행렬구조 메모리 참조표를 사용한 페트리네트 제어기의 하드웨어 구현

Hardware Implementation of Petri Net-Based Controller with Matrix-Based Look-Up Tables

장 래 혁, 정 승 권, 권 욱 현
(Naehyuck Chang, Seungkweon Jeong and Wook Hyun Kwon)

Abstract : This paper describes a hardware implementation method of a Petri Net-based controller. A flexible and systematic implementation method, based on look-up tables, is suggested, which enables to build high speed Petri net-based controllers. The suggested method overcomes the inherent speed limit that arises from the microprocessors by using of matrix-based look-up tables. Based on the matrix framework, this paper suggests various specific data path structures as well as a basic data path structure, accompanied by evolution algorithms, for sub-class Petri nets. A new sub-class Petri net, named Biarced Petri Net, resolves memory explosion problem that usually comes with matrix-based look-up tables. The suggested matrix-based method based on the Biarced Petri net has as good efficiency and expandability as the list-based methods. This paper shows the usefulness of the suggested method, evaluating the size of the look-up tables and introducing an architecture of the signal processing unit of a programmable controller. The suggested implementation method is supported by an automatic design support program.

Keywords : PN, MG(marked Graph), look-up table, hardware, programmable controller, automatic design

I. 서론

순차 제어(sequential control)기의 언어로 LD(ladder diagram)가 많이 사용되어 왔으나, 가독성(readability), 검증성, 유지 보수성 등이 떨어지는 단점이 있으므로[1]-[4], 이러한 단점을 보완할 수 있는 연구로 PN(Petri net), 이를 기초로한 Grafecet 또는 SFC(sequential function chart) 등을 사용하여 공정을 모델링하고 해석하거나, 제어기를 설계하는 연구 또한 지속적으로 수행되었다[1][2][3][5][6]. 특히, 공정의 모델링과 검증 방법, LD와의 상호 변환 및 비교 해석[2][6][7][8], 제어기의 합성[9]과 제어기의 축소(reduction)[10]에 관한 연구가 순차 제어와 관련된 PN의 주된 연구 분야이다. 이들 연구 결과는 고성능의 순차 제어기를 구현하는데 PN이 매우 효과적임을 보여 준다.

설계된 제어기를 실제로 구현하는 연구는 제어기의 설계 및 검증에 관한 연구 못지 않은 중요한 부분이다. PN을 사용한 제어기의 구현에 관한 연구는 소프트웨어에 의한 방법과 하드웨어에 의한 방법으로 구분할 수 있다. 소프트웨어 방법은 PN을 프로그램 언어로 묘사하고 컴파일, 또는 해석(interpret)을 통해 컴퓨터에서 수행한다[11]-[15]. 소프트웨어 구현 방법에서는 복잡한 기능을 손쉽게 부가할 수 있으나, 충분한 속도를 얻지 못해 적용 범위에 제한이 생기는 경우가 있을 수 있다.

이러한 문제점을 극복하기 위해 PN을 하드웨어로 구현할 수 있다. PN을 하드웨어로 구현하는 방법으로는 하드와이어(hardwire) 형태로 구성하는 방법과[16]-[22], 메모리 참조표(look-up table)를 사용하는 방법을 들 수 있다. 하드와이어 방법은 특정 PN 모델의 구조를 게이트수준에서 구현하는 것으로, 수행속도가 매우 빠르다는 장점이 있으나, 복잡한 모델을 수용하기 힘들어 그 적용 범위가 제한된다. 메모리 참조표를 사용한 구현은 모델에 관계없이 고정된 사항에 따라 구현 구조를 설정하고 모델에 따라서 변하는 부분은 메모리 참조표에 저장한다[4][23]. 메모리 참조표를 사용하는 경우에는 하드와이어 방법보다 복잡한 모델의 수용이 가능하며, 소프트웨어 방법에 비하여 속도가 빠르다는 장점이 있다.

메모리 참조표의 구조는 크게 리스트 구조와 행렬 구조로 나뉠 수 있다. 리스트 구조의 참조표는 메모리 사용형태가 효율적인 반면에 참조 형태가 복잡하고 불규칙하므로 소프트웨어 구현 방법에는 적합하나 하드웨어로 구현하는 경우에는 적용이 곤란하다. 따라서 하드웨어 형태로 구현할 경우에는 행렬 구조의 참조표를 사용하는 것이 용이하나, 행렬 구조를 사용하여 참조표를 구현하는 경우에는 메모리 사용형태가 비효율적이며 그 증가도가 매우 크다는 단점을 안고 있다.

PN의 하위 계열로 분류되는 FSM(finite state machine)의 경우는 구조가 간단하고 정형화된 구현 모델이 존재하므로[24], 메모리 참조표를 사용하여 구현할 경우 구조의 설정이 명확하다. FSM의 경우에도 행렬구조의 메모리 참조표를 사용하면 메모리 폭증 문제가 발생하므로, 이진 (binary) 분기 구조의 메모리 참조표를 사용하여 메모리 사용량을 줄이는 연구가 발표되었다. 이진 분기 구조는 수행단계의 증가에 따른 수행 속도의 저하를 가져올 수 있으나, 메모리 사용량을 매우 효과적으로 감소시킨다[5][23]. 반면에, PN의 경우에는 FSM과는 달리 메모리 참조표를 사용한 하드웨어의 구조 설정이 어려우며, FSM에 적용된 이진 분기 구조를 PN에 그대로 적용할 수 없다. 관련 연구로는 주문형 반도체를 사용한 PN 하드웨어 구현 연구가 있는데[26][27], 이 방법에서는 행렬 구조의 참조표를 그대로 적용하여 실용적인 면에서 불리하다. 이 연구에서는 PN를 계층적으로 분해하여 메모리 폭증 문제의 해결을 시도하였으나, 일반적인 분해 방법에 대한 결과가 없으며, 리스트 구조의 참조표에 비하여 메

접수일자 : 1997. 2. 10., 수정완료 : 1998. 3. 6.
장래혁 : 서울대학교 컴퓨터공학과
정승권, 권욱현 : 서울대학교 전기공학부

모리 사용량이 월등히 많다.

본 연구의 목적은 비교적 빠른 속도와 유연성을 동시에 수용하는 메모리 참조표를 사용한 구현 방법을 PN에 적용하되, 간단한 구조와 효율적인 메모리 사용에 초점을 맞추는 것이다. 본 연구의 특징은 변형된 행렬 구조의 참조표를 사용하여, 하드웨어 형태로 PN 제어기를 구현한 것에 있다. 특히, 여러 가지 구현 구조와 알고리듬 및 메모리 사용량과 구조적 문제점을 여러 PN 계열에 따라서 제시하였다.

본 논문은 5장으로 구성되어 있으며, 2장에서는 PN와 여러 가지 하위 계열 PN을 정의하고, 일반 PN을 변환하는 방법을 제시하였다. 3장에서는 변형된 행렬 구조 메모리 참조표를 사용한 PN의 구현방법을 PN 계열에 따라 제시하고, 4장에서는 제한된 구현 방법을 순차 제어기에 응용하는 예를 보였으며, 결론은 5장에서 언급하였다.

