論文 89-26-4-17

FSM 설계를 위한 하드웨어 흐름도와 하드웨어 기술 언어에 관한 연구

(A Study on a Hardware Flow-Chart and Hardware Description Language for FSM)

李 丙 鎬*, 趙 仲 彙**, 鄭 正 和*

(Byung Ho Lee, Joong Hwee Cho and Jong Wha Chong)

要 約

본 논문에서는 논리 설계 자동화를 위한 레지스터 전송 레벨의 하드웨어 흐름도와 하드웨어 기술언어 SDL-Ⅱ (symbolic description language)를 각각 제안한다.

SDL-Ⅱ는 일반화된 FSM (finite state machine)의 동작 및 구조적 특성을 제안하는 하드웨어 흐름도로 표현하고 이의 각 기호에 1대 1 대응하며 제어부와 데이타 전송부를 함께 기술하도록 구문을 설정한다. 또한 여러가지 설계 요구조건을 하드웨어 흐름도로 표현하고 이를 SDL-Ⅱ로 기술하여 본 논문의 유효성을 보인다.

Abstract

This paper describes hardware flow-chart and SDL-II, which are register-transfer level, to automate logic design.

Hardware flow-chart specifies behavioral and structural charaterstics of generalized FSMs (Finite State Machine) using the modified ASM (Algorithmic State Machine) design techniques. SDL-II describes the hardware flow-chart which specifies the control and the data path of ASIC (Application Specific IC).

Also many examples are enumerated to illustrate the features of hardware flow-chart and SDL-II.

Ⅰ. 서 론

디지탈 시스템의 설계는 스위칭 이론이 대두된 이래 계속 변화되고 있는 과제이며 최근들어 VLSI를 이용한 ASIC (application specific integrated circuit)이 통신, microprocessor 분야 등에서 다양하게 요구되고 또한 이의 집적도가 증가함에 따라 이에 대한 설계 자동화가 여러 관점에서 요구되고 있다.[1]

최근에는 반도체 기술의 진보에 의해 VLSI 칩 위에 실현할 수 있는 시스템의 규모가 대규모화함에 따라 기존의 하드웨어 설계 방법으로 설계하는 경우 설

*正會員, 漢陽大學校 電子工學科
(Dept. of Elec. Eng., Hanyang Univ.)
**正會員, 仁川大學校 電子工學科
(Dept. of Elec. Eng., Inchun Univ.)

接受日字 : 1988年 9月 29日
(※ 본 연구는 1986년도 한국과학재단 일반연구비의 지원으로 이루어졌음.)

계시간이 오래 걸릴 뿐만아니라 설계의 정확성을 판단하고 이를 수정하는데 많은 어려움이 따르게 되었다. 따라서 복잡하고 반복적인 하드웨어 설계를 컴퓨터 시스템을 사용하여 체계적으로 행함으로써 최적설계라는 목표를 달성하지 못하여 다소의 생산비용의 증가는 감수하더라도 "정확한 설계"를 "짧은 기간"에 완성하여야 할 필요가 있다는 것이 설계 자동화 시스템 실현의 주요한 동기가 되고 있다. 이와 같은 디지탈 시스템의 설계 자동화는 최근들어 "silicon compiler"라는 이름[2~11]으로 많은 연구가 진행되고 있는데 이는 크게 1) 논리 합성(logic synthesis)과 2) 레이아웃 합성(layout synthesis)으로 구분된다.[3]

이중 레이아웃 합성은 gate-array[12~13], standard cell[14], PLA 등과 같은 분야로 나뉘어 많은 연구가 진행되어 어느정도 국내에서도 실용화에 수렴하고 있다.[13]

그런데 논리 합성은 설계 요구조건의 다양성으로 인하여 알고리듬 레벨, 또는 레지스터 전송 레벨 등에서 설계 요구조건을 기술하는 하드웨어 기술 언어(HDL: hardware description language)[7]의 구문구조가 복잡하며 또한 이로부터 생성한 게이트 레벨의 논리회로도가 설계 전문가에 의한 설계에 비해 설계 면적의 증가, 설계 최적화의 어려움 등의 이유로 아직 많은 연구가 요구되고 있다.[10]

알고리듬 레벨에서 기술된 HDL은 1) State synthesis 2) Operator to function unit binding 3) Variable to register binding 4) Connection binding 등과 같은 micro-architecture design[8~16]을 거쳐 레지스터 전송 레벨로 변환된다.[4] 또한 레지스터 전송 레벨에서 기술된 HDL은 데이타 전송부와 제어부의 분류, 조건의 집약, bit-wise, macro expantion 등과 같은 과정을 거쳐 게이트 레벨의 HDL로변환된다.[15]

따라서 silicon compiler의 논리 합성을 위와 같이 수행하기 위해서는 레지스터 전송 레벨의 HDL[16]과 이를 게이트 레벨의 논리회로도로 변환하는 컴파일러의 설계가 요구된다.[3] 이를 위해 AHPL[17~19],DDL[19], SDL[19] 등과 같은 HDL과 이들의 컴파일러가 발표되어 있으나 이들이 갖는 각각의 제약성으로 인하여 시스템의 표현에 문제점이 지적되고 있다.[19] 또한 [21-22]등과 같은 HDL은 제어부 설계를 주목적으로 하여 언어가 정의되므로 제어부는 물론 데이타 전송부에 대한 설계요구가 매우 중요시되는 ASIC의 설계를 위하여는 많은 문제가 있다.[10]

따라서 본 논문에서는 설계 요구 조건을 레지스터 전송 레벨에서 동작 및 구조적 특성을 함께 분석하여 표현하는 하드웨어 흐름도를 ASM (algorithmic state machine)[20] 및 MASM (modified ASM)[19] 도표를 변형한 하드웨어 흐름도 SAD (state analysis diagram)를 제안하고 이 흐름도의 각 기호에 1 대 1 대응하며 일반화된 FSM을 계층적으로 기술할 수 있는 하드웨어 기술언어 SDL-Ⅱ를 제안한다.

