A Schedulability-Preserving Transformation Scheme from Boolean- Controlled Dataflow Networks to Petri Nets

Cong Liu Edward A. Lee

University of California at Berkeley
Berkeley, CA, 94720, USA
{congliu,eal}@eecs. berkeley.edu

Abstract

This paper presents a transformation scheme from Boolean-controlled dataflow (BDF) networks to Petri nets (PN) that has the following properties. A BDF has a bounded length schedule, if and only if the transformed Petri net has a bounded length schedule. A BDF has a bounded memory schedule if the transformed PN has a bounded memory schedule. The existence of bounded memory schedule of BDF is proved to be undecidable. This transformation scheme provides a way to construct a bounded memory schedule of a BDF using exsiting PN scheduling techniques. In BDF models tokens are associated with data, and actors communicate through FIFO channels. A general PN semantics does not distinguish tokens and does not preserve the order of tokens in a place. The proposed method uses different places to hold the Boolean tokens with different values. The ordering of Boolean tokens at each control input port is preserved by enforcing that new Boolean tokens are not produced until the previous Boolean token is consumed. Initial Boolean control tokens are handeled by use of a preamble. Furthermore, based on the transformation, the consistence notion originally defined in BDF is extended to PN.

1 Introduction

The ever-increasing complexity of embedded systems require designers to use formal models to precisely describe system behaviors and hide implementation details. Concurrent models, such as dataflow networks [6], Kahn process networks [4][5], and Communicating Sequential Processes [3], are often used because they explicitly express the inherent concurrency in most embedded system applications. In a concurrent model, a system is consisted of a set of processes which communicate with the environment and other processes, and run at their own speed. These concurrent processes often share a physical resource (e.g. CPU). Hence, their implementation often requires solving a fundamental scheduling problem, i.e. sequencing the operations of concurrent processes under certain constraints (e.g. bounded memory). Depending on when the scheduling decision are made, scheduling algorithms are classified as three categories. Static scheduling makes all decisions at compile-time. Quasi-static scheduling computes a static schedule as much as possible while leaving data-dependent choices resolved at run-time. Dynamic scheduling make all decisions at run-time. Static scheduling is restricted to specifications without run-time choices, such as the specifications modeled by synchronous dataflow (SDF) networks [6]. Lee and Buck extended SDF to Boolean-controlled dataflow (BDF) network [1] by introducing actors that have conditional production and consumption rate. It can model specification with run-time data-dependent choices. In fact, BDF is proved to be equivalent to a universal Turing machine. It means any program can be modeled by a BDF. However, the scheduling problem, the existence of bounded memory schedule, is undecidable. It means no algorithms can prove or disprove schedulability of an arbitrary BDF in finite time. All existing BDF scheduling techniques, such as bounded length scheduling, clustering, and state-enumeration, are based on heuristics. They can successfully construct schedules only for a subset of schedulable BDF.

This paper presents a transformation scheme from BDF to Petri Nets (PN) [7]that has the following properties. A BDF has a bounded length schedule, if and only if the transformed Petri net has a bounded length schedule. A BDF has a bounded memory schedule if the transformed PN has a bounded memory schedule. Hence, this transformation scheme provides a way to construct a bounded memory schedule by taking advantage of exsiting PN scheduling techniques.

The most recent advancement [2] on quasi-static

scheduling of PN adopts a T-invariant-guided state space search. A T-invariant is a set of instances of transitions. If they are fired, the resulting state is the same as before the firings. In this approach, the heuristic used to prune the search space considers the pre-history of a state and is not based on a local property, such as the place bound. Hence, it is more justifiable than those using simple bounding box on state space. Directly apply the PN scheduling techniques to BDF or any state enumeration based BDF scheduling algorithms have the state explosion problem, because the state of a BDF is determined not only the number of tokens on each arc, which corresponds to the state of a PN, but also the value and the order of the tokens. This problem makes direct state enumeration inefficient.