II. PN과 하위계열 PN

PN은 6 튜플 (P, T, IN, OUT, M_0)로 구성되며, 각 튜플은 다음과 같다. $P=(p_1, \ldots, p_n)$은 플레이스의 집합, $T=(t_1, \ldots, t_m)$는 천이 집합, IN : $(P\times T)\rightarrow N$ 은 입력 함수, OUT : $(P\times T)\rightarrow N$ 은 출력 함수, $M_0=(\mu_0(p_0), \ldots, \mu_0(p_n))$은 플레이스의 초기 마킹이다. 여기서, N 은 음이 아닌 정수이고, $|\cdot|$는 집합 $\cdot$의 총 원소 수이며, $|P|=n$, $|T|=m$ 이다. 플레이스 p의 입력 천이 집합은 $\cdot p=\{t\in T : IN(p,t)\neq 0\}$, 출력 천이 집합은 $p\cdot=\{t\in T : OUT(p,t)\neq 0\}$로 정의되고, $\cdot t=\{p\in P : IN(p,t)\neq 0\}$는 천이 t의 입력 플레이스 집합, $t\cdot=\{p\in P : OUT(p,t)\neq 0\}$는 천이 t의 출력 플레이스 집합으로 정의된다[28].

1. 하위 계열 PN의 정의

PN에서는 천이에 연결되는 입출력 플레이스의 수에 제한을 두지 않으나, 본 논문에서 제안하는 이분 PN (이진 분기 PN, Biarced PN)[1]에서는 천이의 입출력 플레이스의 갯수가 2개 이하로 제한되고, 플레이스의 출력 천이의 갯수도 2개 이하로 제한된다. 그렇지만 제한되는 갯수가 2개이므로 일반 PN과 동일하게 판단 및 병렬성의 표현이 가능하다.

메모리 참조표를 사용한 하드웨어 구현은 시스템 클럭에 동기되어 동작하며, 입출력 신호도 클럭에 동기되어 인식된다. 이러한 동기식 동작에서는, PN 구현에서 흔히 가정되는 한번에 한 가지의 사건만이 일어난다는 가정이 성립되지 않는 경우가 발생할 수 있다. 이러한 경우에 우선 순위 PN[29]은 효과적으로 충돌 해결 (선택동작, conflict resolution)을 수행한다. 본 논문에서는 우선 순위 PN을 이용하여 동시에 2개 이상의 사건이 샘플되는 경우에도 예측가능한(deterministic) 동작을 얻는다.

정의 1 : 이분 PN $=(P, T, IN, OUT, M_0)$는 우선 순위 PN이며, $\forall t\in T$, $|\cdot t|\leq 2$이고 $|t\cdot|\leq 2$이다. 그리고, $\forall p\in P$, $|p\cdot|\leq 2$이다.

본 논문에서는 보다 범용성 있는 구현 구조를 설정하기 위해서 근원 플레이스, 흡수 플레이스, 근원 천이, 흡수 천이를 고려하여, 하위 계열(sub-class) PN을 다음과 같이 정의한다. MG(marked graph)는 하위 계열 PN이며, $\forall p\in P$, $|\cdot p|\neq 0$이고 $|p\cdot|\neq 0$이면, $|\cdot p|=1$이고 $|p\cdot|=1$이다.

정의 2 : 이분 MG는 하위 계열 MG이며, $\forall t\in T$, $|\cdot t|\leq 2$이고 $|t\cdot|\leq 2$이다.

본 논문에서 언급하는 하위 계열 PN의 제약 조건은 표 1과 같이 정리된다.

표 1. 하위 계열 PN의 제약 조건.
Table 1. Restriction of sub-class PNs.

명칭	플레이스 관련 제약 조건	천이 관련 제약 조건
이분 PN	$\|p\cdot\|\leq 2$	$\|\cdot t\|\leq 2$ 이고 $\|t\cdot\|\leq 2$
MG	$\|\cdot p\|\leq 1$ 이고 $\|p\cdot\|\leq 1$	없음
이분 MG	$\|\cdot p\|\leq 1$ 이고 $\|p\cdot\|\leq 1$	$\|\cdot t\|\leq 2$ 이고 $\|t\cdot\|\leq 2$

2. 이분 구조 하위 계열 PN으로의 변환

본 소절에서는 일반 PN과 일반 MG를 제안된 이분 PN과 이분 MG로 변환하는 방법에 대해서 언급한다. 이분 PN과 이분 MG의 결합과 산개는 그림 1에서와 같이 이분 결합(join)과 이분 산개(fork)의 다단 연결로 표현이 가능하다. 이분 결합과 산개 변환은 기본 축소 규칙(simple reduction rules)[28]의 일종이므로, 이분 결합과 산개 변환을 이용할 경우 원래 PN또는 MG의 안정성(safeness), 유한성(boundedness), 도달성(reachability) 및 포함성(coverability)이 그대로 보존된다. 지금부터, 편의상 이분 PN과 이분 MG를 일반 PN과 MG와 구별하기 위해서 ($\widehat{P}, \widehat{T}, \widehat{IN}, \widehat{OUT}, \widehat{M}_0$)으로 표기한다. 이분 결합과 이분 산개 변환 시 생성되는 추가의 플레이스와 천이를 빈(dummy) 플레이스 $\widehat{P}$와 빈(dummy) 천이 $\widehat{T}$라 부른다. 즉, $\widehat{P}\equiv\widehat{P}-P$, $\widehat{T}\equiv\widehat{T}-T$ 이다. 또, 변환 전 PN 및 MG의 마킹은 이진 PN 또는 이진 MG 플레이스의 마킹의 종속 벡터(sub-vector)가 된다. 즉, $\widehat{M}=(M\ \widehat{M})^T$ 이다.

변환 전의 PN 또는 MG에서 $M_i[t_j\rangle M_{i+1}$이고, 변환된 이분 PN 또는 이분 MG에서 $\widehat{M}_i[t_{seq}\rangle\widehat{M}_{i+1}$라 가정할 경우, $\widehat{M}$을 제외한 다른 원소가 서로 같다면, t_j에 의한 PN 또는 MG의 동작이 t_{seq}에 의한 이분 PN또는 이분 MG의 동작과 동등하다고 정의한다. 이 때, t_{seq}를 t_j와 동등한 천이군이라 부르며, t_1에 의한 점화에 따라서 M_1에서 M_2로 마킹이 변화되는 것을 $M_1[t_1\rangle M_2$로 표기한다[29].

MG와는 달리 PN을 이분 구조로 변환하는 과정에서는 이분 산개 및 이분 결합 변환과 더불어 이분 판단 변환이 사용된다(그림 2참조). 이분 판단 변환은 이분 결합과 산개 변환과는 달리 기본 축소 규칙에 속하지 않으며, 이분 판단 변환의 경우는 변환 시 판단의 우선 순위가 생기게 된다. 동기식 동작을 하는 경우, 사건 간에 우선 순위를 두어 일관성(persistency)을 유지하는 것은 실제 시스템의 구현 시에 매우 유용한 방법이다.

