論文95-32A-8-19

DAMUL : ASIC 설계용 상위레벨 합성기

(DAMUL : High-level synthesizer for ASIC design)

金 基 鉉 *, 鄭 正 和 *

(Ki-Hyun Kim, and Jong-Wha Chong)

요 약

본 논문에서는 ASIC 라이브러리나 FPGA를 이용하여 회로 합성을 수행하기 위한 상위 레벨 합성기 DAMUL을 제안한다. DAMUL은 셀 라이브러리의 특정 셀에 대한 VHDL기술 형식을 정의하여 합성전에 특정 셀에 대한 할당을 수행한다. 연결구조는 멀티플렉서를 지원하며, 셀라이브러리에 존재하는 연결 구조 수의 최소화를 할당 목적으로 한다. 또한 IC의 상태(status)및 제어(control)뿐만 아니라 IC 동작에 필요한 레지스터를 구현 하고자 전역 변수에 전용 레지스터를 할당한다. 벤치마크 회로에 대한 합성 결과를 통하여 제안된 시스템의 효용성을 보인다.

Abstract

This paper presents a new high-level synthesizer for ASIC designs using ASIC library or FPGAs. DAMUL defines the VHDL description for a specified hardware and allocate some VHDL codes, which describe the behavioral specification, to the corresponding hardware before the synthesis. The interconnections are implemented by the multiplexers, and the objective of allocation is the minimization of the number of multiplexers. Also, the dedicated registers is used for global variables, in order to implement the other necessary registers as well as status and control registers. The effectiveness of the proposed system is shown by the synthesis results of benchmark circuits.

I. 서 론

상위 레벨 합성에 대한 연구는 1970년대 말 부터 시작되었으며 특히 1980년대 중반부터 알고리듬 레벨에서의 기술로 부터 자동적인 레지스터 전송 레벨의 회로 합성을 하는 상위 레벨 합성이 CAD 분야의 중요 연구 분야로 대두되면서 다양한 아키텍춰, 목적함수등을 만족하는 많은 시스템들이 연구.발표되었다. 특히 DSP에 적용하기 위한 상위레벨 합성기는 SEWHA[1]에서 최초로 파이프라인 개념을 도입한 후로 멀티 포트 메모리[2], retiming[3], conditional resource sharing[4] 등과 같은 다양한 개념의 합성기가 개발되었으며, 특히 특정 형태의 아키텍춰를 설정하여 제한된 응용 범위의 합성을 지원하는 DSP 전용 상위 레벨 합성기인 CATHEDRAL-III는 실제 설계에 응용될 수 있는 유일한 것으로 평가되고 있다.[5] 또한 프로세서 개발에 적용되는 상위 레벨 합성기를 개발하기 위하여 산업체에서 많은 노력을 기울이고 있다.[6-7] ASIC 설계를 위한 상위 레벨 합성기에 대해서도 많은 연구가 수행되었다. Chippe[8]는 게이트 어레이를 타겟칩으로 하여 하드웨어 수 및 동작 시간등을 반복적으

* 漢陽大學校 電子工學科

(Dept. of Elec. Eng., Hanyang Univ.)

接受日字:1995年1月17日, 수정완료일:1995年8月4日

로 조절하면서 합성을 수행하는 최초의 지능형 시스템이다. Hebe[9]는 하드웨어의 할당을 수행하면서 하드웨어 자원 공유 문제가 발생하면 순차적인 동작에 의한 시간 제한을 부여하고 각 연산간의 최대/최소 시간 제한에 의해 상대적인 동작 시간뿐만 아니라 입력 신호에 대한 상대적인 동작 시간을 결정하기 때문에 ASIC 설계 지원용으로 개발되었으나 조건 분기등의 처리가 미흡하다. HIS[10]의 경우, 제어 흐름 그래프의 조건 분기에 따라 경로를 설정하고 부여된 제한 조건에 의한 상태수의 최소를 목적으로하여 스케쥴링을 수행한다. 초기 할당과 coloring방법을 이용한 할당 최적화를 하여 하드웨어의 할당을 수행한다. 그러나 상대적인 동작 시간 제한에 대한 처리가 미흡하다. [12]에서 SAW[11]를 이용하여 설계를 한 결과, 카운터나 타이머 등이 포함되어 있는 회로를 설계할 때 설계자에 의한 설계와 합성기에 의한 설계는 많은 차이를 보이므로 동작 기술과 구조 기술이 혼용된 형태를 지원할 수 있어야 한다고 하였다. 그러나 현재까지 개발된 합성기의 경우 설계자에 의한 회로 설계와 같은 형태를 지원하지 못하고 있다.

따라서 본 논문에서는 [12]에서 기술한 구조 기술의 지원 대신 일정한 형태의 VHDL 기술에 대해 특정 하드웨어를 매핑함으로서 ASIC 라이브러리 사용 및 FPGA로의 구현을 지원하고자 한다. 또한 마이크로프로세서와의 인터페이스를 통하여 칩의 동작을 제어받는 경우 command 및 status 레지스터 같은 레지스터(이하 "전역 레지스터"라 함)군이 존재하게 된다. 이와 같은 레지스터는 대부분 특정한 의미가 부여되기 때문에 기존의 상위 레벨 합성기에서와 같이 변수들을 공유할 수 없는 레지스터 들이다. 따라서 이러한 레지스터들을 지원하기 위하여 2개 이상의 프로세스에서 읽기/쓰기를 행하는 변수에 대해서는 레지스터 공유를 하지 않고 별도의 레지스터를 할당한다.

본 논문의 각 부분에 대한 설명은 다음과 같다. 3장에서는 본 논문에서 설정한 타겟 아키텍춰에 대해 설명하고 4장에서는 본 논문에서 사용하는 중간 형태인 HSFG 및 선행 할당 과정에 대해 기술한다. 5장에서는 비동기 회로에 대해 설명하고 6장에서는 간략하게 스케쥴링에 대해 설명한다. 7장에서는 clustering 방법에 의한 할당 알고리듬에 대해 설명하고 8장에는 벤치마크에 대한 실험 결과에 대해 논한다.