In BDF models tokens are associated with data, and actors communicate through FIFO channels. A general PN semantics does not distinguish tokens and does not preserve the order of tokens in a place. The proposed method uses different places to hold the Boolean tokens with different values. The ordering of Boolean tokens at each control input port is preserved by enforcing that new Boolean tokens are not produced until the previous Boolean token is consumed. The tranformed PN is equivalent to the original BDF with the constraint that the control arcs, which connects to control port, have a bound of exact one. This implies that the traces contained in the PN is subset of traces contained in the BDF. Note that we do not impose bounds on arcs except control arcs. So if there are more than one initial Boolean control tokens in a BDF, it may be possible that after a finite number of firings the BDF reach a state where there is at most one Boolean tokes on each control arc. Then the proposed transformation still applies. Finally the consistence notion originally defined in BDF is extended to PN. This is due to the fact there is a one-to-one correspondence between the complete cycles in a BDF and the T-invariants in the transformed PN. The transformation scheme preserves consistence.

The following of the paper is organized the following. In Section 2, we introduce basic definitions and notations of BDF and Petri nets and their scheduling. A tranformation scheme from a BDF to a Petri net is described in Section 3.

2 Preliminaries

2.1 Boolean-controlled dataflow network

A BDF extends SDF by introducing dynamic actors that has conditional inputs or output ports. These actors satisfies the following properties:

- The number of tokens consumed by an input port, or produced by an output port per firing is a two-valued function of the value of a Boolean token that is recieved by another input port (the *control port*) of the same actor. Such an input port is called a *conditional port*. One of the two values of the function is zero.
- The control port could be an output where the control token's value annouces whether there are data on the conditional port.
- Control ports are never conditional ports, and always consume exactly one Boolean token per firing.

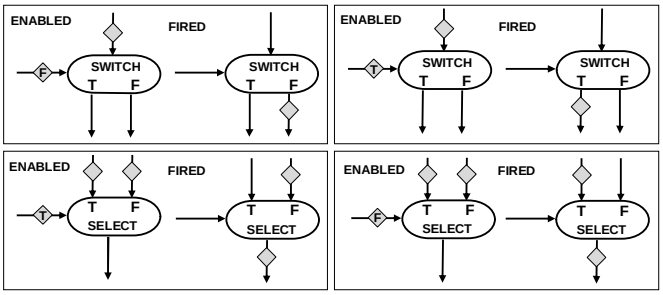

Figure 1. Structure and semantics of SWITCH and SELECT actors.

The canonical examples of such actors are SWITCH and SELECT. The structure and semantics are of the two actors are illustrated in Figure 1.

2.2 Petri nets

A *Petri net* is a 5-tuple $PN = (P, T, F, W, M_0)$, 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 a set of *arcs*; $W : F \to \{1, 2, ...\}$ is a *weight function*; $M_0 : P \to \{0, 1, ...\}$ is the *initial marking*.

Given a transition t_i and a place p_j, if $(t_i, p_j) \in F$, t_i is called an *input transition* of p_j, and p_j is called an *output place* of t_i. Similarly if $(p_j, t_i) \in F$, t_i is called an *output transition* of p_j, and p_j is called an *input place* of t_i. The set of input transitions of p_j is denoted by ${}^\bullet p_j$. The set of output transitions of p_j is denoted by $p_j^\bullet$. The set of input places of t_i is denoted by ${}^\bullet t_i$. The set of output places of t_i is denoted by $t_i^\bullet$. A *marking* is a mapping $M : P \to \{0, 1, ...\}$. It represents the state of a PN. A transition t_i is called *enabled* if $\forall p_j \in {}^\bullet t_i, M(p_j) \geq W(p_j, t_i)$. A transition may or may not fire whenever it is enabled. A *firing* of an enabled transition t_i at marking M is denoted by $M[t_i > M', \forall p_j \in {}^\bullet t_i, M'(p_j) = M(p_j) - W(p_j, t_i), \forall p_k \in t_i^\bullet, M'(p_k) = M(p_k) + W(t_i, p_k)$. A marking M_n is said to be *reachable* from the inital marking M_0 if there exists a *firing sequence* $\beta = t_i t_j ... t_k$ such that $M_0[t_i > M_1[t_j > M_2 ... [t_k > M_n$ or simply $M_0[\beta > M_n$. The incidence matrix $A = [a_{ij}]$ is a $|T| \times |P|$ matrix, where $a_{ij} = F(t_i, p_j) - F(p_j, t_i)$.

