비동기회로 합성을 위한 병행 프로그램의 페트리 넷으로의 변환

유동훈, 이동익
광주과학기술원 정보통신공학과
광주광역시 광산구 쌍암동 572번지
E-mail : {dhyoo, dilee}@kjist.ac.kr

Translating Concurrent Programs into Petri Nets for Synthesis of Asynchronous Circuits

Dong-Hoon Yoo and Dong-Ik Lee
Department of Information & Communications, K-JIST
572 Ssangam-Dong, Kwangsan-Ku, Kwangju, 506-712
E-mail : {dhyoo, dilee}@kjist.ac.kr

Abstract

We introduce a high level synthesis methodology for automatic synthesis of asynchronous circuits from a language based on CSP. The input language is a high level concurrent algorithmic specification that can model complex concurrent control flow, logical and arithmetic computation and communications between them. This specification is translated into Petri net which has actions. These actions are refined to synthesize the controllers and to allocate the data resources. We use the automatic synthesis through Signal Transition Graphs (STGs) that allows to take advantage of logic synthesis methods to optimize the circuit.

1 서론

최근, VLSI 기술의 복잡성이 증대됨에 따라 나타나는 문제점들을 해결하기 위한 대안으로써 비동기회로 설계 기법이 대두되고 있다. 비동기회로는 동기회로에 비해 모듈성(modularity), 저전력 소비, 평균적인 계산 시간, 클럭 분배 문제의 제거, 견고성 등의 장점을 가진다. 이러한 비동기회로를 기술하기 위한 방법으로 CSP (Communicaing Sequential Processes)[1]와 페트리 넷(Petri Net)[2]이 널리 사용되어지고 있다.

CSP를 기반으로 한 기법에서는 시스템의 행동을 채널을 이용하여 서로 통신을 하는 프로세스들로 나타내고, 이를 기술하기 위한 병행 프로그램 언어를 사용하고 있다. 이러한 병행 프로그램은 컴파일러 이론에서 사용되는 지역적인 최적화 방법(peephole optimization)과 구문 직접 변환(syntax-directed translation)을 통해 회로로 변환이 된다[3][4]. 이 기법은 회로 설계자에게 회로의 행동을 편리하게 기술 할 수 있는 방법을 제공해 주지만, 일반적으로 여분의 기능을 가진 최적화 되지 못한 회로를 생성해 낸다.

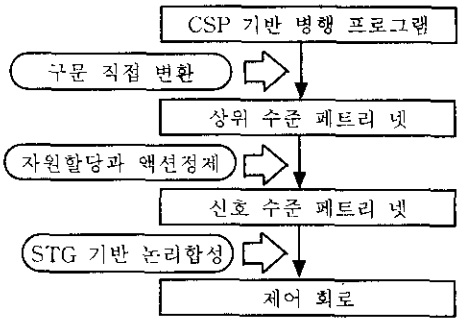
페트리 넷을 기반으로 하는 기법[5]에서는 신호전이 그래프(Signal Transition Graph : STG)[6]를 이용하여 회로를 기술한 후에 논리합성을 수행한다. 이 기법은 전역적으로 최적화된 회로를 자동적으로 생성해 낼 수 있지만, 낮은 수준(신호 전이 수준)의 회로 명세 기법이므로 사양명세가 번잡해지는 불편함이 있다.

본 논문에서는 이 두 가지 기법의 장점을 동시에 고려한 비동기회로의 합성 방법을 제안한다. 즉, 회로의 행동을 기술하기 위한 방법으로 CSP 기반의 병행 프로그램 언어를 제공하고, 이를 페트리 넷으로 변환시킨 후 논리합성을 통해 회로를 생성한다. 기존의 방법[7]에서는 [4]에서 사용된 각각의 컴포넌트들을 STG로 변환시킨 후 이를 다시 페트리 넷 합성(Petri net composition)을 통해 하나의 STG를 생성한 후에 논리합성을 수행하는 비효율적인 방법이었다. 본 논문에서는 병행 프로그램에서 직접 페트리 넷을 생성해내고 이를 이용하여 논리합성을 수행하는 방법을 제안한다.