SDL-Ⅱ는 시스템을 분석하여 설계가 완료되어 design library에 게이트 레벨의 HDL인 NDL (network description language)[19]로 저장되어 있는 모듈을 이용하면서 새로이 설계되거나 design library의 정보를 수정하여 사용하고자 하여 기술되는 모듈 기술부와 2개 이상의 독립적인 하위 레벨의 모듈이 존재하면 이들을 하나의 모듈로 구현하기 위하여 기술되는 상위 레벨의 모듈 기술부로 구성된다. 한편, 하위 레벨의 모듈 기술부와 상위 레벨의 모듈 기술부의 분류는 단지 계층적 설계의 개념을 도입하기 위한 것으로 이들에 대한 SDL-Ⅱ의 기술상의 차이는 없다. 모듈 기술부는 입, 출력 변수 등의 선언부와 하드웨어 흐름도 기술부로 나누어 각각 기술하는데 하드웨어 흐름도 기술부는 상태(state) 블럭으로 나누어 기술하도록 구문을 설정한다.

또한 실제의 설계 요구조건을 하드웨어 흐름도로 표현하고 이를 SDL-Ⅱ로 기술함으로써 본 논문의 유효성을 보인다.

Ⅱ. 하드웨어 흐름도와 하드웨어 기술 언어

1. 하드웨어 흐름도 : SAD

ASM을 이용한 디지탈 시스템의 설계 방법은 FSM을 쉽게 표현할 수 있어 mealy와 moore machine 의 제어부를 설계할 수 있다.[20~21] 한편 [19]에서는 제어부 및 데이타 전송부를 함께 표현할 수 있도록 ASM 도표를 수정한 MASM 도표를 제안하였으나 구조 특성은 표현할 수 없으며 동작 특성을 표현함에 있어서도 기호 사용의 제약성으로 인하여 범용적 사용에 문제가 있다. 이를 해결하기 위해 본 논문에서는 [19~20]의 흐름도를 확장하여 PAD (problem analysis diagram)와 유사한 형태를 갖는 레지스터 전송 레벨의 하드웨어 흐름도 SAD (state analysis diagram)을 제안한다. SAD는 제어부와 데이타 전송부를 함께 표현하도록 FSM을 일반화시켜 순서 논리회로, 조합 논리회로의 동작적 특성은 물론 구조적인 특성도 표현하도록 기호를 설정하는데 표 1과 같다.

일반화된 FSM의 출력 변수에 대하여는 2개의 식이 성립되는데 출력변수=출력함수(상태변수)와 같은 moore type의 출력을 표시하기 위해 상태 기호를 설정하고 출력변수=출력함수(상태변수, 입력변수)와 같은 mealy type의 출력을 표시하기 위해 조건 출력 기호를 설정한다. 조건에 따른 상태 천이와 조건

표 1. SAD의 기호
Table 1. The symbols of SAD.

기 호	이 름	의 미
Moore 출력	상태기호	FSM에서 임의의 상태가 시작할때 존재하며 Moore type의 출력을 기술한다.
Mealy 출력	조건출력 기호	Mealy type의 출력이 있는 경우에 존재한다.
조건 T F	단수조건 판단기호	if 조건 then goto 기호1 else goto 기호2와 같은 의미를 표시한다.
조건1 T T 조건n T	복수조건 판단기호	if 조건1 then goto 기호1. if 조건2 then goto 기호2. ..... if조건n then goto 기호n 과 같은 의미를 표시한다.
경우1 조건 경우2 경우n	경우조건 판단기호	switch 조건 case 경우1 : goto 기호1 ..... case 경우n : goto 기호n과 같은 의미를 표시한다.
	구조기호	입,출력 특성만을 알고 있는 구조블럭을 표시할때 사용한다.
	선언기호	임의의 모듈에서 사용되는 각종 변수들을 선언할때 사용한다.

출력 기호에 앞서 임의의 조건식에 대한 참과 거짓을 판정하기 위해 3가지 조건 판단 기호를 설정한다.

그리고 설계 요구조건을 분석할 때 입,출력 변수만의 구조적 블럭으로 표현되는 function unit를 표현하기 위해 구조 기호를 설정하고 설계하고자 하는 모듈의 입,출력 변수 및 클럭 등을 선언하는 선언부 기호를 설정한다. 또한 상태들의 천이 연결을 표현하기 위한 상태 연결 기호를 설정한다.