Ⅱ. 전체 시스템 구성

본 논문에서 제안하는 전체 시스템 구성은 그림 1과 같다. VHDL로 기술된 입력에 대해 상위 레벨 합성을 위한 중간 형태인 HSFG(Hanyang Sequential Flow Graph)[13]를 생성한 후 카운터, 멀티플렉서, 시프트 레지스터등 ASIC 설계에 기본적으로 많이 이용되는 하드웨어에 대한 선행할당을 한다. HSFG 에는 선행할당(pre-allocation)의 결과도 표현할 수 있다. 스케쥴링은 다양한 형태의 제한 조건을 그래프로 표현한 후 그래프의 포함 및 중첩 관계에 의해 제한 조건의 수를 최소로 하여 전체적인 동작 시간을 최소화한다. 할당(allocation)은 clustering 방법을 이용하여 멀티플렉서의 수가 최소가 되도록 연산자와 메모리의 할당을 수행한다.

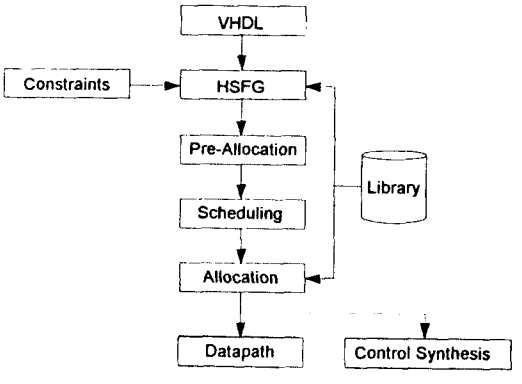


그림 1. 시스템 구성
Fig. 1. The system configuration.

Ⅲ. Target 아키텍춰

ASIC vendor의 셀라이브러리는 입력 수가 고정된 멀티플렉서를 지원하며 또한 mux-based FPGA도 입력 수가 고정된 멀티플렉서에 의해 연결 구조를 구성한다. 따라서 DAMUL은 고정된 입력의 멀티플렉서를 최소한으로 사용하는 것을 사용을 목적으로 하며, target 아키텍춰는 그림 2와 같은 구조의 멀티플렉서를 이용하여 연결 구조를 구현하였고 각 프로세스별로 합성이 수행된다. 프로세스간에 공유하는 변수(전역 변수)는 전역 레지스터(GLB Reg)에 할당하고 하나의 프로세스에서만 사용하는 국부 변수는 그림 2의 REG 1과 같이 각 프로세스내에 있는 레지스터에 할당된다.

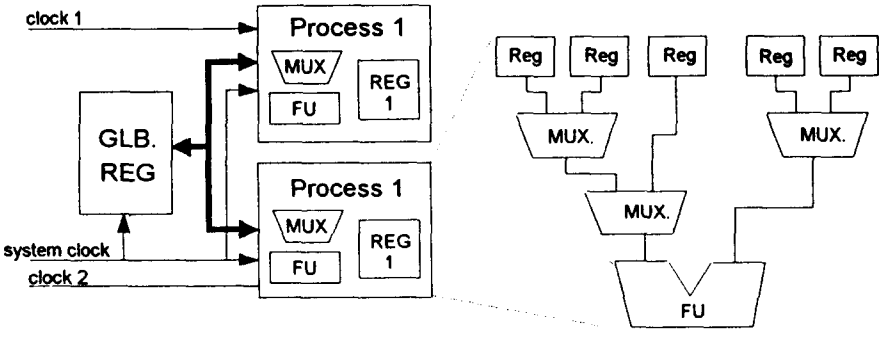


그림 2. Target 아키텍춰
Fig. 2. The target architecture.

각 프로세스는 시스템 클럭 또는 별도의 클럭에 동기되며 전역 레지스터의 경우는 시스템 클럭에 의해 구동된다.

Ⅳ. 선행 할당(pre-allocation)

HSFG[13]는 제어 흐름 그래프(control flow graph)의 한 표현 형태이다. HSFG는 ASIC 설계에 적합한 아키텍춰를 지원하기 위하여 카운터, 시프트 레지스터, 디코더 및 멀티플렉서등 ASIC 설계시 많이 사용하는 하드웨어를 노드로 갖을 수 있다. HSFG의 노드로 표현될 하드웨어는 VHDL로 기술된 입력이 HSFG로 변환된 후, 기 정의된 패턴을 탐색함으로서 구하게 된다. 따라서 제한된 형태이지만 VHDL의 기술 방식 및 HSFG의 표현 형태를 미리 결정하면 쉽게 하드웨어로 변환할 수 있다. VHDL의 기술문에 대한 하드웨어 변환 과정은 그림 3과 같다.

```
변수값 설정
while(변수값 비교) loop
  .....
  변수값 가감
  wait until(클럭 정의)
end loop
```

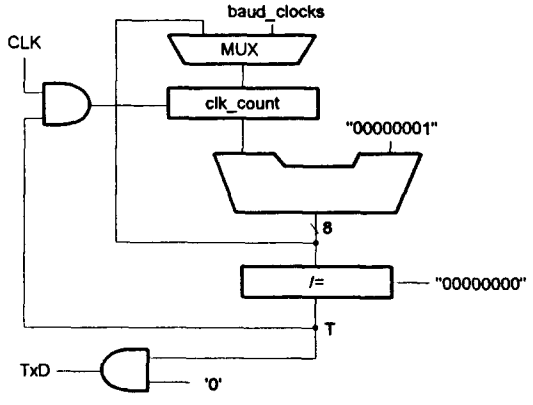
(a) VHDL 기술 (b) HSFG

그림 3. 카운터의 VHDL 기술 및 HSFG
Fig. 3. VHDL description of counter and corresponding HSFG.
(a) VHDL description
(b) corresponding HSFG