2장에서는 합성 시스템의 전체적인 개요를 설명하고, 3장에서는 본 논문에서 사용되는 기본적인 모델이 대하여 설명한다. 4장에서는 자세한 합성 기법에 대하여 순차적으로 설명한다. 결론 및 향후 과제는 5장에 나타나 있다.

2 개요

본 논문에서 제안한 기법의 전반적인 개요가 [그림 1]에 나타나 있다. 그림에서 알 수 있듯이 회로 설계자



[그림 1] 개요

는 단지 프로그래밍만을 수행함으로써 회로를 생성해낼 수 있다.

회로를 먼저 병행 프로그램 언어로 기술 한 후 상위 수준 페트리 넷으로 변환한다. 상위 수준 페트리 넷은 자원 할당과 액션 정제를 통해 신호수준의 페트리 넷으로 변환이 되는데, 이 단계에서 데이터 패스와 제어 회로가 분리된다. 본 논문에서는 단지 제어 회로의 생성과 데이터 패스와의 제어 신호 전달에 대해서만 논의한다. 마지막으로 기존의 STG를 기반으로 한 논리합성 기법에서 신호 수준 페트리 넷을 이용하여 제어 회로를 생성한다.

3 회로의 모델링

3.1 병행 프로그램 언어

회로의 기술에 사용되는 병행 프로그램 언어의 기본적인 구문을 CSP와 Dijkstra의 조건 명령문(guarded command)[8]에 기반 하여 다음과 같이 BNF형식으로 정의한다.

[정의 1] 병행 프로그램 언어 구문

c를 채널 지정어(channel identifier), v를 변수 지정어(variable identifier)라고 했을 때 병행 프로그램 언어의 기본적인 구문은 다음과 같다.

$$
\begin{array}{lll}
(process) & P ::= & \textbf{\textit{skip}}\ (the\ empty\ command) \\
 & & \mid v := E\ (assignment) \\
 & & \mid I\ (input\ command) \\
 & & \mid O\ (output\ command) \\
 & & \mid P_1 ; P_2\ (sequence) \\
 & & \mid P_1 \| P_2\ (parallel\ composition) \\
 & & \mid \textbf{if}\ G\ \textbf{\textit{fi}}\ (choice) \\
 & & \mid \textbf{\textit{do}}\ G\ \textbf{\textit{od}}\ (loop) \\
(expr) & E ::= & v \mid \neg E \mid E_1 \wedge E_2 \mid E_1 \vee E_2 \\
 & & \mid \textbf{\textit{true}} \mid \textbf{\textit{false}} \\
(input) & I ::= & c? \mid c?v \\
(output) & O ::= & c! \mid c!E \\
(guard) & G ::= & E \rightarrow P \mid I \rightarrow P \\
 & & \mid E \,\&\, I \rightarrow P \mid G_1 \square G_2
\end{array}
$$

병렬합성 제어구문인 $P_1 \| \ldots \| P_n$에서 만약 P_i가 P_j에 대응하는 IO 명령문을 포함하고 있다면, $i \neq j$ 이고 P_j 또한 P_i에 대응하는 IO 명령문을 포함하고 있어야 한다.

```
process BUF1(A?bool, B!bool)
{
    bool x;

    forever {
        A?x ; B!x
    }
}
```

[그림 2] 한 개의 변수를 저장하는 버퍼

';' 연산자는 피연산자들을 순서적으로 수행하고, '||' 연산자는 피연산자들을 병행적으로 수행한다. 병행 연산자의 기능에 대해서는 4.3절에서 상세히 다루도록 하겠다. **if** *G* **fi** 제어문은 G의 조건 중에서 그 값이 참(true)이 되는 명령문을 수행하며, 모든 조건의 값이 거짓(false) 일 때에는 수행을 마친다. **do** *G* **od** 제어문은 G의 모든 조건이 거짓 값을 가질 때까지 참의 값을 갖는 명령문들을 반복 수행한다. 프로세스간의 모든 통신은 채널을 통해 이루어지며, 데이터의 이동 또한 채널을 통해 이루어진다. 회로를 기술하는 병행 프로그램은 채널을 통해 서로 통신을 하는 하나 이상의 프로세스(process)로 구성된다.