A vector $x \in \mathbb{N}^{|T|}$ is called a *T-invariant* if $A^T x = 0$. A T-invariant x is said to be *minimal* if there exists no other T-invariant $x' \neq x$ with $x' \leq x$.

2.3 Quasi-static schedules

A *quasi-static schedule* or *bounded memory schedule* is a finite list of guarded firings where:

- The system executing the schedule returns to its initial state, regardless of the outcome of the conditions or choices.
- Firing rules are satisfied at every point in the schedule.

A *bounded-length schedule* is a quasi-static schedule where the number of firings required to return the system to its initial state is bounded by a constant. If each firing is associated with execution time, bounded-length schedule can be used to estimate if a hard deadline can be met in real-time applications. Generating bounded-length schedule starts with finding the cyclic firing sequences that the system state after the firings is the same as before the firings. Such cycles must exists for all possible outcomes of conditions or choices.

3 Transformation Scheme

It is known SDF is equivalent to a special case of PN, *maked graph*. More specifically, there is a transformation scheme that one-to-one maps actors and acrs of a SDF to transitions and places of a PN. So we restrict our attension to the following BDF actor: SWITCH, SELECT, and Boolean stream generators.

3.1 SWITCH and SELECT actors

The transformations of SWITCH and SELECT actors are shown in Figure 2. If there is a *true* Boolean token present at the control port of the actors in a BDF, a token will be present in pT of the transformed PN. For the SWITCH actor, if transition A, transformed from the actor A, is fired, transition T or F is enabled dependending on whether there is a token in pT or pT, respectively. Note that pT and pT can not both have tokens, because it corresponds to possible out of order execution of the orignal BDF. It is avoided by introducing a synchronization mechanism discussed later. Presence of a token in place pTa or pFa is used to indicate that the BDF actor finishes execution. They can not both have tokens because of the synchronization mechanism.

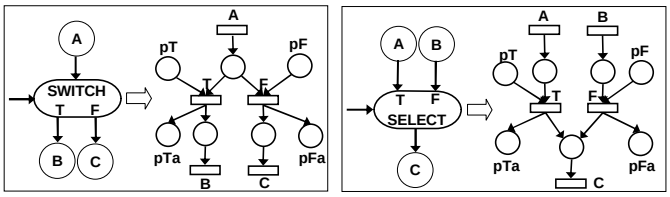

Figure 2. Transformation of SWITCH and SELECT actor to PN.

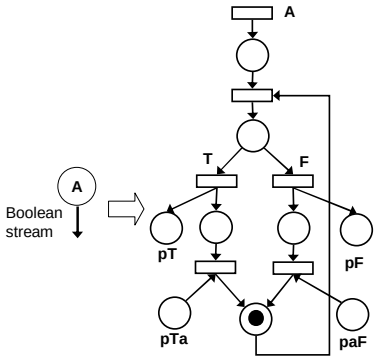

Figure 3. Conversion of Boolean stream actor to PN.

3.2 Boolean control stream generators

The transformation of Boolean control stream generator is shown in Figure 3. The core part is a synchronization and delivery machinary. It is triggered by the transition A, transformed from the actor A, and non-deterministically chooses to fire transition T or F, which corresponds to the Boolean generator generate a *true* or *false* token. The token will be routed to the corresponding place pT or pF, respectively. In order to ensure the ordering of tokens, the machinary can continue to generate new tokensonly when it recieves acknoledgement from the controlled actors. The acknolegement is repsented by the presence of a token in place pTa or pFa.