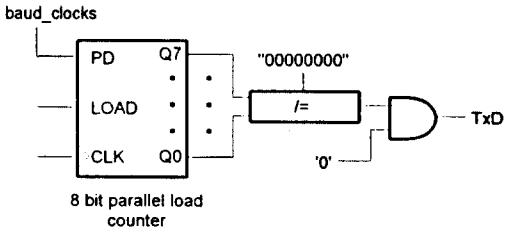
그림 3은 counter로 대체될 수 있는 VHDL 기술 형태 및 HSFG를 보여주고 있다. 따라서 기술된 동작 사양으로 부터 생성된 HSFG에 그림 3 (b)와 같은 HSFG가 있는 지를 찾아서 대응하는 카운터로 대체하여야 한다. 선행 할당의 필요성 및 과정을 설명하기 위하여 그림 4를 살펴보자.

```
clk_count := baud_clocks;
while( clk_count /= "00000000") loop
   TxD <= '0';
   wait until (CLK = '0') and (not CLK'stable);
   clk_count = clk_count - "00000001";
end loop;
Tx <= 1;
```

그림 4. 4 bit 카운터의 예
Fig. 4. An example of a 4 bit counter.

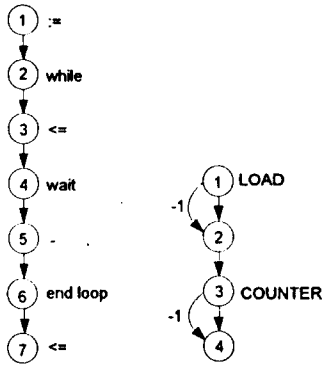


(a) 패턴 매칭 전



(b) 패턴 매칭 후

그림 5. 4 bit 카운터에 대한 회로 합성 결과
Fig. 5. The synthesis result on 4 bit counter.
(a) Before pattern matching
(b) After pattern matching



(a) 패턴 매칭 전의 HSFG (b) 패턴 매칭 후의 HSFG

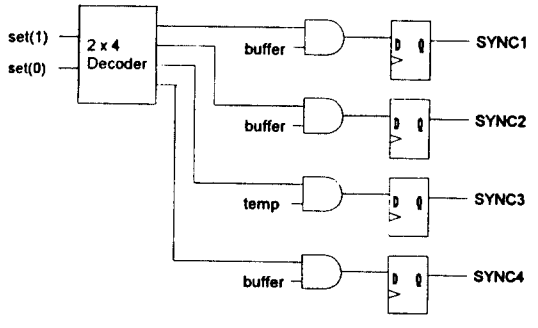
그림 6. 4 bit 카운터의 패턴 매칭
Fig. 6. A pattern matching of 4 bit counter.
(a) HSFG before a pattern matching
(b) HSFG after a pattern matching

그림 5는 그림 4의 예에 대한 카운터의 패턴 매칭 전과 후의 결과에 대한 회로를 보여준다. 그림 5에서 패턴 매칭에 의한 회로 크기는 1개의 카운터와 비교기만 필요한 반면, 패턴 매칭을 하지 않은 경우에는 1개의 감산기, 레지스터, 멀티플렉서 및 비교기가 요구된다. 또한 그림 5 (a)보다는 그림 5(b)에 의한 방법에 의해 실제 회로가 설계된다. 따라서 실제적인 하드웨어 설계 및 비용감소의 면을 고려하면 패턴 매칭이 필요하다.

그림 5 (b)와 같은 회로를 합성하기 위하여 패턴 매칭을 수행하는 과정은 다음과 같다. 그림 4의 VHDL 기술은 그림 6(a)의 HSFG로 변환된다. 그림 6(a)에서 카운터의 매칭은 먼저 반복문이 시작되는 노드(노드 2) 를 찾는다. 반복문 시작 노드가 갖는 조건문을 분석한 후, 반복문 시작(예 while)과 끝나는 부분(예 end loop)의 사이에서 조건문에 나타나는 변수(그림 4의 "clk_count")의 가감을 수행하는 노드(노드 5)가 있는 지를 찾는다. 변수의 가감 수행하는 노드가 존재하면 그림 6(b)의 HSFG로 변환한다. (그림 6(b)에서 에지에 부여되는 값 -1은 노드 1(3) 과 노드 2(4)가 동시에 동작할 수 없고 최소한 1이상의 제어 스텝 차이가 필요하다는 것을 의미한다.) 이때 만일 그림 6과 같이 반복문을 카운터로 대체할 경우 카운터는 초기화와 구동이 동시에 수행되지 못하기 때문에 데이타의 loading과 카운터의 동작으로 분리하여야 한다. 따라서 그림 6(b)에서 카운터를 나타내는 노드 3에 COUNTER라는 특성을, 노드 1은 카운터의 초기값을 설정하는 부분이 되므로 노드 1은 LOAD라는 특성을 부여한다.

```
case(set(1 downto 0)) is
  when "00" =>
       sync1 <= buffer;
  when "01" =>
       sync2 <= buffer;
  when "10" =>
       sync3 <= temp;
  when "11" =>
       sync4 <= buffer;
end case;
```

(a) CASE문의 예



(b) 하드웨어 구현

그림 7. CASE문의 구현
Fig. 7. The implementation of a CASE statement.
(a) An example of a CASE statement
(b) The hardware implementation

또한 그림 4와 같이 반복문 내에 wait until문에 의해 카운터를 구동하는 클럭을 기술할 수 있으며, wait문에 의해 클럭을 정의하지 않으면 시스템 클럭에 의해 카운터가 구동되는 것으로 간주한다. 카운터에 의해 반복문이 구현될 지라도 일반 반복문의 처리처럼 조건 검색을 하여야 한다. 따라서 그림 6(b)의 노드 1에서 노드 2, 노드 3에서 노드 4로 연결된 가중치를 갖는 방향성 에지는 카운터에 새로운 값이 부여되고 조건을 검색하여야 하기 때문에 필요하다.