이러한 구문을 기반으로 하여 정의된 병행 프로그램 언어를 이용하여 변수 한 개를 저장할 수 있는 버퍼에 관한 회로를 [그림 2]와 같이 기술 할 수 있다. 기술된 회로는 채널 *A*로부터 1 비트의 데이터를 받아들여 변수 *x*에 저장한 후 다시 이 데이터를 채널 *B*로 내보내는 기능을 갖는다.

3.2 페트리 넷

각 프로세스는 레이블 페트리 넷으로 변환된다. 마킹(marking)은 넷의 상태(state)를 표시하는데 이하에서는 마킹과 동의어를 혼용해서 쓴다.

[정의 2] 레이블 페트리 넷 (Labeled Petri Net)

레이블 페트리 넷 N은 (A, P, δ, M_0)로 나타낸다. A는 액션(action)들의 집합, P는 장소(place)들의 집합, $\delta \subseteq 2^P \times A \times 2^P$는 전이(transition)관계, $M_0: P \rightarrow N$은 초기 마킹(initial marking)을 각각 나타낸다. (N은 자연수의 집합)

[정의 3] 전이 발생 (Transition Firing)

각 전이 (p, a, q)는 상태(state) M에 있을 때 만약 $\forall p' \in p: M(p') > 0$ 이면 발생(fire) 할 수 있다. 전이 발생 후의 다음 상태 M'을 다음과 같이 정의한다.

$$
M'(p') = \begin{cases} \text{만약 } p' \in p \backslash q \text{ 이면, } M(p') - 1 \\ \text{만약 } p' \in q \backslash p \text{ 이면, } M(p') + 1 \\ \text{그 밖의 경우이면, } M(p') \end{cases}
$$

본 논문에서 사용되는 모델의 액션 A = SUKURU $\{\varepsilon\}$은 다음과 같은 형을 가진다.

• S = (IUO)×{+,-}는 입출력 선(wire) I 와 O에서 발생하는 저수준 신호 전이들의 집합이다. s+와 s-는 각각 신호 s의 양전이(rising transition)와 음전이(falling transition)를 나타낸다.

• K에 해당하는 액션은 덧셈, 비교 등과 같은 연산이나 논리 표현들을 나타낸다. 이러한 액션들은 데이터 처리 연산으로 생각할 수 있다.
• R = C×{?,!}(×X)는 통신을 하는 액션들의 집합을 나타낸다. C는 프로세스간의 통신이 이루어지는 채널들의 집합이다. "?"와 "!"는 CSP 형태의 동기식 송신과 수신을 각각 나타낸다. 데이터 X는 채널을 통해 전달되어 질 수 있다.
• ε는 단순히 공진이(dummy transition)를 나타낸다.

4 합성 기법 (Synthesis Methodology)

4.1 병행 프로그램의 상위 수준 페트리 넷으로의 변환

회로의 기능을 기술한 병행 프로그램을 먼저 상위 수준의 페트리 넷으로 변환한다. 각 순차적 제어 프로그램(sequential control program) γ는 마킹이 된 초기 장소(place) *first(γ)*와 마킹이 되지 않은 장소(place) *last(γ)*로 구성된 넷(net)으로 나타내어진다.

[정의 4] 제어 프로그램의 변환

$P = P_1||..||P_n$을 병행 프로그램이라고 할 때, P를 다음과 같이 변환한다.