III. 메모리 참조표를 사용한 PN의 구현 방법

메모리 참조표를 사용한 하드웨어 구현에서는 정형화

1) Binary PN은 safe PN을 지칭하는 것으로 널리 사용되므로, 혼동을 피하기위하여 노드에 연결된 아크의 개수가 2개로 제한된다는 뜻으로 Biarced PN으로 명함.

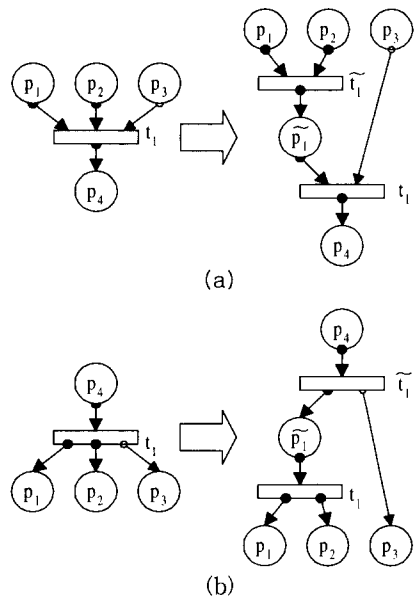

그림 1. 이분 결합(a) 및 이분 산개 변환(b).

Fig. 1. Biarced fork and biarced join conversion.

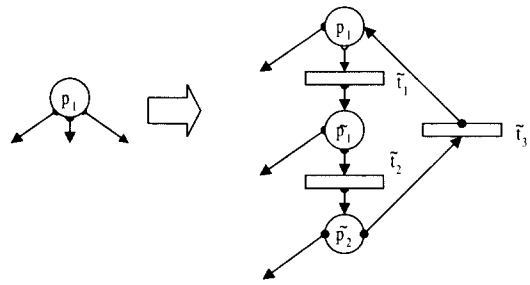

그림 2. 이분 판단 변환.

Fig. 2. Biarced decision conversion.

된 구현 모델의 구성 요소를 참조표 형태로 구성하며, 참조표는 메모리 칩으로 구현된다. 각 구성 요소의 입력은 메모리 참조표의 주소가 되며, 출력은 데이타가 된다. PN을 수행하는 장치를 구현할 경우 일반적으로 유한성(boundedness)이 가정된다. 플레이스의 마킹을 k 비트의 이진 코드로 표현할 경우 최대(2^k-1) -유한 PN을 표현할 수 있다. 또, PN을 수행할 때, 다수의 자원으로 병렬처리를 하지 않는 경우에는 PN의 천이를 병렬적으로 점화할 수 없으므로, 동시에 한 개만의 천이가 점화된다는 아토믹 천이의 가정이 흔히 고려된다. PN의 점화 규칙은 활성화(enable)된 천이는 점화하여도 좋다(may fire)라고 규정되어 있으나, 즉시 점화(immediate firing)의 가정은 활성화된 천이를 즉시 점화시킨다[8]. 본 논문에서 제안하는 구현 모델은 토큰 게임의 경우와 같은 수행 방법을 따르도록 하였으며, 유한성, 아토믹 천이, 즉시 점화를 전제로 한다.

1. 일반 PN의 메모리 참조표를 사용한 구현

1.1 PN의 메모리 참조표를 사용한 구현의 기본 구조

PN의 정의에 따라서 메모리 참조표를 사용한 구현은 O(p), I(t), O(t) 및 m(p), Q의 참조표로 구성된다. 이들 참조표는 다음과 같다.

m(p) : 각 플레이스의 마킹 M를 저장하는 메모리 참조표로, 표의 입력은 플레이스의 번호이며 대응되는 출력은 해당 플레이스의 마킹 값이다.

O(p) : 각 플레이스의 출력 천이 $p\bullet$를 나타내는 메모리 참조표이다. 표의 입력은 플레이스의 번호이며 출력은 해당 플레이스의 출력 천이의 번호이다.

I(t) : 각 천이의 입력 플레이스 $\bullet t$를 나타내는 메모리 참조표이다. 표의 입력은 천이의 번호이며 출력은 해당 천이의 입력 플레이스의 번호이다.

O(t) : 각 천이의 출력 플레이스 $t\bullet$를 나타내는 메모리 참조표이다. 표의 입력은 천이의 번호이며 출력은 해당 천이의 출력 플레이스의 번호이다.

Q : 점화가 가능한 천이를 가리키는 플레이스의 번호를 저장하는 FIFO이다. 점화된 천이의 출력 플레이스의 번호를 다시 큐의 입력으로 되돌린다.

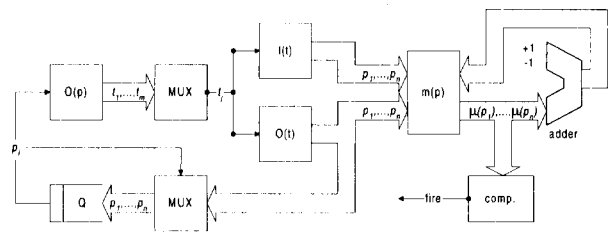

그림 3. 메모리 참조표를 사용한 PN 구현의 기본 데이타 경로 구조.

Fig. 3. Structure of data path for memory-based implementation of PN.

이들 참조표 외에, 입력 플레이스의 마킹 값으로 활성화를 검사하는 비교기(comp. 그림 3)가 필요하며, 점화 시 입력 플레이스의 마킹과 출력 플레이스의 마킹 값을 가감할 수 있는 연산기 (adder 그림 3)와 이들을 제어하는 제어 회로가 요구된다. 데이타 경로와 제어 회로는 다음의 알고리듬에 기초하여 구현된다.

알고리듬 1 : 초기화

```
do 초기화
  ∀t_k, t_k∈T do
    if μ_0(p_k)≠0 do
      p_k를 Q에 삽입; /* Q의 초기화 */
      m(p_k)←1; /* m(p)의 초기화 */
    end do
    else do
      m(p_k)←0;
    end do
  end do
end do
```

알고리듬 2 : PN 수행

```
do PN 수행 while(Q가 비어있지 않은 동안)
  p_i를 Q로부터 추출;
  t_i← O(p_i) 중에서 선택된 천이;
  if( ∀p_j∈•t_i, m(p_j) ≥1) do
    ∀p_j∈•t_i, m(p_j) ← m(p_j) −1;
    ∀p_k∈t_i•, m(p_j) ← m(p_j) +1,
    p_k를 Q에 삽입;
  end do
  else do
```

p_i 를 다시 Q에 삽입;
end do
end do

판단 기능을 수행할 때, 선택된 천이가 활성화되지 않았을 경우에는 선택기에서 출력된 플레이스를 다시 Q로 삽입하여야 PN의 수행이 비정상적으로 정지되지 않는다. PN의 모의 실험이나 해석에는 확률적 또는 무작위의 판단 방법이 흔히 사용된다. 그러나 실제로 PN을 외부사건과 연계하여 수행함으로써 순차제어에 적용시키는 경우에는 이와 같은 불확정적인 판단 방법은 유용하지 않고, 판단을 수행하기 위한 결정적인 근거(transition predicates)가 요구된다.