Case문의 경우는 멀티플렉서나 디코더로 볼 수 있는 데 그림 7에서 보듯이 when 다음의 조건 후에 나타나는 실행문이 같은 변수나 신호에 대한 데이타 천이가 아닌 경우 디코더로 대응할 수 있다.

V. 비동기 회로 지원

본 논문에서 극히 제한적으로 비동기 회로를 지원하며 사용자에 의해 선택적으로 적용된다. 클럭에 비동기적으로 동작할 수 있는 회로는 status 레지스터를 예로 들 수 있다. 즉, status 레지스터는 설계자가 설정한 상태가 발생하면 해당 비트가 '0' 또는 '1'이 되어 합성하고자 하는 회로의 현재 상태를 나타낸다. 따라서 그림 8과 같이 status 레지스터는 레지스터의 상태를 나타내는 조건들에 의해 구동될 수 있다.

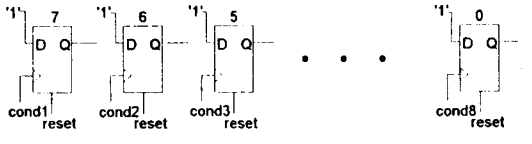
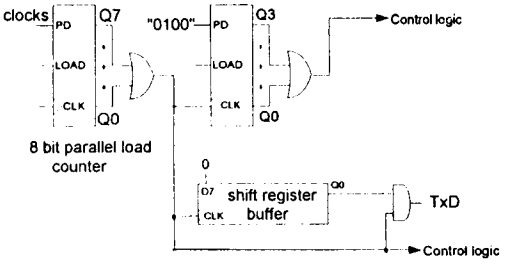


그림 8. 상태 레지스터의 예
Fig. 8. A status register.

또 다른 비동기 회로는 그림 9와 같이 이중 반복문이 카운터로 대체되는 경우이다. 그림 9 (a)와 같이 기술된 문장은 그림 9(b)와 같이 안쪽에 있는 반복문에 매핑되는 카운터의 출력 결과가 바깥쪽에 있는 카운터를 구동하는 것으로 간주한다. 이 경우 그림 9 (b)에서 카운터 2가 클럭에 동기되어 동작하지 않고 카운터 1의 출력 결과에 따라 게이트 지연 후 동작한다. 따라서 이중 반복문의 경우 카운터 2와 같이 바깥쪽 반복문에 대한 카운터는 비동기적으로 동작한다.

```
temp := "0100";
while ( temp /= "0000") loop
  temp := temp - "0001";
  count := clocks;
  while ( count /= "00000000") loop
    TxD <= buffer(0);
    count := count - "00000001";
  end loop;
  buffer := '0' & buffer(7 downto 1);
end loop;
```

(a) 이중 반복문의 예



(b) 하드웨어 구현

그림 9. 이중 반복문의 구현
Fig. 9. The implementation of a nested loop.
(a) An example of a nested loop
(b) The hardware implementation

VI. 스케줄링

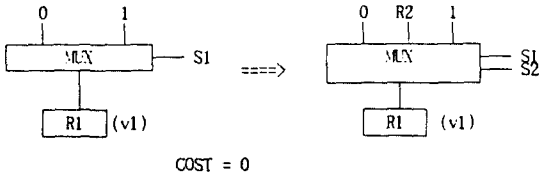
Control-dominated ASIC의 경우 조건분기 및 시간제한등에 의해 동작이 기술된다. 따라서 이러한 조건분기 및 시간제한등의 처리가 중요하다. 본 논문에서 적용한 스케줄링 알고리듬은 제한 조건을 그래프로 표현한 후, 그래프간에 공유하는 노드가 존재하면 공유 노드들로 구성된 새로운 그래프로 중첩되는 그래프를 대체한다. 각 그래프간의 공유 노드가 존재하지 않을 때까지 위 과정을 반복 수행한다. 공유 노드가 없는 그래프들에 대해 각 그래프 내의 노드들을 2 개의 시간 영역에 할당함으로서 스케줄링을 수행한다. 스케줄링 과정은 [13] 에서 자세히 기술하였다.

VII. 연결 구조 최소화를 위한 할당 알고리듬

스케줄링이 끝난 후 연산과 변수들에 대해 할당을

수행한다. 할당을 위해 Clustering 방법과 CG(compatiblity graph)를 이용한 알고리듬을 제안한다. CG의 노드는 변수를 나타내며 에지는 변수의 레지스터 공유를 나타낸다. 할당시 고려할 점은 FPGA나 고정된 입력을 갖는 ASIC 라이브러리를 지원하기 때문에 목적함수가 연결구조의 최소화라는 점이다. 예를 들면 그림 10 (a)와 같이 레지스터 R1의 입력 수가 2이고 입력 선택을 위해 4:1 멀티플렉서가 할당되어 있다고 가정하자. 변수 V1이 R1에 새로이 할당될 경우 V1의 입력이 되는 V3가 레지스터 R2에 할당되어 있다면 그림 10 (b)와 같이 입력 수가 3으로 증가한다. 입력 수가 증가하더라도 입력연결구조를 위해 멀티플렉서의 증가는 이루어 지지 않는다. 따라서 변수 V1이 레지스터 R1에 할당될 경우의 cost는 0이 되고 연결 신호선 수는 입력 신호와 멀티플렉서 제어신호의 증가에 의해 기할당된 신호선에 비해 2만큼의 증가분이 발생한다. 신호선의 증가분을 포함한 전체 신호선 수를 WIRE_COST로 표현하여 메모리 할당시에 사용한다.

V1 := V3; /* V3이 R2에 할당되어 있는 경우 */



(a) 할당 전 (b) 할당 후

그림 10. Cost 계산 예
Fig. 10. An example of the cost calculation.
(a) Before memory allocation
(b) After memory allocation

위에서 설명한 바와 같이 멀티플렉서의 수를 최소화하기 위해 다음과 같이 할당을 수행한다.