한편 FSM에서 상태가 이동하는 모양은 다음상태=상태 천이 함수(현재상태) 또는 다음상태=상태 천이 함수(현재상태, 입력변수)로 하드웨어 흐름도에서 임의의 상태 블럭 구성은 상태 블럭의 첫번째 기호인 상태 기호는 반드시 하나를 포함하고 조건 출력 기호와 조건 판단 기호 및 구조 기호는 선택적으로 zero번 이상 포함하며 상태 블럭의 각 경로의 마지막에는 다음 상태로의 연결을 위한 상태 연결 기호를 설정한다. 한편 하나의 모듈에 선언 기호는 하나씩 설정한다.

2. 새로운 하드웨어 기술 언어 : : SDL-Ⅱ

디지탈 시스템의 규모가 증대하고 다양화함에 따라 가산기, 크기 비교기 등과 같은 기본적인 function unit는 설계가 완료되어 design library에 보관되어 있다가 필요할 때 이용되는 설계 방식이 많이 이용되고 있다.

그러나 시스템 설계시 필요로 하는 모든 function unit가 design library에 설계완료되어 있지는 않을 것이며 어떠한 function unit는 설계자에 의해 변경될 필요도 있다.

따라서 시스템 설계자가 필요에 따라 design library에 존재하는 function unit만으로 혹은 전혀 존재하지 않는 function unit만으로 혹은 전자와 후자가 혼합된 형태로 설계할 수 있도록 설계 요구 조건을 기술할 수 있는 하드웨어 기술 언어가 요구되었다.

이와 같은 이유로 본 연구에서 제안하는 레지스터 전송 레벨의 하드웨어 기술 언어 SDL-Ⅱ는 다음과 같이 크게 2부분으로 나누어 기술될 수 있는데 SDL-Ⅱ에서는 하위 레벨 모듈 기술부는 반드시 포함되어야 하며 상위 레벨 모듈 기술부는 하위 레벨 모듈 기술부가 2개 이상 존재하는 경우에만 기술하나 하위, 상위 레벨 모듈 기술부의 기술에서의 차이는 없다.

(1) Library에 존재하지 않는 function unit로 부분적으로 library를 이용하면서 표 1의 기호에 1대 1 대응되는 구문 구조에 의해 기술되는 하위 레벨 모듈 기술부

(2) 하위 레벨 모듈 기술부에 의해 기술된 function unit와 부분적으로 library의 모듈을 이용하면서 표 1의 기호에 1대 1 대응되는 구문구조에 의해 기술되는 상위 레벨 모듈 기술부

SDL-Ⅱ 구문의 정의에 앞서 SDL-Ⅱ에서 사용하는 종단 기호에 대한 의미를 표로 작성하면 표 2와 같다.

한편 SDL-Ⅱ의 구문 구조는 다음과 같이 설정하는데 이에 대해 확장된 BNF 형태로 표현하면 부록과 같다.

1) 상태(state) 기호와 SDL-Ⅱ 구문

Moore type의 출력을 기술하기 위한 상태 기호가 하드웨어 흐름도에 존재하면 이에 대응하는 SDL-Ⅱ 구문은

상태 기호 이름 moore type의 출력들; →상태 기호가 수행한 후 수행될 기호 이름.

이다.

Moore type의 출력이 존재하지 않는 상태 기호는 모듈 기술의 선언부에 기술된 클럭을 1개 소비하면서 제어 순서를 설정하기 위한 것으로

상태 기호 이름→상태 기호가 수행된 후 수행될 기호 이름.

이다. 한편,

상태 기호 이름 STOP.

의 형태로 기술된 SDL-Ⅱ 구문은 모듈의 동작을 종료하기 위한 문장으로 하드웨어 구성에는 영향이 없

표 2. SDL-Ⅱ의 기호표
Table 2. The symbols of SDL-Ⅱ.

기 호	의 미
+, −, *, %	연산기호
/	1) 연산기호 2) 조건판단에서 조건과 천이기호사이의 구분기호
>, <, =>, =<, <>, ==	관계연산자
?	사용자 정의 임의연산자
!, &, ¦, @	각각 NOT, AND, OR, XOR 의미의 논리연산자
←	데이타 전송부의 전달기호
=	데이타 전송부의 연결기호
*\A	레지스터 A의 모든 bit를 논리적으로 AND한 논리값
+\B	레지스터 B의 모든 bit를 논리적으로 OR한 논리값
n1#n2	n2(0 또는 1)가 n1(>1)개 연속됨을 표시
n1 : n2	레지스터의 bit번호중 n1부터 n2까지를 표시하는 것으로 SDL-Ⅱ에서는 MSB의 bit번호를 0으로 하며 n1=n2인 경우는 n1 : n2 대신에 n1만으로 기술
,	1) 레지스터의 dit concatenation 표시로 이를이용하면 shift와 rotate를 수행할 수 있다. 2) 조건 판단 기호의 조건식들 사이 또는 천이 목적지 기호사이의 구분표시
;	상태기호 또는 조건출력기호의 출력사이 및 출력과 천이문사이의 구분 또는 변수 선언부의 변수 구분표시
.	변수 선언부의 각 type 선언문장 및 각 기호의 기술종류 표시
→	무조건 goto문 표시
{ }	하위 레벨 모듈 및 design library의 기술을 위한 operand 및 operator의 기술

으므로 조합 논리 회로의 기술은 이 문장을 포함한 또 다른 하나의 상태 기호 SDL-Ⅱ 구문으로 기술한다.

