedulability for Free Choice Petri-Nets. A Free Choice Petri-Nets is quasi-statically schedulable when there is a cyclic finite sequence that produces the tokens of the net to their initial places for every possible solution of the control at the choice places [1]. Sgroi suggest an algorithm that first evaluates schedulability of the net to find out whether or not the specification is correct. If the net is schedulable, this algorithm calculates a quasi-static schedule by splitting the net into schedulable sub-components. After that, this algorithm provides a software implementation by go over the schedule and switching transitions with the appropriate code. Figure 2 shows Free Choice Net and not Free Choice Net.

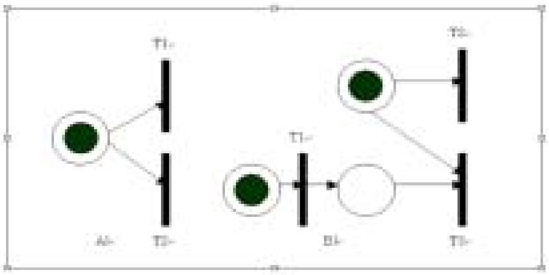
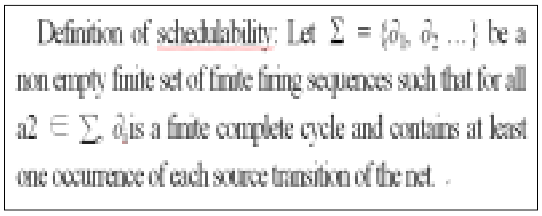
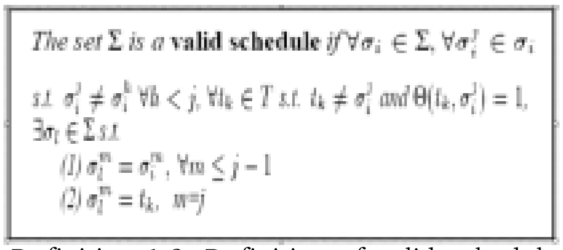


Figure 2: Free Choice Net (A), Not Free Choice Net (B)

8. Quasi-static Scheduling of Free Choice Petri-Nets


Definition of schedulability: Let $\Sigma = \{\partial_1, \partial_2 \ldots\}$ be a non empty finite set of finite firing sequences such that for all a2 $\in \Sigma$, ∂_i is a finite complete cycle and contains at least one occurrence of each source transition of the net.

Definition 1.1: Definition of schedulability


The set Σ *is a* **valid schedule** *if* $\forall \sigma_i \in \Sigma, \forall \sigma_i^j \in \sigma_i$ s.t. $\sigma_i^j \neq \sigma_i^h \; \forall h < j, \forall t_k \in T$ s.t. $t_k \neq \sigma_i^j$ and $\Theta(t_k, \sigma_i^j) = 1$, $\exists \sigma_l \in \Sigma$ s.t.
(1) $\sigma_l^m = \sigma_i^m, \forall m \leq j-1$
(2) $\sigma_l^m = t_k$, $m=j$

Definition 1.2: Definition of valid schedule

This definition of schedulablitiy can be extended to non-static Data Flow networks. When the net contains non-deterministic choices that model data dependent structures like if-then-else or while-do, a valid schedule is a set of finite complete cycles, one for every solution of non-deterministic choices. Shedulability also includes a meaning that the existence of at least one valid schedule that make sure there is no unbounded accumulation of tokens in any place

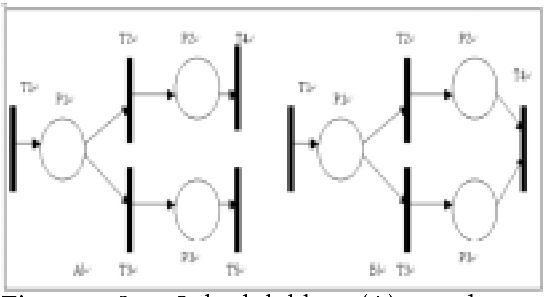


Figure 3: Schedulable (A) and not Schedulable (B) Free Choice Petri-Nets

In order to find a valid schedule, it is necessary to find a valid group of finite complete cycles that net is first split into as many conflict free sub-components as the number of possible resolutions for the non-deterministic places. Now, each of the sub-components is to be scheduled statically. When all the components are schedulable, a valid schedule is a set of the finite complete cycle. If only one sub-component is not schedulable, then the whole net is not schedulable. The next figure 3 shows the Schedulable and not schedulable Free Choice Petri-Nets. F(s) in A) is a(1,1,0,1,0) + b(1,0,1,0,1), where a, b = 0, 1, 2 ⋯ while F(s) in B) is (2,1,1,1) which is valid, (2,2,0,1) which is unbounded, and (2,0,2,1) which is also unbounded.

The benefits of using this technique are explained below. First, Quasi-Static Scheduling minimizes the execution runtime overhead much more than dynamic scheduling does, since it finishes a lot of work at compile time. And then, schedulability is decidable due to the computation model of Free-Choice Petri-Nets. Finally, it is quite easy to create a complete algorithm to solve the scheduling problem for any Petr-Net that is quasi-statically schedulable.

9. Conclusion

In this paper, several techniques are presented for scheduling of embedded system using a Petri Nets. As I mentioned in the beginning, there are three types of scheduling to be considered. But only the quasi-static scheduling technique was considered in this paper. And then, some sophisticated techniques were presented to resolve concurrent behavior of tasks.

Four different techniques are explained. One is Buck and Lee's Boolean Data Flow Networks Model. The second one is proposed by Thoen. The third one is proposed by Lin. Sgroi suggested another techniques using Free Choice Petri-Nets. Each of these techniques has cons and pros but I personally like the solution suggested by Sgroi. Again, finding solutions for quasi-static scheduling is an important problem in this field. I believe using Petri-Nets is very important to find these solutions because determining whether the net is schedulable can be clearly revealed only with a tool like

Petri-Nets and . After that, Petri-Nets can also help to schedule each component correctly. Therefore, using a Petri Nets is one of the best solutions for scheduling of Embedded System.

감사의 글

이 논문을 적는데 많은 도움을 주신 아래 참고문헌에 기재된 모든 분들께 감사의 말씀을 드립니다.

참고문헌

[1] M. Sgroiy. Synthesis of Embedded Software Using Free-Choice Petri Nets. UC Berkeley, 1999

[2] F. Theoen et al. Real-time multi-tasking in software synthesis for information processing systems. In Proceedings of the International System Synthesis Symposium, 1995.

[3] J. Buck. Scheduling dynamic data flow graphs with bounded memory using the token flow model. Ph. D dissertation. UC Berkeley, 1993.

[4] B.Lin. Software synthesis of process-based concurrent programs. In Proceedings of the Design Automation Conference, Jun 1998.

이태용

1997년~2001년 Southern Illinois University Computer Science, US (학사).
2001년~2002년 Dept. of Parks, St. Louis, US, (Web search engine programmer).
2002년~2004년 Southern Illinois University Computer Science, US (석사).
2004년~현재 한국전자통신연구원 이동통신연구단 무선시스템연구그룹 휴대인터넷시스템 연구팀소속 연구원

관심분야는 Network Traffic Engineering, MPLS (Mulitprotocol Label Switching), Petri-Net, Internet Quality of Service, OFDMA, CDMA