1. 연산자 할당

연산에 대응하는 연산자를 각 연산형별(예, +, *, 카운터등)로 할당한다. 같은 연산형의 연산들에 대해 입.출력 변수의 유사성을 계산하여 2 차원 matrix를 구성한다. 유사성 계산은 [14]의 방법을 적용하여 같은 이름의 입력 변수 와 출력 변수의 수를 찾는다. 할당할 연산자의 수가 K개인 경우, 구성된 matrix에서 다른 연산들과 가장 큰 유사성의 합을 갖는 연산 노드들 중 같은 제어 스텝에 동작하는 연산 노드가 K개 존재하면 그 제어 스텝에 있는 K개 연산 노드를 seed들로 선택하고 각각 집합 CLi(i = 1, ..., K)에 할당한다. 그림 11은 연산 노드들이 속해 있는 3개의 집합 영역을 보여주고 있다. 집합 CL은 연산자 할당이 된 연산들의 집합이며, 집합 CAND는 유사성이 큰 연산자에 할당되기 위해 선택된 연산들의 집합으로서 각 집합 CLi(i=1,...,K)에 속한 연산들과 유사성이 가장 큰 연산 노드들이 선택된다. 따라서 2개 이상의 집합 CLi에서 같은 연산 노드들을 선택할 수 있다. 이러한 연산들은 유사성이 큰 하나의 CLi에 선택될 수 있도록 하고 새로운 연산 노드를 집합 CAND에 포함시킨다. CAND에 포함된 연산 노드들을 각 집합 CLi에 포함시키면서 유사성 matrix를 재구성한다. 모든 연산들이 할당될 때까지 위 과정을 반복한다.

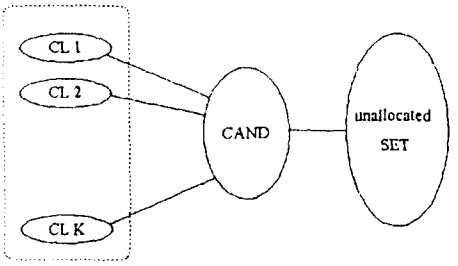


그림 11. 할당되는 연산들의 집합
Fig. 11. The sets for the allocating operations to FUs.

그림 12는 연산자 할당에 대한 예를 보여준다. 그림 12의 최종 결과에서 +2가 ADDER1에 할당되지 못하고 ADDER2에 할당된 것은 같은 시간인 step 2에서 동작하는 +3이 이미 ADDER1에 할당되어 있기 때문이다.

step 1 : v3 := v1 +1 v2;
step 2 : v5 := v3 +2 v4; v2 := v1 +3 v6
step 3 : v3 := v1 +4 v5; v2 := v3 +5 v1

(a) 스케줄링 결과

	1	2	3	4	5	
1	0	0	1	2	1	4
2	0	0	0	0	1	1
3	1	0	0	1	2	4
4	2	0	1	0	1	4
5	1	1	2	1	0	5

(b) 유사성 매트릭스 구성

(c) Seed 선택 (2개의 가산기 사용)

(d) Seed와 유사성이 가장 큰 연산을 선택

	2	4,1	5,3
2	0	0	1
4,1	0	0	0
5,3	1	0	0

(e) 유사성 매트릭스 수정

(f) 최종 결과

그림 12. 연산자 할당 예

Fig. 12. An example for F.U. allocation
- (a) A scheduling result
- (b) .The compatibility matrix construction
- (c) The seeds selection
- (d) The allocation of operations with maximum compatibility value
- (e) The compatibility matrix reconstruction
- (f) The final result

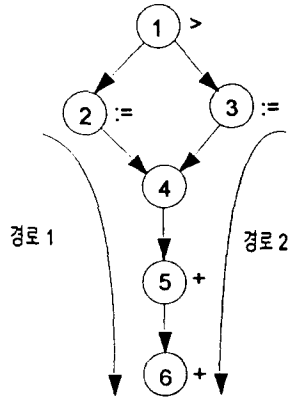
2. 메모리 할당

메모리에 저장될 변수는 크게 전역 변수와 내부 변수로 구분할 수 있다. 전역 변수는 VHDL 기술문에서 하나의 프로세스(process)문에서 사용되지 않고 2개 이상의 프로세스 문에서 읽기/쓰기를 하는 변수이고 내부 변수는 하나의 프로세스문 내에서만 사용되는 변수이다. 전역 변수의 경우 시스템 레지스터로 사용되는 경우가 많고 또한 프로세스별로 합성을 수행하기 때문에 각 프로세스에서 전역 변수의 읽기/쓰기를 예측하기가 어렵다. 따라서 전역 변수는 변수간의 메모리 공유를 하지 않고 별도의 메모리를 할당한다. 또한 시프트 레지스터와 같이 데이타의 입출력과 관련된 레지스터의 경우 별도로 레지스터를 할당한다. 따라서 본 논문에서 제안하는 메모리 할당 알고리듬은 내부 변수(이하 "변수"라 함)에 대해서만 수행한다.

본 논문에서는 같은 변수 명을 갖는 변수들을 같은 레지스터에 할당한다. 그러나 조건 분기등에 의해 같은 변수명을 갖는 변수들이 다른 조건에 나타나기 때문에 처음부터 하나의 변수로 처리할 경우, 레지스터를 공유할 변수를 찾기가 쉽지 않다. 따라서 같은 변수명을 갖더라도 다른 그래프 경로에 나타나는 변수들은 새로운 변수로 분리한다. 분리된 변수들에 의해 그림 13에서 temp가 두 경로에서 나타나므로 경로 1에 있는 변수 temp의 변수 번호는 1, 경로 2에 있는 변수 temp는 변수 번호 2를 부여한다. 변수를 분리한 후 분리된 변수의 life time을 구한다. 각 변수의 life time이 구해지면 1) 같은 변수명, 2) life time의 중첩, 및 3) 동작 조건등을 검색하여 변수명이 같거나, 동작 조건이 중첩되지 않거나, 또는 변수의 life time이 중복되는 기간 중 동작 조건이 다른 변수들은 레지스터 공유가 가능한 변수라 하고 각 변수간의 레지스터 공유 가능 여부를 나타내는 CG를 구성한 후 이를 CVM (Compatible Variable Matrix)로 표현한다. 예를 들면 두 변수 Vi와 Vj가 레지스터 공유가 가능한 경우 $CVM(v_i, v_j) = 1$ 이 된다.

```
if( x > y ) then        (1)
   temp := a;           (2)
else
   temp := b;           (3)
end if                  (4)
a <= x + y;             (5)
b <= x + temp;          (6)
```