1.2 일반 PN의 메모리 참조표를 사용한 구현의 구조적 문제점

수행되어야 하는 PN가 n개 플레이스와 m개 천이를 가지고 있는 임의의 PN이라 할 때, 참조표 O(p)에서 출력되는 천이 수는 최대 m개이다. O(p)로부터 출력된 출력 천이의 활성화를 검사하기 위해 조사해야 되는 입력 플레이스의 갯수는 최대 n개이다. 점화 후, 하나의 해당 출력 천이 당 최대 n개 입력 플레이스와 최대 n개 출력 플레이스의 마킹을 갱신해야 한다. 그러나, 이들 최대 갯수는 실제 값과는 매우 큰 차이를 보이므로 이들을 효율적으로 저장할 수 있는 참조표는 리스트 구조와 같이 각 주소마다 가변적인 길이를 가져야 하므로, 리스트 구조의 참조표는 PN을 소프트웨어로 구현할 경우에 매우 유용하다. 그러나, PN을 하드웨어로 구현하려는 경우에는 복잡하고 불규칙한 리스트 구조의 참조표를 사용하는 것은 용이하지 않다. 따라서 하드웨어 형태로 구현할 경우 행렬 구조의 참조표를 사용해야 하나, 행렬 구조의 참조표를 사용할 경우, 이러한 구조적인 문제는 플레이스 수와 천이 수를 늘림에 따라 더욱 증폭된다. 본 연구에서는 이러한 문제를 해결하기 위해서, 플레이스 수와 천이 수에 관계없이 행렬 구조 참조표의 열(column) 수를 고정할 수 있는 방법을 제안한다. 제안된 방법은 앞에서 정의한 이분 구조의 하위 계열 PN을 사용한다.

2. 이분 구조 하위 계열 PN의 메모리 참조표를 사용한 구현 구조

2.1 이분 MG

앞 절에서 언급한 문제에 따라서, 그림 3의 구조에서는 O(p), I(t), O(t)의 데이터 폭이 넓은 점 외에도 m(p)와 Q에서 여러 주소를 동시에 접근해야 하는 문제가 있다. 특히 m(p)의 경우 구현하기가 매우 힘든 구조를 요구한다. 이분 MG의 경우에는 O(p), I(t), O(t)의 데이터 폭이 모두 2워드 이하로 제한되는 장점 외에도, m(p)를 구현하기가 매우 수월하다는 장점이 있어 구현이 매우 용이한 구조를 얻을 수 있다.

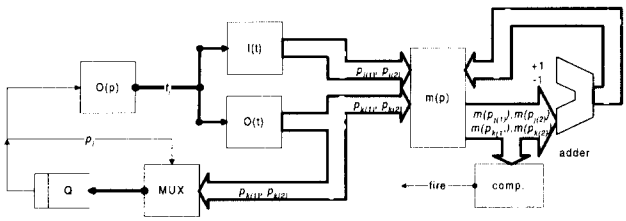

그림 4. 알고리듬 3에 따른 메모리 참조표를 사용한 이분 MG 구현의 데이터 경로 구조.

Fig. 4. Structure of data path for memory-based implementation of biarced MG with Algorithm 3.

알고리듬 3 : 이분 MG 수행 1
do 이분 MG 수행 while(Q가 비어있지 않은 동안)
p_i 를 Q로부터 추출;
$t_i \leftarrow$ O(p_i); /* MG이므로 O(p_i)의 출력은 1개 */
$p_{j(1)}, p_{j(2)} \leftarrow$ I(t_i); /* 이분 MG이므로 I(t_i)의 출력은 2개 */
$p_{k(1)}, p_{k(2)} \leftarrow$ O(t_i); /* 이분 MG이므로 O(t_i)의 출력은 2개 */
if(null이 아닌 $p_{j(1)}, p_{j(2)}$에 대하여,
m($p_{j(1)}$) ≥1이고 m($p_{j(2)}$) ≥1) do
m($p_{j(1)}$) ← m($p_{j(1)}$) −1;
m($p_{j(2)}$) ← m($p_{j(2)}$) −1;
m($p_{j(1)}$) ← m($p_{j(1)}$) +1;
m($p_{j(2)}$) ← m($p_{j(2)}$) +1;
$p_{k(1)}, p_{k(2)}$가 null이 아니면 Q에 삽입;
end do
else do
p_i를 다시 Q에 삽입
end do
end do

알고리듬 3으로 제어하는 하드웨어의 데이타 경로 구조를 그림 4에 나타내었다. 굵은 선으로 표시된 것과 같이, Q(p), I(t), O(t), MUX, m(p)의 출력 포트 개수가 1개, 2개 또는 4개로 제한되므로 그림 3의 구조가 실제로 구현이 용이하지 않은 반면, 그림 4의 구조는 쉽게 구현이 가능하다. 한 천이의 점화에 걸리는 시간이 출력 플레이스의 수에 따라 저하하지 않게 하려면, 다수의 출력 플레이스를 한 번에 삽입할 수 있는 Q가 마련되어야 한다. 이와 같은 구조적인 문제로, 실제로 구현할 때는 이진 분기 등의 방법으로 최대 출력 플레이스 수를 제한하는 방법이 매우 도움이 된다.

알고리듬 3에서는 점화 후 다음에 점화하려는 천이를 Q에서 추출한 플레이스의 출력 천이로 선정하며, 점화한 천이의 출력 플레이스는 다시 Q의 뒷부분에 삽입한다. 즉, 현 시점에서 점화된 천이의 출력 플레이스가 가리키는 (다음 사이클에 점화 가능한) 천이의 처리를, 가장 뒤로 미루게 된다. 반면, 알고리듬 4는 다음에 활성화를 검사할 천이를 현재 점화된 천이의 출력 플레이스로부터 찾는다. 활성화 검사 후, 점화가 되지 않은 경우에는 새로운 플레이스를 Q에서 받아들인다. 즉, 점화가 계속 이루어지면 Q에서는 플레이스가 추출되지 않는다. 선택기는 점화 여부에 따라서 O(p) 의 입력을 O(t) 또는 Q에서 받아들일 것인가를 선택한다.

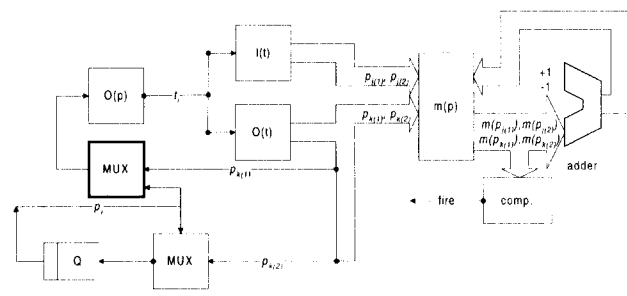

그림 5. 알고리듬 4에 따른 메모리 참조표를 사용한 이분 MG 구현의 데이터 경로 구조.

Fig. 5. Structure of data path for memory-based implementation of biarced MG with Algorithm 4.