3.3 Initial Boolean tokens

It can be seen from the above transformation, we preserve the ordering of tokens by synchronization, i.e. at most one control token is present at the control port. So if there are more than one initial Boolean tokens in a BDF, the above transformation would not be able to generate a PN. However, it is possible that the initial Boolean tokens can be consumed by a number of firings of actors. That is, a BDF with initial Boolean tokens after a finite number of

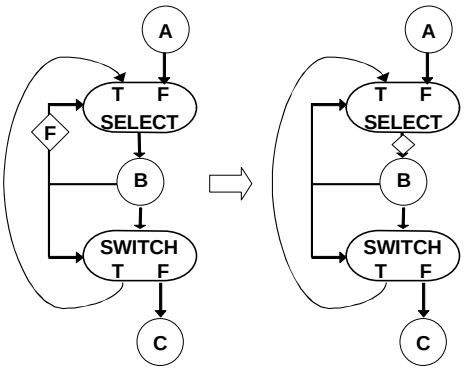

Figure 4. Transformation of BDF by use of a preamble.

firings reaches a state where there are no or one Boolean token at its control ports. For example, the left BDF in Figure 4 is transformed to the right BDF by firing actor A and $SELECT$. The finite firing sequence is called a preamble. By use of a preamble, we redefined the initial state of a system. However, if the PN transformed from the new BDF has a bounded-memory schedule, we can construct the schedule for the orignal BDF by adding the preamble as a prefix of the schedule. The rationale of imposing bounds on Boolean control arcs also comes from the fact that each firing of SWITCH or SELECT actor will consume exactly one Boolean token. So accumulating Boolean tokens at control ports will not help enabling actors.

3.4 Tansformation Algorithm

Algorithm 1 Transformation Algorithm

```
if (initial Boolean tokens > 1) then
    transform BDF by preamble;
end if
transform SDF actors;
transform SWITCH actors;
transform SELECT actors;
transform Boolean actors;
handeling initial tokens;
```

First, the BDF is checked with the number of initial Boolean tokens. If it is more than one, a preamble-based-transformation is invoked. The transformation procedure starts with replacing SDF actors to transitions, and replacing arcs with places. The SWITCH, SELECT and Boolean actors are transformed to the PN structures mentioned above. Finally, the initial

3.5 Schedulability

Proposition 1 *A BDF has a bounded length schedule, if and only if the transformed PN has a bounded length schedule.*

Sketch of the proof: $Leftarrow$, basde on the above transformation it can be shown the transformed PN is equivalent to a BDF with bounds of the control arcs to be exact one. Hence every firing sequence of the PN corresponds to a firing sequence of the BDF, but not vice versa. In other words, the PN contains firing sequences which are a subset of firing sequences of the BDF. Thus, the same schedule generated for the PN can be applied to the BDF. $Rightarrow$, the necessary and sufficient condition for a BDF to have a bounded-length schedule is: First, it must have complete cycles for all possible outcomes of conditions and choices. Second, for all possible Boolean sequences, it is possible to construct a corresponding acyclic precedence graph. We can show that the first condition is equivalent to existence of T-invariants for all possible choices. The second condition decomposes the BDF to a regular SDF. The PN can corrspondingly decompose to marked graph components. It is easy to check the equivalence of a SDF to the corresponding marked graph. □

Proposition 2 *A BDF has a bounded memory schedule if the transformed PN has a bounded memory schedule.*

Sketch of the proof: the PN contains firing sequences which are a subset of firing sequences of the BDF. Thus, the same schedule generated for the PN can be applied to the BDF. □

3.6 Consistence

A *strongly consistent* BDF is one where there exists a solution to the balance equation regardless of possible values of the symbolic variables. A *weakly consistent* BDF is one where there exist a solution to the balance equation only for some values of the symbolic variables. Weakly consistence implies the correlation between different conditions enforced by the complete cycles. For example, if two symbolic variables must have the same value so that the balance equation have a solution. It means the two conditions are either the same or opposite in a complete cycle. In a transformed PN, it means for every T-invariant x, if the *true* transition of the first choice is contained in x, the *true* or *false* transition of the second choice is contained in x.