(a) VHDL 기술



(b) 변수 분리

그림 13. 같은 변수 명의 변수들 분리

Fig. 13. The seperation of variables with the same name.
- (a) VHDL description
- (b) A variable seperation

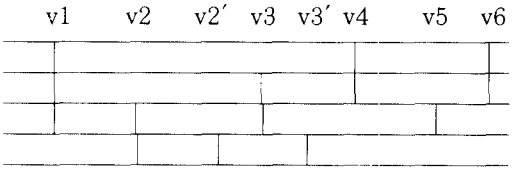
그림 13 (b)에서 변수 1과 2에 대한 CVM(1,2) 는 조건 1)을 만족하므로 1의 값을 갖는다. CVM이 구성된 후 변수명이 같은 변수들을 하나의 레지스터에 할당하기 위하여 변수명이 같은 변수들을 하나의 대표 변수로 처리한다. 그림 13에서 변수 1과 변수 2는 같은 변수 명을 갖으므로 변수 1(또는 2)를 대표 변수로 한다. 대표 변수가 설정되면 CVM을 수정하게 된다. 즉, CVM(V_i, V_k) = CVM(V_i, V_k) & CVM (V_i, V_k) (여기서 k = 1, ..., n)로 한다. 같은 변수 명을 모든 변수에 대해 대표 변수를 설정한 후 레지스터 할당을 수행한다.

K개의 레지스터와 n개의 변수에 대해 ΣCVM (V_i, V_j) 이 가장 작은 변수(V_i)들 중 서로 공유할 수 없는 변수 K개를 seed로 선택한다. 선택된 seed를 레지스터에 할당한 후 레지스터가 갖는 CVM의 행 번호(row index)는 seed의 행번호로 한다. Seed 변수가 할당된 각 레지스터에 대해 다음과 같이 clustering을 수행한다. 각 레지스터에 할당될 가능성이 있는 candidate 변수를 선택한다. CVM(R_i, V_j) = 1 (여기서 i = 1,...,k , j = 1,...,n)인 변수 V_j 에 대해 변수 Vj가 레지스터 Ri에 할당될 경우 cost를 계산한 후 레지스터 Ri에 할당될 변수 중 최소의 cost를 갖는 변수를 레지스터 Ri의 candidate 변수로 한다. 하나의 변수가 2개 이상의 레지스터에 할당될 수 있는 candidate 변수가 되는 경우 가장 작은 할당 cost를 갖는 레지스터의 candidate 변수가 되도록 하고, 새롭게 candidate 변수들을 구성한다. 중복되는 변수가 존재하지 않으면 candidate 변수를 해당되는 레지스터에 할당하고 CVM을 수정한다. 변수 Vj가 레지스터 Ri에 할당되면 CVM(R_i, V_k) = CVM (R_i, V_k) & CVM(V_j, V_k) (여기서 k = 1,...,n)로 CVM을 수정한다. 모든 변수가 할당될 때까지 위 과정을 반복적으로 수행한다. 변수 Vn를 레지스터 Ri에 할당 시 사용하는 cost 함수는 다음과 같다.

$$\text{Cost}(Ri,Vn) = \Delta\text{MUX}(Ri) + \Delta\text{MUX}(\text{data succedents of } Ri) \quad (1)$$

단, ⊿MUX(x) : 레지스터(또는 I/O 포트) x의 입력변화에 따른 MUX의 증가분

그림 9에서 레지스터 R1에 변수 V1이 할당되면 변수 V1에 데이타 종속관계를 갖는 후행 노드들이 할당된 레지스터의 입력단도 변하게 된다. 따라서 ⊿MUX(data succedents of Ri)는 데이타 종속관계에 따른 연결구조 변화를 cost에 반영하기 위한 값이다. 또한 ASIC 설계에서의 데이타 천이는 DSP의 바이트 단위의 천이와는 달리 비트 단위의 조합에 의한 천이가 발생한다. 따라서 cost 계산은 비트 단위 비교로 수행한다. 이상에서 설명한 메모리 할당 알고리듬에 대한 예는 그림 14와 같다.



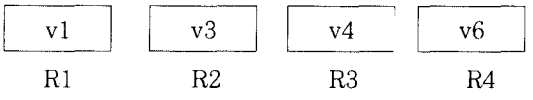
(a) 그림 12의 변수들의 life time

	v1	v2	v2′	v3	v3′	v4	v5	v6
v1	0	0	1	0	1	0	0	0
v2	0	0	1	0	0	1	0	1
v2′	1	1	0	1	0	1	1	1
v3	0	0	1	0	1	0	0	0
v3′	1	0	0	1	0	1	1	1
v4	0	1	1	0	1	0	1	0
v5	0	0	1	0	1	1	0	1
v6	0	1	1	0	1	0	1	0

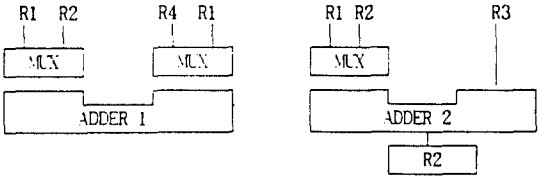
(b) CVM 구성

	v1	v2	v3	v4	v5	v6
v1	0	0	0	0	0	0
v2	0	0	0	1	0	1
v3	0	0	0	0	0	0
v4	0	1	0	0	1	0
v5	0	0	0	1	0	1
v6	0	1	0	0	1	0