2) 조건 출력(conditional output) 기호와 SDL-Ⅱ 구문

Mealy type의 출력을 기술하기 위한 조건 출력 기호가 하드웨어 흐름도에 존재하면 이에 대응하는 SDL-Ⅱ 구문은

조건 출력 기호 이름 mealy type의 출력들; → 조건 출력 기호후 수행될 기호 이름.

이다. 한편, 조건 출력 기호는 상태 기호와는 달리 mealy type의 출력이 존재하는 경우에만 기술한다.

3) 조건 판단(condition-decision) 기호와 SDL-Ⅱ 구문

조건이 하나인 즉, IF 조건식 THEN 기호 이름1 ELSE 기호 이름2 형태인 단수 조건 출력 기호가 하드웨어 흐름도에 존재하면 이에 대응하는 SDL-Ⅱ 구문은

조건 판단 기호 이름 (조건식)/(조건식이 참일 때 천이할 기호, 조건식이 거짓일 때 천이할 기호).

이다.

조건식이 여러개이며 각각의 조건식이 참일 때 해당하는 기호로 천이하는 즉, IF 조건식1 THEN GOTO 기호 이름1 IF 조건식2 THEN GOTO 기호 이름2 .... IF 조건식n THEN GOTO 기호 이름n 형태인 복수 조건 판단 기호가 하드웨어 흐름도에 존재하면 이에 대응하는 SDL-Ⅱ 구문은

조건 판단 기호 이름 (조건식1, ..., 조건식n) / (조건식1이 참일 때 천이할 기호, ..., 조건식n이 참일 때 천이할 기호).

이다.

또한 임의의 논리식 또는 논리식 쌍이 가질수 있는 값이 여러가지로 존재하여 각각의 경우에 따라 해당하는 기호로 천이하는 즉, SWITCH 논리식 CASE 경우1 : GOTO 기호 이름1 .... CASE 경우n : GOTO 기호 이름n 형태인 경우 조건 판단 기호가 하드웨어 흐름도에 존재하면 이에 대응하는 SDL-Ⅱ 구문은

조건 판단 기호 이름 (논리식)/(논리식이 가질 수 있는 값의 경우)/(각 경우에 따라 천이할 기호).

이다.

4) 구조(structure) 기호와 SDL-Ⅱ 구문

설계 조건에 대한 분석이 구조적 특성을 표현하는 구조 기호가 주어지면 각 구조 블럭별로 나누어 다음과 같이 기술된다.

블럭 이름(INPUT : 입력변수이름; OUTPUT : 출력변수이름; INOUT : 입, 출력 동시사용변수이름; SIGNAL : signal 변수이름; CLOCK : 클럭이름).

블럭 이름은 design library의 기호 이름을 표시하는 것으로 AND2, OR2, NOT, DFF 등과 같이 지정된 것과 사용자가 정의하는 것이 있으며 변수 이름 등은 선택적으로 기술한다.

5) SIGNAL signal 변수, 6) MEMORY memory(register) 변수, 7) LIBRARY design-library 이름, 8) CLOCK 클럭 이름, 9) RESET 모듈의 시작 상태기호 이름을 선택적으로 기술한다.

7) LIBRARY design-library 이름, 8) CLOCK 클럭이름, 9) RESET 모듈의 시작 상태기호 이름을 선택적으로 기술한다.

입력 변수와 출력 변수는 전체 시스템의 입,출력 변수중 해당 모듈 기술부에 속하는 변수만을 기술한다. Signal 변수는 시스템의 입,출력과는 관계없이 기술하는 모듈에 속하는 매개 변수를 signal 변수로 선언한다. Memory 또는 register 변수는 기술하는 모듈에서 사용하는 기억장치 요소를 선언하며 design library 이름은 모듈 기술부에서 사용하는 design library에 속하는 모듈의 이름을 선언한다. 한편 클럭 이름은 기술하는 모듈의 동기를 위한 동기 클럭을 기술하는데 클럭 이름이 생략된 모듈은 비동기식 모듈이다. 또한 모듈의 시작 상태 이름은 모듈이 동작될때의 초기상태 이름을 지정하는 것이다.

한편 구조적 특성을 기술하는 모듈의 선언부에는 동작 모듈의 선언부에서 MEMORY 선언부와 RESET 선언부를 제외하고 선택적으로 기술한다.

이와 같은 SDL-Ⅱ 구문을 사용하여 동작 특성을 표시하는 모듈은

동작 특성 선언부
SBEGIN
동작 특성 기술부
SEND
ENDSEQSDL

이며 구조 특성을 표시하는 모듈은

구조 특성 선언부
CBEGIN
구조 특성 기술부
CEND
ENDSEQSDL

이다.

Ⅲ. 실　　험

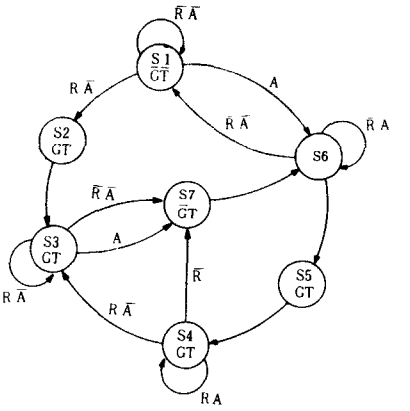
제안한 SAD와 SDL-Ⅱ의 유용성을 보이기 위해 실제의 하드웨어 설계에 적용하면 다음과 같다.