- ***if*** $\beta_1 \rightarrow S_1 \square ... \square \beta_n \rightarrow S_n$ ***fi***는 $[\beta_1:S_1 \square ... \square \beta_n:S_n \square \neg B_1 \wedge ... \wedge \neg B_n]$로 변환한다.
- ***do*** $\beta_1 \rightarrow S_1 \square ... \square \beta_n \rightarrow S_n$ ***od***는 $*[\beta_1:S_1 \square ... \square \beta_n:S_n \mid \neg B_1 \wedge ... \wedge \neg B_n]$로 변환한다.

B는 β에 포함된 논리식이다. $*[a|B]$는 a를 반복 수행하다가 B를 만족하면 수행을 마침을 의미한다.

[정의 5] 변환 함수 (compilation function) C [9]

γ을 순차적 제어 프로그램이라고 할 때

(1) 마킹이 되지 않은 넷을 다음과 같이 정의한다.

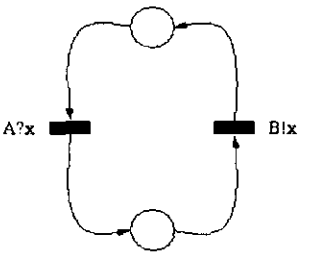
- *만약 $\gamma = a$ $(a \in A)$ 이면 $C(\gamma)$는 first(γ), last(γ)의 두 장소와 전이 a를 갖는다. 전이 관계는 (first(γ), a, last(γ))로 나타낸다.*
- *만약 $\gamma = \gamma_1;\gamma_2$ 이면 $C(\gamma)$는 $C(\gamma_1)$의 $last(\gamma_1)$과 $C(\gamma_2)$의 $first(\gamma_2)$을 동일하게 함으로써 얻어진다. 이때 $first(\gamma) = first(\gamma_1)$ 이고 $last(\gamma) = last(\gamma_2)$로 정의한다.*
- *만약 $\gamma = \gamma_1 \| \gamma_2$ 이면 $C(\gamma)$는 $C(\gamma_1)$과 $C(\gamma_2)$의 $first(\gamma_1)$과 $first(\gamma_2)$, $last(\gamma_1)$과 $last(\gamma_2)$를 각각 동일하게 함으로써 얻어진다. 이때 $first(\gamma) = first(\gamma_1)$ $(= first(\gamma_2))$ 이고 $last(\gamma) = last(\gamma_1)$ $(= last(\gamma_2))$으로 정의한다.*
- *만약 $\gamma = *[\gamma_0|B]$ 이면 $C(\gamma)$는 $C(\gamma_0)$과 $C(B)$의 $first(\gamma_0)$, $last(\gamma_0)$, $first(B)$를 동일하게 함으로써 얻어진다. 이때 $first(\gamma) = first(\gamma_0)$ $(= last(\gamma_0) = first(B))$ 이고 $last(\gamma) = last(B)$로 정의한다.*

(2) 페트리 넷 $N(\gamma)$은 $C(\gamma)$에 초기 마킹 M_0를 다음과 같이 정의함으로써 얻어진다.

$$M_0(s) = \begin{cases} \text{만약 } s = first(\gamma) \text{ 이면,} & 1 \\ \text{그 밖의 경우에는,} & 0 \end{cases}$$

[정의 4]와 [정의 5]를 이용하여 [그림 2]의 회로를 상위 수준 페트리 넷으로 변환한 결과가 [그림 3]에 나타나 있다.

4.2 상위 수준 페트리 넷의 신호 수준 페트리 넷으로의 변환



[그림 3] 상위 수준 페트리 넷

상위 수준의 페트리 넷으로는 논리 합성을 할 수 없기 때문에 액션 A = SUKURU{ε} 중에서 K와 R에 해당하는 액션은 자원 할당(resource allocation)과 액션 정제(action refinement)를 통해 각 액션을 논리 합성이 가능한 신호 수준으로 변화시켜 주어야 한다. 자원 할당은 연산(computations), 채널, 변수 등에 직접적인 하드웨어 자원을 할당하는 것을 말한다. 이때 할당되는 하드웨어 자원으로는 매크로모듈(macromodule)[4][10]을 사용할 수 있다. 하드웨어 자원과 제어 회로와의 통신은 요구(request)와 응답(acknowledgement) 두 개의 선(wire)에서 4 단계 시그널링(4-phase signaling)[11]과 번들드 데이터 프로토콜(bundled data protocol)[10]을 이용하여 이루어진다.