알고리듬 4에 따른 하드웨어는 그림 5에서 볼 수 있다. 굵은 선으로 표시된 새로운 MUX가 다음에 점화하려는 천이를 선정하는 플레이스를 결정한다. 알고리듬 4는 가능한 Q의 사용을 제한할 수 있는 알고리듬이므로, 알고리듬 3에 비하여 Q에서 생길 수 있는 구조적 문제점을 완화시킬 수 있다.

알고리듬 4 : 이분 MG 수행 2

```
do 이분 MG 수행 while(Q가 비어있지 않은 동안)
   p_i 를 Q로부터 추출;
   do forever
      t_i ← O(p_i);   /* MG이므로 O(p_i)의 출력은 1개 */
      p_j(1), p_j(2) ← I(t_i);   /* 이분 MG이므로 I(t_i)의 출
                                    력은 2개 */
      p_k(1), p_k(2) ← O(t_i);   /* 이분 MG이므로 O(t_i)의
                                    출력은 2개 */
      If(null이 아닌 p_j(1), p_j(2)에 대하여, m(p_j(1)) ≥1
      이고 m(p_j(2)) ≥1) do
         m(p_j(1)) ← m(p_j(1)) −1;
         m(p_j(2)) ← m(p_j(2)) −1;
         m(p_j(1)) ← m(p_j(1)) +1;
         m(p_j(2)) ← m(p_j(2)) +1;
         p_j가 null이 아니면 p_i ← p_k(1);
         p_k(2)가 null이 아니면 Q에 삽입;
      end do
      else do
         p_i를 다시 Q에 삽입
         break;
      end do
   end do
end do
```

2.2 이분 PN

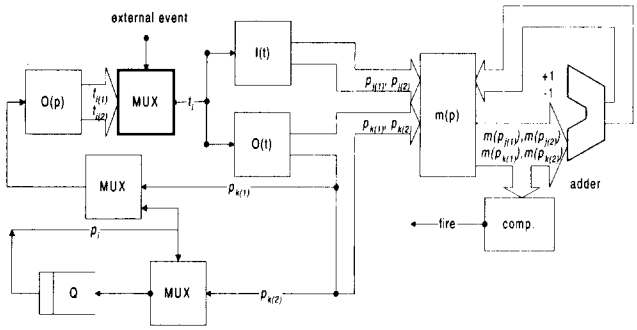

그림 6. 알고리듬 5에따른 이분 PN의 메모리 참조표를 사용한 구현의 데이터 경로.

Fig. 6. Structure of data path for memory-based implementation of biarced PN with Algorithm 5.

알고리듬 5 : 이분 PN 수행

```
do 이분 PN 수행 while(Q가 비어있지 않은 동안)
   p_i 를 Q로부터 추출;
   do forever
      t_i(1), t_i(2) ← O(p_i);   /* 이분 PN이므로 O(p_i)의
                                    출력은 2개 */
      t_i ← 외부 사건이 선택한 t_i(1)또는 t_i(2);
      p_j(1), p_j(2) ← I(t_i);   /* 이분 MG이므로 I(t_i)의 출
                                    력은 2개 */
      p_k(1), p_k(2) ← O(t_i);   /* 이분 MG이므로 O(t_i)의
                                    출력은 2개 */
      If(null이 아닌 p_j(1), p_j(2)에 대하여, m(p_j(1)) ≥1
      이고 m(p_j(2)) ≥1) do
         m(p_j(1)) ← m(p_j(1)) −1;
         m(p_j(2)) ← m(p_j(2)) −1;
         m(p_j(1)) ← m(p_j(1)) +1;
         m(p_j(2)) ← m(p_j(2)) +1;
         p_j가 null이 아니면 p_i ← p_k(1);
         p_k(2)가 null이 아니면 Q에 삽입;
      end do
      else do
         p_i를 다시 Q에 삽입
         break;
      end do
   end do
end do
```

그림 3의 일반 PN의 구현 구조와 달리, 이분 PN의 경우에는 O(p), I(t), O(t)의 데이터 폭과, Q와 m(p)의 포트 수가 PN을 표현할 수 있는 최소 한으로 한정되므로 구현성이 매우 향상된다. 그림 6의 데이터 경로 구조는 이분 MG구현 구조에 외부 사건에 따라 충돌 해결을 수행할 수 있도록 굵은 선으로 표시된 MUX가 더 추가된다. 이분 MG 구현의 알고리듬 4와 같이 Q의 사용 빈도를 줄여 구조적 문제점을 완화하였다.

3. 행렬 구조의 메모리 참조표를 사용한 구현 장치의 용량

메모리 참조표를 사용한 구현에서는 사용된 메모리 참조표의 크기에 따라서 수행할 수 있는 대상의 크기가 제한되는데, 행렬 구조의 참조표에서는 참조표의 데이터 폭과 주소 공간이 각각 독립적으로 대상의 크기를 제한한다. 본 논문에서는 PN의 크기를 플레이스 수와 천이 수로 결정하고, 이에 따라 메모리 참조표를 사용한 구현 장치의 용량을 정의한다. 이들 참조표의 크기의 단위는 "워드(word)"이며, 워드 길이(word length)의 단위는 "비트(bit)"로 나타낸다. 워드 길이는 모든 플레이스와 천이의 번호를 표현할 수 있을 만큼 충분하여야 한다. 만일 플레이스와 천이를 구분 짓는데 이진 코드(binary code)가 사용된다고 하자. 그러면 O(p)의 최소한의 워드 길이는 $\lceil \log_2 \lvert T \rvert \rceil$ 이고, Q, O(t)와 I(t)의 최소한의 워드 길이는 $\lceil \log_2 \lvert P \rvert \rceil$ 이다. 여기서 $\lceil \cdot \rceil$ 은 $\cdot$ 보다 작지 않은 최소의 정수이다. m(p)의 경우에는 최대 허용되는 마킹의 수를 표현할 수 있을 만큼의 최소한의 워드 길이가 보장되어야 한다. 최대 유한성을 ξ라 할 때, m(p)의 워드 길이는 $\lceil \log_2 \xi \rceil$ 가 되어야 한다.

3.1 총 플레이스 수 및 천이 수만을 고려한 경우

행렬 구조의 메모리 참조표를 사용한 구현 구조의 용량을 총 플레이스 수와 총 천이 수로 정의할 수 있다. 이 경우, 수행하려는 대상의 총 플레이스 수와 총 천이 수가 한계를 넘지 않으면 수행 가능하다고 판단한다. [n개의 플레이스와 m 개의 천이의 용량]을 가진 메모리 참조표를 사용한 구현에 필요한 메모리 참조표의 크기는 표 2와 같다 (증명 생략).