(c) 분리된 변수 병합

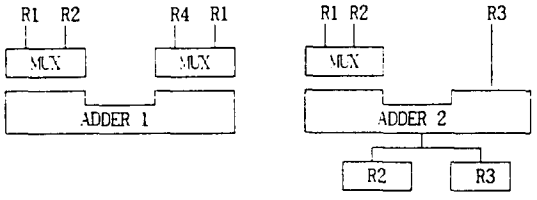


(d) Seed 선택 (4개의 레지스터 이용)



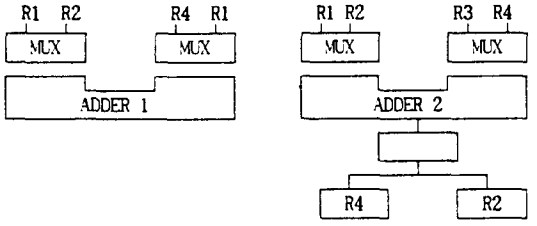
(e) Seed 선택 후의 하드웨어 할당 구조

* v5가 R3에 할당될 경우



COST = 0

* v5가 R4에 할당될 경우



COST = 1 개의 MUX 증가분

(f) 메모리 할당에 따른 하드웨어 연결 구조

그림 14. 메모리 할당의 예

Fig. 14. An examle for memory allocation.
(a) The life time of variables in fig. 12
(b) CVM construction
(c) Merging seperated variables
(d) Seeds selection
(e) The hardware structure after seeds selection
(f) The hardware structure by memory allocation

그림 14에서 조건 분기가 없는 데도 변수를 분리한 이유는 분리된 변수의 처리 과정을 보여주기 위해서이다. 또한 그림 14 (f)에서 변수 v5가 R4에 할당될 경우 연결 구조의 cost는 1개의 입력 증가가 아닌 2x1 멀티플렉서 1개의 증가분에 의해 표현된다. 만일 그림 10과 같은 경우가 발생하면 이 때는 cost는 0이 된다. 따라서 본 논문에서는 입력 수의 증가가 아닌 멀티플렉서 수의 증가를 연결 구조 할당의 cost 함수로 하여 셀 라이브러리에 등록된 멀티플렉서를 활용하고자 한다.

위에서 설명한 메모리 할당 알고리듬을 기술하면 그림 15와 같다.

```
REGISTER_ALLOCATION( ){
  Allocate shift register.
  Divide nodes with same variable by their execution
    condition:
  Construct CVM(Compatible variable matrix) for the
    divided nodes:
  Modify CVM by merging the nodes with same variable:
  Select seed variables by CVM:
  Allocate the seeds to the registers:
  While( all variables are allocated )
    K_WAY_CLUSTER():
}
K_WAY_CLUSTER(){
  Calculate the COST for each variable:
  Determine candidate variables by COST:
  IF(A variable is selected to candidates more than two
      registers)
  Tie break by WIRE_COST:
  Allocate the candidates to the registers:
```

그림 15. 레지스터 할당 알고리듬

Fig. 15. Register allocation algorithm.

Ⅷ. 실험 결과

본 논문에서 제안한 상위 레벨 합성기는 Sparc 1+에서 구현하였으며 벤치마크 회로에 대해 실험을 하였다. 표 1은 본 논문의 실험 결과와 [10]의 실험 결과를 보여준다. [10]은 벤치마크 HLsynth89를 적용하여 실험한 결과이고 본 논문은 벤치마크 HLsynth92를 적용한 결과이다. 본 논문이 HLsynth89를 적용하지 못한 것은 HLsynth89에서는 ISPS 로 기술되어 있는 회로들이 있기 때문이다. 예를 들면 i8251의 경우 [10]은 ISPS로 기술된 예제회로에 대한 합성 결과이고 본 논문은 VHDL로 기술된 예제회로이다. 따라서 1992년에 나온 벤치마크 회로에는 hunter 부가 수신부에 포함되어 기술되어 있지만 ISPS에 의해 기술되어 있는 벤치마크에는 표 1의 [10]의 결과와 같이 수신부와 hunter부가 분리되어 있다. 또한 데이타패스의 width도 [10]은 8인 반면 DAMUL은 1이다. DAMUL의 데이타패스의 width가 1인 이유는 예제회로가 비트 단위의 처리로 기술되어 있기 때문에 [10]에서 보여준 데이타 패스의 width와 다르다. 따라서 본 논문의 결과에 대한 비교 보다는 참고 자료로서 [10]의 결과를 인용·제시하였다. DAMUL의 할당 결과, 카운터에 의해 감산

기가 대체됨으로서 칩 면적을 감소시킬 수 있었다. 레지스터 할당 결과는 2개 이상의 프로세스문에서 사용하는 변수의 경우 1개의 레지스터에 할당하였기 때문에 전반적으로 레지스터의 수가 증가하였다. 따라서 내부 변수를 저장하기 위한 레지스터의 수가 작다. GR로 나타낸 것이 2개 이상의 프로세스에서 사용하는 신호를 시스템 레지스터에 할당한 결과이며 내부 변수를 저장하는 레지스터의 수는 LR로 나타내고 있다. 표 1에서 본 논문의 결과와 [10]의 결과의 차이를 극명하게 보여주고 있는 것이 연결 구조의 형태이다. [10]의 경우 다입력 멀티플렉서를 공유하고 있지만 본 논문의 경우 레지스터 할당 시 비트별 처리를 하면서 멀티플렉서의 입력수가 작은 것을 사용하도록하였다. 표 1 에서 연결 구조로 사용한 멀티플렉서의 종류별 사용된 수를 보면 1개의 논리 블럭에서 구현될 수 있는 멀티플렉서의 사용이 많음을 알 수 있다. [10]의 결과를 보면 멀티플렉서의 크기도 크면서 총 입력수도 본 논문의 결과보다 큼을 알 수 있다.

표 1. 실험 결과
Table 1. The experimental results.