1. 68020 bus arbiter

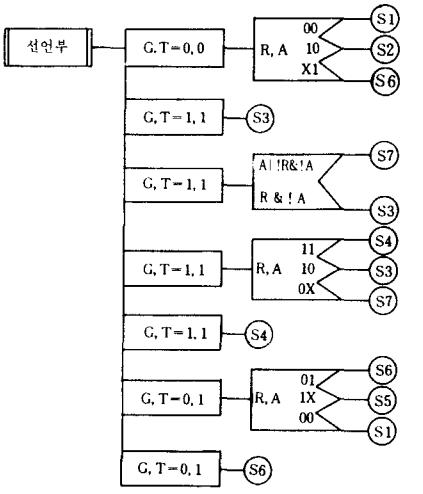
68020 bus arbiter의 그림 1(a)와 같은 상태도[21] 를 SAD로 표시하면 그림 1(b)와 같으며 SDL-Ⅱ로 기술하면 그림 1(c)와 같다.

2. 임의의 상태도

그림 2(a)와 같은 상태도를 SAD로 표시하면 그림 2(b)와 같으며 SDL-Ⅱ로 기술하면 그림 2(c)와 같다.



(a)



(b)

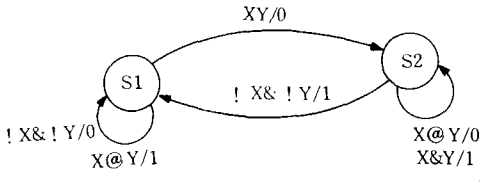
```
SEQSDL bus_Arbiter.
INPUT R; A.
OUTPUT G; T.
CLOCK CLK.
RESET S1.
SBEGIN
  S0 G, T=0, 0;
     →C1.
     C1 (R, A) / (00, 10, x1) / (S1, S2, S6).
  S1 G, T=1, 1;
     →S3.
  S2 G, T=1, 1;
     →C1.
     C1 (A| ! R& ! A, R& ! A) / (S7, S3).
  S3 G, T=1, 1;
     →C1.
     C1 (R, A) / (11, 10, 0x) / (S4, S3, S7).
```

```
    S4 G, T=1, 1;
       →S4.
    S5 G, T=0, 1;
       →C1.
       C1 (R, A) / (01, 1x, 00) / (S6, S5, S1).
    S6 G, T=0, 1;
       →S6.
SEND
ENDSEQSDL
```

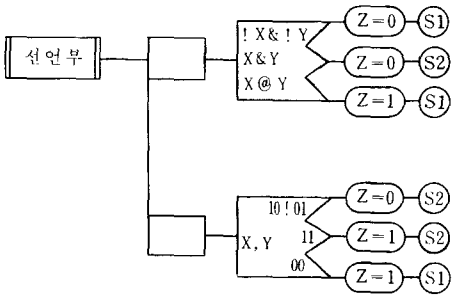
(c)

그림 1. Bus arbiter의 예
(a) 68020 bus arbiter의 상태도
(b) 하드웨어 흐름도
(c) SDL-Ⅱ 기술

Fig. 1. The example of bus arbiter.
(a) State diagram for 68020 bus arbiter.
(b) Hardware flow-chart.
(c) SDL-Ⅱ description.



(a)



(b)

```
SEQSDL Any_State.
INPUT X; Y.
OUTPUT Z.
CLOCK CLK.
RESET S1.
SBEGIN
  S1 →C1.
     C1(! X& ! Y, X&Y, X@Y) / (01, 02, 03).
     01 Z=0;
        →S1.
     02 Z=0;
        →S2.
     03 Z=1;
        →S1.
  S2 →C1.
     C1 (X, Y) / (10!01, 11, 00)/(01, 02, 03).
     01 Z=0;
        →S2.
     02 Z=1;
        →S2.
     03 Z=1;
        →S1.
SEND
ENDSEQSDL
```

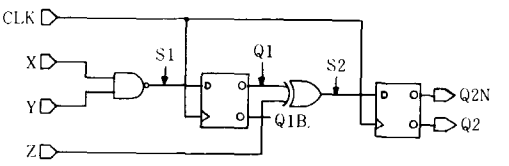
(c)

그림 2. 임의의 상태도의 예
(a) 임의의 상태도
(b) 하드웨어 흐름도
(c) SDL-Ⅱ 기술

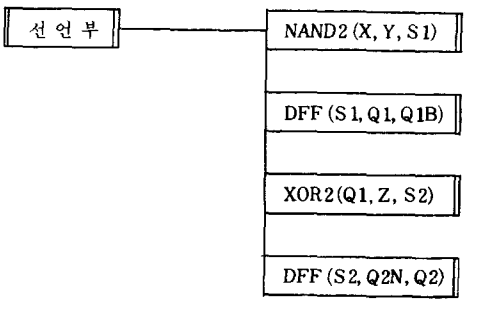
Fig. 2. The example of a state diagram.
(a) A state diagram.
(b) Hardware flow-chart.
(c) SDL-Ⅱ description.

3. 임의의 논리회로도

그림 3(a)와 같은 논리회로도를 하나의 모듈로 구성하여 SDL-Ⅱ로 기술하면 3(b)와 같다.