Definition 1 *(Pairwise transition dependency)*
Given two transitions t_i, t_j of a Petri net, t_i is said to be dependent *on t_j if t_j appears in every (minimal) T-invariant containing t_i, i.e., $R_1 = \{(t_i, t_j) \in T^2 \mid \forall x \in \chi,\ t_i \in \|x\| \Rightarrow t_j \in \|x\|\}$.*

For example, the BDF in Figure 5 is weakly consistent. The symbolic variables associated with the SWITCH and SELECT actor must have the same value so that the balance equation has a solution. The transformed PN, there are two minimal T-invariants. One minimal T-invariant contains both T transitions, and the other one contains both F transitions.

4 Conclusion and future research directions

This paper presents a transformation scheme from BDF to PN that has the following properties. A BDF has a bounded length schedule, if and only if the transformed Petri net has a bounded length schedule. A BDF has a bounded memory schedule if the transformed PN has a bounded memory schedule. However, whether the schedulability of the BDF implies the schedulability of the transformed PN or not remains an open question. Future research directions includes exploration of other transformation schemes, and comparing the schedulability of the BDF and PN.

References

[1] J. T. Buck. *Scheduling dynamic dataflow graphs with bounded memory using the token flow model*. PhD thesis, University of California, Berkeley, 1993.

[2] J. Cortadella, A. Kondratyev, L. Lavagno, C. Passerone, and Y. Watanabe. Quasi-static scheduling of independent tasks for reactive systems. In *Proceedings of the 23rd International Conference on Applications and Theory of Petri Nets*, pages 80–100, 2002.

[3] C. A. R. Hoare. *Communicating Sequential Processes*. International Series in Computer Science. Prentice-Hall, 1985.

[4] G. Kahn. The semantics of a simple language for parallel programming. In *Information processing*, pages 471–475, Aug 1974.

[5] G. Kahn and D. B. MacQueen. Coroutines and networks of parallel processes. In *Information processing*, pages 993–998, Aug 1977.

[6] E. A. Lee and D. G. Messerschmitt. Static scheduling of synchronous data flow graphs for digital signal processing. *IEEE Transactions on Computers*, Jan. 1987.

[7] T. Murata. Petri nets: properties, analysis, and applications. *Proceedings of the IEEE*, 77(4):541–580, Apr. 1989.

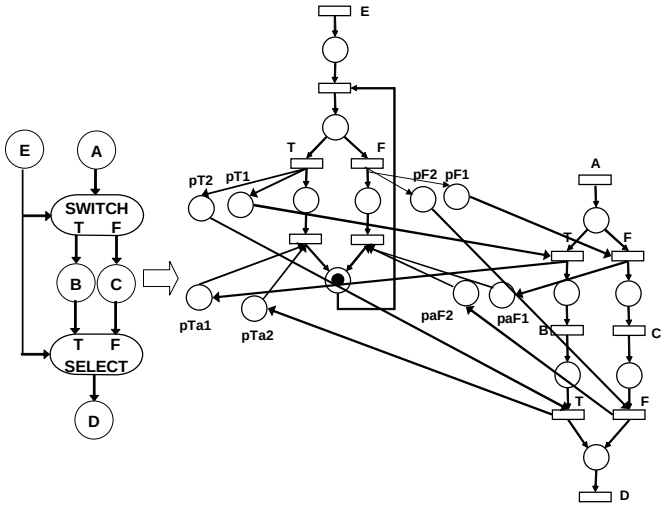

Figure 5. Conversion of if-then-else BDF to PN.