하드웨어 자원이 할당된 후에는 매크로확장(macro-expansion)[3][12]을 통하여 상위 수준의 액션을 하드웨어 자원을 제어할 수 있는 신호로 변환하여 준다. 이 단계를 액션 정제(action refinement)라고 한다. 이 단계를 거친 후에는 각 액션이 신호 전이(signal transition)을 나타내는 페트리 넷을 얻을 수 있다. [그림 2]에 대한 신호 수준 페트리 넷이 [그림 4]에 나타나 있다.

4.3 병렬 합성 (Parallel Composition)

P_1, P_2을 병행 프로그램이라고 할 때, 변환된 신호 수준 페트리 넷을 각각 N_1, N_2라고 하자. 페트리 넷에서 전이(transition)는 모든 입력 장소(place)가 토큰(token)을 가지고 있을 때에만 발생(fire) 할 수 있기 때문에 동기화 수법으로 이용된다. 두 페트리 넷을 병렬 합성을 할 경우에는 동일한 신호에 대해서 동기가 이루어져야 한다. 또한, 같은 액션 이름을 가진 전이가 하나 이상 있을 수 있기 때문에 모든 조합의 경우를 생각해 주어야만 한다.

[정의 6] 병렬 합성 (Parallel Composition)

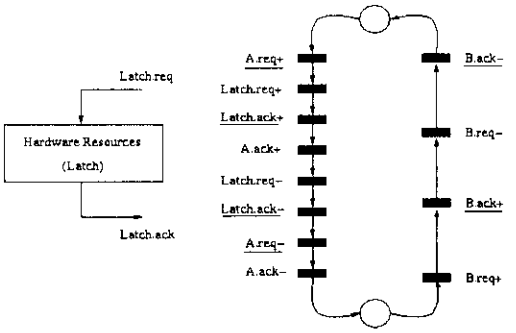
$i \in \{1,2\}$에 대해 $N_i = (A_i, P_i, \rightarrow, M_{0i})$을 $P_1 \cap P_2 = \varnothing$인 페트리 넷이라고 하자. 넷의 병렬 합성을 다음과 같이 정의한다.

$$N_1 \| N_2 = (A_1 \cup A_2, P_1 \cup P_2, \delta', M_{01} \cup M_{02})$$

여기에서

$$\delta' = \{(I,a,O) \in \delta_1 \cup \delta_2 \mid a \notin A_1 \cap A_2\}$$
$$\cup \{(I_1 \cup I_2, a, O_1 \cup O_2) \mid a \in A_1 \cap A_2 \wedge (I_i, a, O_i) \in \delta_i\}$$

이다.



[그림 4] 신호 수준 페트리 넷

4.4 논리 합성 (Logic Synthesis)

여러 단계의 변환을 거친 후 얻게 되는 신호 수준의 페트리 넷을 이용하여 논리 합성을 수행 할 수 있다. 본 논문에서는 기존의 STG 기반의 논리 합성 도구인 **petrify** [5]를 이용하여 속도 독립 회로(speed independent circuit)을 생성한다. 페트리 넷의 안전성(safeness), 생존성(liveness), 유한성(boundedness), 일관성(consistency), 지속성(persistency), 완전상태코딩(complete state coding: CSC)등을 이 단계에서 **petrify**를 이용하여 검사한다. 신호 수준 페트리 넷이 완전상태코딩의 조건을 만족하지 못할 경우에는 **petrify**를 이용하여 해결하여 준다. 최종적으로 생성된 [그림 2]에 대한 제어 회로가 [그림 5]에 나타나 있다.