			Data Path Width	# of C_STEP	# of FUs	# of registers / total inputs	TYPE /# of MUX.
DAVIC	i8251	main	1	5	CNT8(1) + (1)	4 / 29	MUX21/22 MUX41/56
		Tx	1	9	CNT8(1) CNT4(1)	2 / 9 (SR 1 / 8)	MUX21/12 MUX41/9
		Rx	1	17	CNT8(1) CNT4(1) + (1)	4 / 28 (SR 1 / 8)	MUX21/25 MUX41/23
			1			GR 26/111	
	TLC		1	1	0	5/15	MUX21/1 MUX41/3
	GCD		1	2	- (1) COMP(1)	4/18	MUX21/40
	ARMS_COUNTER †		1	1	+ (1) - (1)	6/12	MUX21/3 MUX41/2
	DIFFEQ		16	4	* (2) + (1) - (1) COMP (1)	5/80	MUX21/6
[10]	i8251	main	8	5		1/7	** 6/12
		Tx	8	8	- (1)	5/13	12/31
		Rx	8	13	- (1)	3/11	9/22
		Hunter	8	7	- (1)	2/14	4/11
	MTLC		2	6	0	0	4/10
	GCD		4	1	- (1)	2/8	9/25
	COUNTER		4	1	+ (1) - (1)	1/4	1/3
	DIFFEQ		16	4	* (2) + (1) - (1)	5/80	10/23

CNT8 : 8bit down counter
CNT4 : 4bit down counter
COMP : comparator
LR : Local register
GR : Global register
SR : 8bit Shift register
MUX21 : 2 input multiplexer
MUX31 : 3 input multiplexer
MUX41 : 4 input multiplexer
MUX51 : 5 input multiplexer
** : The number of multiplexers and multiplexer inputs
† : 4개의 process문으로 구성되어 있음.

Ⅸ. 결 론

본 논문에서는 ASIC 라이브러리나 FPGA를 이용하여 회로를 합성하는 상위 레벨 합성기를 개발하였다. 개발된 상위 레벨 합성기는 ASIC 라이브러리의 특정 셀의 사용을 지원하며 프로세스 간에 공유하는 전역 변수는 별도로 처리하여 시스템 레지스터를 구현할 수 있도록 하였다. 또한 하드웨어 할당 시 ASIC과 FPGA의 라이브러리 이용을 목표로 하여 cost함수를 설정함으로서 단순히 연결 구조의 입력 수 보다는 실제 구현되는 하드웨어의 단가를 줄이고자 하였다.

그러나 계층 설계에 대한 지원, RAM 이나 FIFO같은 메모리 소자에 대한 지원등에 연구가 더욱 진행되어야 하며 특히 논리 합성 툴에서 적용하는 VHDL의 하드웨어 모델링을 적용함으로서 사용자가 쉽게 이용할 수 있도록 하는 것이 앞으로의 연구 과제이다. 또한 다양한 형태의 기술 형태를 지원할 수 있는 방법론에 대한 연구가 필요하다.

참 고 문 헌

[1] N. Park and A. Parker," SEHWA:A PROGRAM FOR SYNTHESIS OF PIPELINES", Proc. 23rd DAC, pp. 454 - 460, 1986.

[2] I. Ahmad and C. Y. Roger Chen," Post-Processor For Data Path Synthesis Using Multiport Memories", Proc. ICCAD-91, pp. 276-279, 1991.

[3] M. Potkonjak and Jan Rabaey,"

Optimizing Resiurce Utilization using Transformations", Proc. ICCAD-91, pp. 88-91, 1991.

[4] K. Hwang, A. Casavant, M. Dragomirecky and M. d'Abreu," Constrainted Conditional Resource Sharing in Pipeline Synthesis", ICCAD-88, pp. 52-55, 1988.

[5] S. Note, W. Geurts, F. Catthoor and H. De Man," Cathedral-III : Architecture-Driven High-level Synthesis for High Throughput DSP Applications", Proc. 28th DAC, pp. 597-602, 1991.

[6] G. Zimmermann,"MDS : The Mimola design method", Journal Digital System, vol. VI, no. 3, pp. 337-369, 1980.

[7] C.-J. Tseng, R.-S. Wei, S. G. Rothweiler, M. M. Tong and A. K. Bose,"Bridge:A versatile behavioral synthesis system, Proc. 25th DAC, pp.415-420, 1988.

[8] F. Brewer and D. Gajski," Chippe : A System for Constraint Driven Behavioral Synthesis", IEEE trans. CAD, vlo. 9, No. 7, pp 681-695, 1990.

[9] G. De Micheli, D. C. Ku, F. Mailhot and T. Truong," The Olympus Synthesis System", IEEE Design and Test, pp. 37-53, Oct. 1990.

[10] High Level VLSI Synthesis edited by R. Camposano and W. Wolf, Kluwer Academic Publisher, 1991.

[11] D. Thomas, E. Lagnese, R. Walker, J. Nestor, J. Rajen, R. Blackburn, Algotithm and Register-Transfer Level Synthesis: The System Architect's Workbench, Kluwer Academic Publishers, 1990.

[12] T. E. Fuhrman," Industrial Extensions to University High Level Synthesis Tools: Making It Work in the Real World", Proc. 28th DAC, pp. 520-525, 1991.

[13] 김 기 현, 정 정 화," ASIC 설계 지원용 스케줄링", 전자공학회 논문지 95년 7월호 게재 예정.

[14] C. Tseng and D. P. Siewiorek,"Facet: A Procedure for the Automated Synthesis of Digital Systems", Proc. 20th DAC pp. 490 - 496, 1983.

[15] M. A. Breuer, Design Automation of Digital Systems, vol. 1, Theory and Techniques, Prentice-Hall, Inc., 1972.

저 자 소 개

金 基 鉉(正會員) 第31卷 A編 第 1號 參照
현재 한양대학교 전자공학과 박사과정

鄭 正 和(正會員) 第31卷 A編 第 1號 參照
현재 한양대학교 전자공학과 교수