3.2 흡수 및 근원 플레이스와 흡수 및 근원 천이를 고려한 경우

제어기를 모델할 경우에 센서나 액튜에이터를 흡수 플레이스나 근원 플레이스, 흡수 천이나 근원 천이로 표현하는 경우가 있다. 흡수 플레이스와 근원 천이는 I(t)에 포함될 필요가 없으며, 근원 플레이스와 흡수 천이는 O(t)에 포함될 필요가 없다. 또, 흡수 플레이스와 근원 천이는 O(p)에 포함될 필요가 없으며, Q에도 흡수 플레이스가 포함되지 않아도 된다. 따라서 대상 PN이 근원 플레이스, 흡수 플레이스, 근원 천이 또는 흡수 천이를 포함하고 있으면, 총 플레이스 수와 총 천이 수만을 고려한 기준으로 판단하였을 때 용량을 초과하는 것으로 판단되는 경우에도 실제로는 수행이 가능할 수 있다. 이들을 고려한 경우 [n 개의 플레이스와 m 개의 천이의 용량]의 메모리 참조표를 사용한 구현에 필요한 메모리 참조표의 크기는 표 3과 같다. PN에 포함된 근원 플레이스 수, 흡수 플레이스 수, 근원 천이 수, 흡수 천이 수를 각각 n_s, n_k, m_s, m_k로 나타낸다 (증명 생략).

표 2. MG의 메모리 참조표의 크기(1).
Table 2. Size of memory table of MG(1).

참조표	데이터 폭 (비트)	주소 공간
m(p)	$\lceil \log_2 \xi \rceil$	n
O(p)	$m \times \lceil \log_2 m \rceil$	n
I(t)	$n \times \lceil \log_2 n \rceil$	m
O(t)	$n \times \lceil \log_2 n \rceil$	m
Q	$\lceil \log_2 n \rceil$	n

주 : 총 플레이스 수와 총 천이만을 고려한 용량의 정의에 따른 구현.

표 3. MG의 메모리 참조표의 크기(2).
Table 3. Size of memory tables of MG(2).

참조표	데이터 폭 (비트)	주소 공간
m(p)	$\lceil \log_2 \xi \rceil$	n
O(p)	$(m-m_s) \times \lceil \log_2 m \rceil$	$n-n_k$
I(t)	$(n-n_k) \times \lceil \log_2 n \rceil$	$m-m_s$
O(t)	$(n-n_s) \times \lceil \log_2 n \rceil$	$m-m_k$
Q	$\lceil \log_2 n \rceil$	$n-n_k$

주 : 근원 플레이스, 흡수 플레이스, 근원 천이, 흡수 천이 수를 모두 고려한 용량의 정의를 따른 구현

3.3 이분 PN 및 이분 MG

근원 플레이스, 흡수 플레이스, 근원 천이, 흡수 천이를 모두 고려한 경우, 이분 PN과 이분 MG의 [n'개의 플레이스와 m'개의 천이의 용량]의 메모리 참조표를 사용한 구현에서 필요한 메모리 참조표의 크기는 4와 같다 (증명 생략). 이분 PN과 이분 MG의 메모리 참조표는 2 워드의 데이터 폭을 가지므로 구현성이 매우 향상된다.

4. 확장성

행렬 구조의 참조표에서는 수행해야 할 대상의 크기가 커져감에 따라서 $|P|$가 증가하면 메모리 참조표의 수평 넓이, 즉 데이터 폭이 커져야 한다. 또, 수행해야 할 모델의 크기가 커져감에 따라서 $|T|$가 증가하면 메모리 참조표의 수직 길이, 즉 주소공간이 늘어나야 한다. 이미 구성된 하드웨어의 메모리 참조표의 주소 공간을 늘리는 것이 메모리 참조표의 데이터 폭을 넓히는 것에 비하여 일반적으로 비용이 적게 들게 된다. 데이터 폭의 경우, 처리하는 장치와 연결할 수 있는 데이터 신호를 늘리려면 메모리 칩의 교환 뿐 아니라 처리하는 장치의 자체의 심각한 변경도 요구되는 경우가 많다. 이분 MG의 경우, 행렬 구조의 참조표를 사용하더라도 처리해야 할 모델의 크기가 커짐에 따라서 생기는 메모리 참조표의 확장은 항상 주소 공간에 대해서만 요구되며, 데이터 폭은 항상 2 워드로 제한되게 된다. 따라서 확장성이 뛰어날 뿐 아니라 적은 비용으로도 가능하게 된다.

표 4. 이분 PN과 이분 MG의 메모리 참조표의 크기.
Table 4. Size of memory tables of biarced MG and biarced PN.

참조표	데이터 폭 (비트)	주소 공간
m(p)	$\lceil \log_2 \xi \rceil$	n'
O(p)	PN: $2 \times \lceil \log_2 m' \rceil$, MG: $\lceil \log_2 m' \rceil$)	$n'-n'_k$
I(t)	$2 \times \lceil \log_2 n' \rceil$	$m'-m'_s$
O(t)	$2 \times \lceil \log_2 n' \rceil$	$m'-m'_k$
Q	$\lceil \log_2 n' \rceil$	$n'-n'_k$

주 : 근원 플레이스 수, 흡수 플레이스 수, 근원 천이 수 및 흡수 천이 수를 고려한 용량의 정의를 따른 구현.

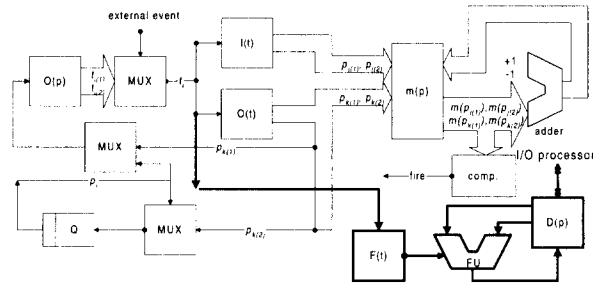

그림 7. 메모리 참조표를 사용한 순차 제어기의 연산 처리 부분의 구조.
Fig. 7. Structure of processing unit of PLC based on memory-based implementation.

IV. 메모리 참조표를 사용한 PN의 구현을 응용한 순차 제어기

1. 순차제어기의 구조

메모리 참조표를 사용한 PN의 구현을 응용한 순차 제어기는 데이터 메모리(DM), 연산 부분(FU)과 이들을 제어하는 회로로 구성된다. 여기에 입출력 접점을 통한 데이터의 전송을 수행하는 입 출력 처리 부분이 결합되는데, 본 논문에서는 이를 입출력 프로세서라 한다. FU는 응용 프로그램의 수행 중 데이터의 연산을 담당하며 입력과 출력은 DM의 정보가 된다. DM에 저장된 정보는 PN의 플레이스와 1:1 대응이 된다. MG의 메모리 참조표를 사용한 구현을 순차 제어기에 응용하기 위해서 DM과

FU의 명령어를 표현할 수 있는 이분 확장 MG를 정의한다.

정의 3 : 순차 제어기를 묘사하는 이분 확장 MG는 $(P', T', IN', OUT', M'_0, f, d')$이며, $f(t)=\{f \mid (t,f)\}$는 천이 t 에서 정의되는 FU의 명령어를 나타내며, $d'(p)=\{d \mid (p,d)\}$는 플레이스 p 에서 정의되는 DM의 원소이다. 이분 PN에 근거한 연산 처리 구조는 그림 7에서 볼 수 있다. 정의 3에 의하여 추가된 부분은 굵은 선으로 표시되었다. F(t)와 D(p)는 $f(t)$와 $d'(p)$를 저장하는 메모리 참조표이다. 입출력 프로세서는 $d(p)$와 M'_0를 D(p)와 m(p)에게 매번 수행마다 전달한다. M'_0는 오프라인(off-line) 계산으로 얻을 수 있다. FU는 명령어와 데이터에 따른 적절한 데이터를 출력하며 이 결과는 다시 D(p)에 저장된다.