(a)



(b)

```
SEQSDL Circuit_Sch.
INPUT X; Y; Z.
OUTPUT Q2.
SIGNAL S1; Q1; Q1B; S2; Q2N.
LIBRARY NAND2; DFF; XOR2.
CLOCK CLK.
CBEGIN
NAND2(INPUT:X, Y; OUTPUT:S1).
DFF(INPUT:S1; OUTPUT:Q1, Q1B; CLOCK:CLK).
XOR2(INPUT:Q1, Z; OUTPUT:S2).
DFF(INPUT:S2; OUTPUT:Q2N, Q2; CLOCK:CLK).
CEND
ENDSEQSDL
```

(c)

그림 3. 임의의 논리회로도의 예
(a) 임의의 논리회로도
(b) 하드웨어 흐름도
(c) SDL-Ⅱ 기술

Fig. 3. The example of a logic diagram.
(a) A logic diagram.
(b) Hardware flow-chart.
(c) SDL-Ⅱ description.

4. 가상 ALU의 설계

다음과 같은 설계 요구 조건을 갖는 시스템은 그림 4 의 SDL-Ⅱ로 표시된다.

(설계요구조건) 입력값A와 B가 4-bit로 주어지고 외부에서 입력되는 선택단자 S에 따라 S=1이면 A와 B에 대하여 곱셈을 행한 후 그 결과를 8-bit wire Z에 출력하고 S=0이면 A를 B로 나누어 몫을 Z의 상위 4-bit에, 나머지를 Z의 하위 4-bit에 각각 출력하는 시스템을 설계하고자 한다.

위와 같은 곱셈기와 나눗셈기가 library에 모두 주어지지 아니하였다 가정할 때 곱셈기는 [23]의 AHPL 기술을 근거로 하고 나눗셈기는 [11]의 slow-division 알고리듬을 이용하여 계층적 설계방식을 이용하여 설계한다. 그림 4(a)는 상위 레벨 모듈이며 그림 4(b)는 (a)의 기술에서 사용되는 모듈로 library에 존재하지 않는 곱셈기와 나눗셈기에 대한 하위 레벨 모듈이다.

```
SEQSDL EXAMPLE.
INPUT A[4]; B[4]; S.
OUTPUT Z[8].
CLOCK CLK.
RESET S1.
SBEGIN
  S1→C1.
    C1 (S) / (O1, O2).
    O1 Z=MULTI(A ? B)[0:7];
       →S2.
    O2 Z[0:3], Z[4:7]=DIV(A ? B)[0:7];
       →S2.
  S2 STOP.
SEND
ENDSEQSDL
```

(a)

```
SEQSDL MULTI.
INPUT A[4]; B[4].
OUTPUT Z[8].
MEMORY AC1[4]; AC2[4]; COUNT[2]; EXTRA[5].
LIBRARY F4[5]; INC[3].
CLOCK CLK.
SBEGIN
  S1 AC1, AC2←A, B;
     EXTRA←5#0;
     →S2.
  S2→C1.
     C1 (AC1[3]) / (S3, S4).
  S3 EXTRA←FA4(EXTRA[1:4]+AC2);
     →S4.
  S4 EXTRA, AC1←0, EXTRA, AC1[0:2];
     COUNT←INC(COUNT+1)[1:2];
     →C1.
     C1 (*＼COUNT) / (S5, S2).
  S5 Z=EXTRA[1:4], AC1.
SEND
ENDSEQSDL
```

```
SEQSDL DIV.
INPUT A[4]; B[4].
OUTPUT Z[8].
MEMORY Q[4]; R[4]; C[4].
LIBRARY INC[5]; FS4[5]; MAG[3].
CLOCK CLK.
SBEGIN
  S1 C←B;
     →S2.
  S2→C1.
     C1 (MAG(D>B)[0]) / (O1, O2).
     O1 C←C[1:3], 0;
        →S2.
     O2 R←A;
        Q←4#0;
        →S3.
  S3→C1.
     C1 (MAG(C>B)[0]) / (O1, S5).
     O1 Q←Q[1:3], 0;
        C←0, C[1:3];
        →S4.
  S4→C1.
```

```
    C1 (MAG{R>=D}[0:1]) / (O1, S3).
    O1 Q←FA4{Q+1}[1:4];
       R←FS4{R-C}[1:4];
       →S3.
  S5 STOP.
SEND
ENDSEQSDL
```

(b)

그림 4. 가상 ALU의 SDL-Ⅱ 기술
(a) 상위 레벨 모듈 기술부
(b) (a)를 위한 하위 레벨 모듈 기술부

Fig. 4. SDL-Ⅱ description of virtual ALU.
(a) High-level module description.
(b) Low-level module description for (a).

Ⅳ. 결 론

본 논문에서는 논리 설계 자동화를 위한 레지스터 전송 레벨의 하드웨어 흐름도인 SAD와 하드웨어 기술 언어인 SDL-Ⅱ를 각각 제안하였다.

설계 요구 조건을 레지스터 전송 레벨에서 동작 및 구조적 특성을 함께 분석하여 ASM 설계방법을 이용하면서 일반화된 FSM을 계층적으로 표현하기 위한 하드웨어 흐름도 SAD를 제안하였다.

SDL-Ⅱ는 시스템을 분석하여 design library를 이용하면서 새로이 설계되거나 design library의 정보를 수정하여 사용하고자 하여 기술되는 모듈 기술부와 2개 이상의 독립적인 하위 레벨의 모듈이 존재하면 이들을 하나의 모듈로 구현하기 위하여 기술되는 상위 레벨의 모듈 기술부로 구성되었다. 각 모듈 기술부는 제안한 하드웨어 흐름도의 각 기호에 1대 1대응하는 구문 구조를 갖도록 구문을 설정하였다. 또한 설계 요구 조건을 여러가지로 표현하고 SDL-Ⅱ로 기술함으로써 본 논문의 유효성을 확인하였다.