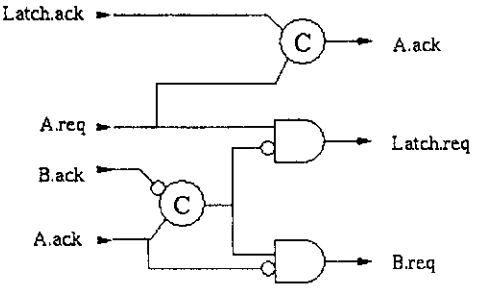
5 결론 및 향후 과제

본 논문에서는 상위 수준의 프로그래밍 언어로부터 전역적으로 최적화 된 제어 회로를 생성하고 동시에 하드웨어 자원을 할당할 수 있는 방법을 제안하였다. 회로 설계자는 프로그래밍 활동을 통하여 최적화 된 제어 회로를 생성할 수 있다.

프로그래밍 언어로부터 변환된 페트리 넷은 여러 지연 모델(*bounded delay model* or *unbounded delay model*)에 기반 한 기법을 이용하여 논리 합성 될 수 있다. 또한, 기존의 방법[12]을 이용하여 회로의 성능 분석, 최적화, 시뮬레이션 등을 생성된 페트리 넷을 입력으로 하여 수행할 수 있다.

병행 프로그램에서 페트리 넷으로의 변환에 관한 형식 검증(formal verification)과 데이터 패스(data path)의 자동 생성, 상위 수준 페트리 넷을 이용한 자원의 스케줄링(resource scheduling) 등은 계속 연구가 이루어져야 할 것이다.

상위 수준 병행 프로그램 언어의 컴파일러는 현재 C++와 flex, bison을 이용하여 제작 중이다.



[그림 5] 버퍼의 제어 회로

참고문헌

[1] C.A.R. Hoare, *"Communicating Sequential Processes"*, Prentice Hall International, 1989

[2] J.L. Peterson, *"Petri Net Theory and the Modeling of Systems"*, Prentice Hall, Englewood Cliffs, NJ, 1981

[3] A.J. Martin, *"Programming in VLSI: From Communicating Processes to Delay-Insensitive Circuits"*, In C.A.R. Hoare, editor, Developments in Concurrency and Communication, UT Year of Programming Series, pages 1-64, Addison-Wesley, 1990

[4] K. van Berkel, *"Handshake Circuits: an Asynchronous Architecture for VLSI Programming"*, volume 5 of International Series on Parallel Computation, Cambridge University Press, 1993

[5] J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev, *"Synthesizing Petri Nets from State-Based Models"*, In proceedings of ICCAD '95, pages 164-171, November 1995

[6] T.-A. Chu, *"Synthesis of Self-timed VLSI Circuits from Graph-theoretic Specifications"*, PhD Thesis, MIT, 1987

[7] M.A. Peña and J. Cortadella, *"Combining Process Algebra and Petri Nets for the Specification and Synthesis of Asynchronous Circuits"*, In proceedings International Symposium on Advanced Research in Asynchronous Circuits and Systems, pages 222-232, 1996

[8] E.W. Dijkstra, *"A Discipline of Programming"*, Prentice Hall, 1976

[9] E. Best, "COSY: Its Relation to Nets and to CSP", In W. Brauer, W. Reisig, and G. Rozenberg, editors, Petri Nets: Applications and Relationship to Other Models of Concurrency, number 255 in LNCS: Advances in Petri Nets, pages 416-440, Springer-Verlag, 1986

[10] I. Sutherland, *"Micropipelines"*, Communications of the ACM (June 1989), The 1988 ACM Turing Award Lecture

[11] C.L. Seitz, *"System Timing"*, In C.A. Mead and L.A. Conway, editors, Introduction to VLSI Systems, chapter 7, Addison-Wesley, 1980

[12] P.N. Kudva, "Synthesis of Asynchronous Systems Targeting Finite State Machines", PhD Thesis, Department of Computer Science, The University of Utah, 1995