2. 자동 설계 환경

본 논문에서는 메모리 참조표를 사용한 PN 구현 방법을 실제 응용에 손쉽게 적용할 수 있는 자동 설계 방법을 제시한다. 제시되는 방법의 수행 과정은 다음과 같다.

1) 시스템 분석 및 PN 구성
2) 구성된 PN의 검증
3) 그래픽 편집기로 PN의 입력
4) PN의 네트 리스트(net list) 출력
5) 네트 리스트 변환을 통한 참조표의 출력

위 과정 중에서 시스템 분석부터 검증까지는 기존 연구 결과를 이용하여 수행이 가능하다. 본 논문에서는 과정 3부터 과정 5까지를 처리하는 소프트웨어 환경을 구축하였다. PN을 입력하는 그래픽 편집기로는 상용의 회로 편집기(OrCAD)를 사용하였다. 상용의 회로 편집기에 PN의 입력에 사용되는 라이브러리를 구현하여 제공하고, 네트 리스트를 출력하는 프로그램을 작성하여 원하는 형식의 네트리스트를 얻는 방법을 사용하였다. 플레이스는 초기 마킹 값이 0인 P와 초기 마킹 값이 1인 I로 구분하여 입력한다. 천이는 T로 시작되는 기호를 사용하여 입력한다. 입력된 결과는 참조표로 자동 변환된다.

그림 8은 이분 PN의 입력 예로 화학 공장 제어기[31]의 일부이다. 일반 PN을 사용하여 입력한 경우 전체 제어기는 $n=78, m=39, n_s=18, n_k=9, m_s=m_k=0$이며, 이를 이분 PN으로 바꾼 경우, $n'=89$, $m'=50$, $n'_s=18$, $n'_k=9$, $m'_s=m'_k=0$이 된다. 이들은 안전 PN이므로 $\xi=1$이고, $\lceil \log_2 78 \rceil = 7$ $\lceil \log_2 39 \rceil = 6$, $\lceil \log_2 89 \rceil = 7$, $\lceil \log_2 50 \rceil = 6$이므로 행렬 구조 참조표의 용량은 표 5와 같이 된다.

표 5. 일반 PN과 이분 PN의 행렬 구조 참조표의 크기 비교 예.

Table 5. Comparison example of matrix-baed look-up tables between PN and Biarced PN.

참조표	일반 PN의 참조표 크기	이분 PN의 참조표 크기
m(p)	78 bits	89 bits
O(p)	16,146 bits	480 bits
I(t)	18,837 bits	700 bits
O(t)	16,380 bits	700 bits
Q	483 bits	560 bits
합계	51,424 bits	2,529 bits

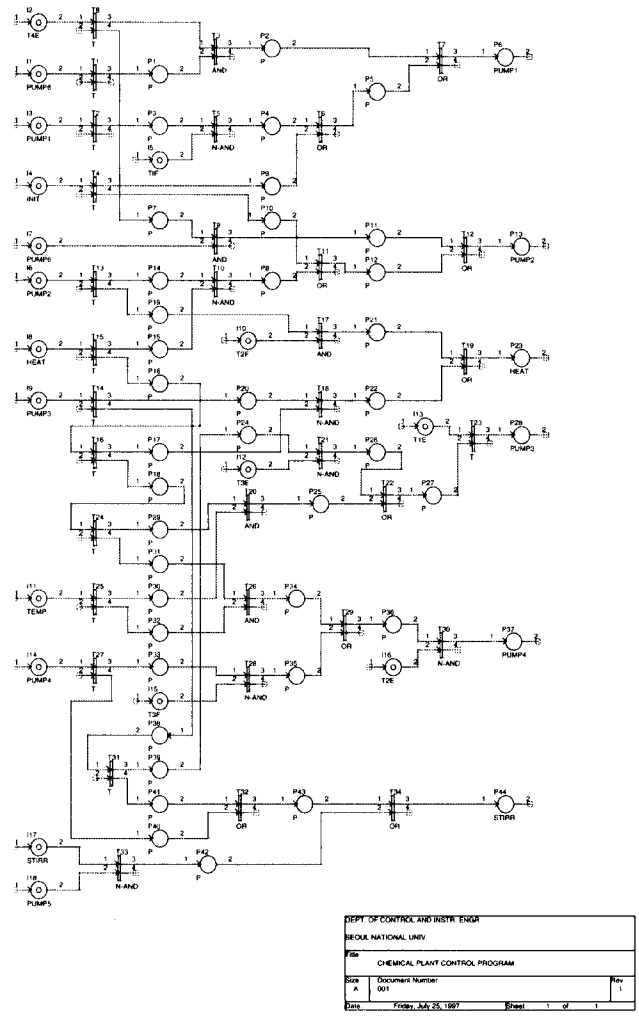

그림 8. 이분 PN의 입력 예.

Fig. 8. Design entry example of a Biarced Petri net controller.

V. 결론

본 논문에서는 PN의 하드웨어 구현 방법과 이를 적용한 순차 제어기에 대하여 연구하였다. 참조표를 사용하여 PN을 구현할 경우, 리스트 구조를 채택하면 메모리 폭증 문제를 해결할 수 있으나, 하드웨어로 구현하는 경우에는 리스트 구조를 적용하기 곤란하다. 반면, 행렬 구조를 사용하여 일반 PN을 구현하는 경우에는 구조적인 문제와 자원의 낭비 및 자원 사용량의 폭증 문제가 발생한다. 본 논문에서는 행렬 구조의 참조표를 사용한 PN의 하드웨어 구현 시 발생하는 문제점을 해결할 수 있는 제약 조건을 일반 PN에 추가하여 하위 계열 PN을 제안하고, 이를 사용한 구현 방법을 제시하였다. 제안된 하위 계열 PN은 수행 단계의 증가를 가져올 수 있으나, 메모리 사용량을 크게 줄여, 구현성을 매우 높였다. 아울러 하위 계열 PN에 따라 구현 구조와 알고리듬을 제시하였으며, 각 구조에 따른 특성을 분석하였다. 제안된 구현 방법을 뒷받침하기 위해서, 컴퓨터를 사용한 자동 설계 환경을 구축하고, 제안된 PN의 구현 방법을 응용한 순차 제어기를 구성하여 실제 적용성을 검증하였다. 본 논문의 결과는 소프트웨어 구현 방법과 하드와이어 구현 방법의 장점을 잘 살릴 수 있는 PN의 구현을 가능하게 한다. 향후 본 논문에서 가정한 제약조건을 완화한 PN을 수행할 수 있는 구조의 연구가 요구된다.

참고문헌

[1] J. Park, N. Chang, G. S. Rho and W. H. Kwon, "Implementation of parallel logic solving algorithm for PLC based on data flow architecture," *IFAC Control Engineering Practice*, vol. 1, pp. 663-670, Aug., 1993.