한편 앞으로의 연구과제는 SDL-Ⅱ의 시뮬레이터와 게이트 레벨의 논리회로도를 생성하는 [17-19]와 같은 SDL-Ⅱ 하드웨어 컴파일러 및 EPLD 시스템[21]과 같은 시스템을 구축하는 것이다.

부 록: : SDL-Ⅱ의 BNF (Backus Naur Form)

```
<sdl_program>: : =<sdl_module> {<sdl_module>}
<sdl_module>: : =<behavior_module> ¦ <structure_module>
<behavior_module>: : =<beh_decl_body> <behavior_body>
<structure_module>: : =<str_decl_body> <structure_body>
<beh_decl_body>: : =<module_name>[<input_name>][<output_name>][<inout_name>]
        [<memory_name>][<library_name>][<signal_name>][<clock_name>][<reset_name>]
<str_decl_body>: : =<module_name>[<input_name>]|[<output_name>][<inout_name>]
                [<library_name>][<signal_name>][<clock_name>]
<module_name>: : = 'SEQSDL' <identifier> '.'
<input_name>: : = 'INPUT' <varible> {';' <variable>} '.'
<output_name>: : = 'OUTPUT' <variable> {';' <variable>} '.'
<inout_name>: : = 'INOUTPUT' <variable> {';' <variable>} '.'
<memory_name>: : = 'MEMORY' <variable> {';' <variable>} '.'
<library_name>: : = 'LIBRARY' <variable> {';' <variable>} '.'
<signal_name>: : = 'SIGNAL' <variable> {'.' <variable>} '.'
<clock_name>: : = 'CLOCK' <identifier> '.'
<reset_name>: : = 'RESET' <state_symbol_name> '.'
<behavior_body>: : = 'SBEGIN' <state_block> {<state_block>} 'SEND' 'ENDSEQSDL'
```

<structure_body> : : = 'CBEGIN' <structure_block> {<structure_block>} 'CEND' 'ENDSEQSDL'
<state_block> : : = <state_box> {<cond_conout_box>}
<state_box> : : = <state_symbol_name> ({<output>} '→' <sdl_symbol_name> '.' | 'STOP.')
<cond_conout_box> : : = <cond_box> | <con_out_box>
<cond_box> : : = <cond_symbol_name> (<single> | <multi> | <case>)
<multi> : : = '(' <expression_list> ') / ('<sbl_symbol_name_list>').'
<case> : : = '(' <case_cond> ') / ('<case_value_list> ') / ('<sdl_symbol_name_list> ').'
<con_out_box> : : = <conout_symbol_name> <output> {<output>} '→' <sdl_symbol_name> '.'
<sdl_symbol_name> : : = <state_symbol_name> | <cond_symbol_name> | <conout_symbol_name>
<sdl_symbol_name_list> : : = <sdl_symbol_name> ',' <sdl_symbol_name> {',' <sdl_symbol_name>}
<state_symbol_name> : : = 'S' <number> {<digit>}
<cond_symbol_name> : : = 'C' <number> {<digit>}
<conout_symbol_name> : : = 'O' <number> {<digit>}
<output> : : = <transfer> | <connection>
<transfer> : : = <destination> '←' <source> ';'
<connection> : : = <destination> '=' <source> ';'
<destination> : : = <variable> {',' <variable>}
<source> : : = <source> ',' <expression> | <expression>
<expression> : : = <expression> ('@' | '&' | '¦') <term> | <term>
<term> : : = '!' <factor> | <factor>
<factor> : : = ['*\' | '+\'] <primary1> | <primary2>
<primary1> : : = <variable> | <element>
<primary2> : : = <constant_list> | <library> | <element>
<element> : : = (<expression>)
<expression_list> : : = <expression> ',' <expression> {',' <expression>}
<case_cond> : : = <variable> {',' <variable>}
<case_value_list> : : = <zero_one_list> {',' <zero_one_list>}
<zero_one_list> : : = <zero_one1> {<zero_one1>}
<library> : : = <variable> '{' <variable> {<operator> <variable>} '}' <bit_list>
<operator> : : = '+' | '−' | '*' | '/' | '%' | '?' | '>' | '<' | '=>' | '=<' | '<>' | '=='
<constant_list> : : = <zero_one> | <number> '#' <zero_one>
<zero_one> : : = '0' | '1'
<zero_one1> : : = <zero_one> | 'x'
<variable> : : = <identifier> | <identifier> <bit_list>
<bit_list> : : = '[' <digit> ']' | '[' <digit> ':' <number> ']'
<alpha_numeric_symbol> : : = <alphabet> | <digit> | '−'
<identifier> : : = <alphabet> {<alpha_numeric_symbol>}
<alphabet> : : = 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h' | 'i' | 'j' | 'k' | 'l' | 'm' | 'n' | 'o' | 'p'
| 'q' | 'r' | 's' | 't' | 'u' | 'v' | 'w' | 'x' | 'y' | 'z' | 'A' | 'B' | 'C' | 'D' | 'E' | 'F'
| 'G' | 'H' | 'I' | 'J' | 'K' | 'L' | 'M' | 'N' | 'O' | 'P' | 'Q' | 'R' | 'S' | 'T' | 'U'
| 'V' | 'W' | 'X' | 'Y' | 'Z'
<digit> : : = '0' | <number>
<number> : : = '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
<structure_block> : : = <structure_name> '(' <structure_dec> {';' <structure_dec>} ').'