[2] A. Falcione and B. H. Krogh, "Design recovery for relay ladder logic," *IEEE Control Systems*, pp. 90-98, Apr., 1993.

[3] K. Venkatesh, M. Zhou and R. J. Caudill, "Comparing ladder logic diagram and Petri nets for sequence controller design through a discrete manufac- turing system," *IEEE Trans. Industrial Electronics*, vol. 41, pp. 611-619, Dec., 1994.

[4] N. Chang, J. Park, K. Koo and W. H. Kwon, "Memory-based implementation of a Petri net and its application to a programmable controller," *Proc. IECON'93*, 1993.

[5] P. C. Baracos, R. D. Hudsin, L. J. Vroomen and P. J. A. Zsombor-Murray, "Advances in binary decision based programmable controllers," *IEEE Trans. Industrial Electronics*, vol. 25, Aug., 1988.

[6] H. Panetto, P. Lhoste, J. Petin and E. Bon, "Contribution of the grafcet model to synchrony in discrete events systems modeling," *Proc. IECON '94*, pp. 1527-1532, 1994.

[7] R. Zurawski and M. Zhau, "Petri nets and industrial applications: a tutorial," *IEEE Trans. Industrial Electronics*, vol. 41, pp. 567-583, Dec., 1994.

[8] J. Zaytoon, P. D. Loor and G. Villermain-Lecolier, "Using a real-time framework to verify the properties of grafcet," *Preprints of IFAC/IFIP Workshop on Algorithm and Architecture for Real-time Control*, pp. 223-238, Jan., 1995.

[9] M. D. Jeng and F. DiCesare, "A review of synthesis techniques for Petri nets with application to automated manufacturing system," *IEEE Trans. Systems, Man and Cybernetics*, vol. 23, pp. 301-312, Jan., 1993.

[10] I. Koh and F. DiCesare, "Modular transformation methods for generalized Petri nets and their application to automatic manufacturing systems," *IEEE Trans. Systems, Man and Cybernetics*, vol. 21, pp. 1512-1521, Nov., 1991.

[11] R. Valette, J. M. Courvoisier, J. M. Bigou and Alburquerque, "A Petri net based programmable logic controller," *Proc. IFIP conference on Computer Applications on Production and Engineering(CAPE)*, 1983.

[12] J. M. Colom, M. Silva and J. L. Villarroel, "On software implementation of Petri Nets and Colored Petri Nets using high level concurrent languages," *Proc. of 7th European Workshop and Application and Theory of Petri nets*, Jan., 1986.

[13] J. L. Briz, J. M. Colom and M. Silva, "Simulation of Petri nets and linear enabling functions," *Proc. IEEE Int. Conf. System, Man and Cybernetics,(San Antonio, USA)*, 1994.

[14] D. C. Hendry, "Heterogeneous petri net methodlolgy for design of complex controllers," *Proc. IEEE Comput. Digt. Tech.*, Sep., 1994.

[15] A. D. Stefano and O. Mirabella, "A fast sequence control device based on enhanced Petri nets," *Microprocessors and Microsystems*, vol. 15, pp. 179-186, May, 1991.

[16] M. Auguin, F. Boeri and C. Andre, "Systematic method of realization of interpreted Petri nets," *Digit. Process.*, 1980.

[17] M. Admaski, "Parallel controller implementation using standard PLD software," *in W. MOORE and W. LUK: 'FPGAs'(Abingdon EE & CS Books), edited paper from the international workshop on field programmable logic and applications*, 1991.

[18] J. Pardey and M. Bolton, "Logic synthesis of synchronous parallel controllers," *Proc. IEEE Conf. on Computer Design*, 1991.

[19] K. Bili'nski, M. Adamski, J. M. Saul and E. L. Dagless, "Petri-net-based algorithm for parallel-cotroller synthesis," *IEEE Proc.-Comput. Digit. Tech.*, Nov., 1994.

[20] K. Bili'nski, J. M. Saul and E. L. Dagless, "Efficient functional verification algorithm for petri-net-based parallel controller design," *IEEE Proc.- Comput. Digit. Tech.*, July, 1995.

[21] T. Kozlowski, E. L. Dagless, J. M. Saul, M. Adamski and J. Szajna, "Parallel controller synthesis using petri nets," *IEE Proc.-Comput. Digit. Tech.*, July, 1995.

[22] N. Chang, W. H. Kwon and J. Park, "FPGA-based implementation of synchronous petri· nets," to appear in *Proc. IECON'96*, Taipei, Taiwan, August, 1996.

[23] L. D. Coraor, P. T. Mulina and O. A. Morean, "A general model for memory-based finite-state machines," *IEEE Trans. Computers*, vol. C-36, pp. 175-184, Feb., 1987.

[24] R. F. Tinder, *Digital Engineering Design, A Modern Approach*. Prentice-Hall International, Inc., 1991.

[25] H. Murakoshi, M. sugiyama, G. Ding, T. Oumi, T. Sekiguchi and Y. Dohi, "A high speed programmable controller based on Petri net," *Proc. IECON '91*, pp. 1966-1971, 1991.

[26] H. Murakoshi, M. Sugiyama, G. Ding, T. Omui, T. Sekiguchi and Y. Dohi, "Memory reduction of fire unit for Petri net controlled multiprocessor," *Proc. IECON '91*, 1991.

[27] F. Anzai, N. Kwawhara, T. Takei, T. Watanabe, H. Murakosi, T. Kondo and Y. Dohi, "Hardware implementation of a multiprocessor system controller by Petri nets," *Proc. IECON '93*, pp. 121-126, 1993.

[28] T. Mutrata, "Petri nets : properties, analysis and applications," *Proceedings of IEEE*, vol. 77, pp. 541-580, Apr., 1989.
[29] R. David and H. Alla, "Petri nets for modeling of dynamic systems a survey," *Automatica*, vol. 30, no. 2, pp. 175-202, 1994.
[30] T. Murata and J. Y. Koh, "Reduction and expansion of live and safe marked graphs," *IEEE Trans. Circuits Systems*, vol. 27, pp. 51-76, Aug., 1993.
[31] I. G. Warnock, *Programmable controllers, operation and Application* Prentice-Hall Inc., 1988.

장 래 혁

1989년 서울대 제어계측공학과 졸업. 동대학원 석사(1992), 동대학 박사(1996). 1997년~현재 서울대학교 컴퓨터공학과 전임강사. 관심분야는 실시간 시스템, 디지털 시스템 하드웨어, 이산현상 시스템.

정 승 권

1993년 서울대학교 제어계측공학과 졸업(학사). 동대학원 석사(1995). 1995년~현재 동대학원 박사과정. 관심분야는 실시간 시스템, 산업용 통신망, 공장 자동화를 위한 컴퓨터 응용.

권 욱 현

1966년 서울대학교 전기공학과 졸업, 1972년 동대학원 석사, 1975년 Brown University 제어공학 박사, 1977년~현재 서울대학교 전기공학부 교수. 관심분야는 제어 및 시스템 이론, 이산 현상 시스템, 산업용 통신망, 분산 공정 제어.