<structure_dec>::=[<structure_input_name>] [<structure_output_name>]
[<structure_inout_name>] [<structure_clock_name>]
<structure_input_name>::='INPUT:'<variable> {','<variable>}
<structure_output_name>::='OUTPUT:'<variable> {','<variable>}
<structure_inout_name>::='INOUTPUT:'<variable> {','<variable>}
<structure_clock_name>::='CLOCK:'<identifier>
<structure_name>::=<identifier>

參 考 文 獻

[1] D.D. Gajski, R.H. Kuhn, "New VLSI tools," Computer, vol. 16, no. 12, Dec. 1983.

[2] D.D. Gajski et al, "Silicon compilation," NATO study Institute on Logic Synthesis and Silicon compilation for VLSI, 1986.

[3] A.C. Parker, "Automated synthesis of digital systems," *IEEE Trans. Design and Test*, vol. 1, no. 4, pp. 75-81, Nov. 1984.

[4] A.V. Goldberg et al, "Approachs toward silicon compilation," *IEEE Circuits and Devices Magazine*, vol. 1, no. 3, pp. 29-39, May 1985.

[5] P.G. Paulin et al, "HAL: A multi-paradigm approach to automatic data path synthesis," *Proc. of the 23rd DAC*, 1986, pp. 263-270.

[6] D.E. Thomas et al, "Automatic data path synthesis," *IEEE Computer, December* 1983, pp. 59-70.

[7] A.C. Parker et al, "MAHA: A program for data path synthesis," *Proc. of the 23rd DAC*, 1986, pp. 461-466.

[8] P. Marwedel, "The MIMOLA design system: tools for the design of digital processors," *Proc. of the 21st DAC*, 1984, pp. 587-593.

[9] C.S. John et al, "Silicon compilation based on a data-flow paradigm," *IEEE Circuits and Devices Magazine*, vol. 1, no. 3, May 1985, pp. 21-28.

[10] R. Camposano et al, "Combined synthesis of control logic and data path," *Proc. of the ICCAD'87*, 1987, pp. 327-329.

[11] R. Jamier et al, "APPLON, a data path silicon compiler," *IEEE Circuits and Devices Magazine*, vol. 1, no. 3, May 1985, pp. 6-14.

[12] Thomas Payne, Robert Wells and Werner Gundel, "A study of automatic placement strategics for very large gate array designs," *Proc. of the ICCAD-87*, pp. 194-197, 1987.

[13] 이건배, 정정화, "게이트 어레이의 자동배치, 배선 시스템," 전자공학회논문지, 제25권 제5호, pp. 100-107, 1988.

[14] 엄낙웅, 이철동, 유영욱, "표준셀배치 배선툴의 개발," 과학기술처 '87특정연구결과 발표회 논문집, pp. 103-105, 1988.

[15] O. Karatsu et al, "An integrated design automation system for VLSI circuits," *IEEE Design and Test*, vol. 2, no. 5, pp. 15-26, 1985.

[16] S.G. Shiva, "Computer hardware description language-tutorial," *Proc. of the IEEE*, vol. 67, no. 12, pp. 1605-1615, Dec. 1979.

[17] Manzer Masud, Sadiq Sait M., "Universal AHPL-A language for VLSI design aumation," *IEEE Circuits and Devices Magazine*, 2., no. 5, Sep. 1986.

[18] F.J. Hill and G.R. Peterson, Digital systems: Hardware Organization and Design, New York: Wiley, 1978.

[19] 조중휘, "VLSI의 논리 설계 자동화를 위한 Symbolic Description Language에 관한 연구," 한양대학교 박사학위 논문, 1986.

[20] C.R. Clare, Designing Logic System Using State Machines, (McGraw-Hill 1973).

[21] Altera Corp., Applications Handbook, 1987.

[22] R. Rudell et al, "A finite-state machine synthesis system," *Proc. ISCAS 85*, pp. 647-650, 1985.

[23] 한국전자통신연구소, 설계 자동화 시스템 개발에 관한 연구, 과학기술처, pp. 377-379. *

著 者 紹 介

趙 仲 彙 (正會員)

1957年生. 1981年 한양대학교 전자공학과 졸업. 1986年 8月 한양대학교 전자공학과 박사학위 취득. 1986年 9日～현재 인천대학교 전자공학과 조교수. 주관심분야는 VLSI CAD 특히 Silicon compiler 및 HDL임.

●

鄭 正 和 (正會員) 第26卷 第1號 參照
현재 한양대학교 전자공학과 부교수

李 丙 鎬 (正會員)

1952年 10月 18日生. 1975年 한양대학교 전자공학과 졸업. 1977年 한양대학교 대학원 전자공학과 졸업 석사학위 취득. 1980年～ 1981年 한국전자통신연구소 연구원. 1981年 9月～현재 한양대학교 전자공학과 부교수. 주관심분야는 VLSI CAD, 알고리듬, Computer 악키텍춰 등임.