상위수준 합성

黃 善 泳, 田 弘 信
西江大學校 電子工學科

Ⅰ. 서 론

최근 VLSI 제조기술이 발전하면서 칩의 집적도가 증가함에 따라 칩의 제작에 있어서 설계의 복잡도가 증가하였고, 그의 해결을 위하여 실리콘 컴파일러로 대변되는 설계 자동화 기법이 대두되었다. 실리콘 컴파일러란 소프트웨어 공학에서 이미 확립된 컴파일러의 개념을 하드웨어 영역에 도입한 것으로, 어셈블리 언어로 프로그래밍 하듯이 직접 레이 아웃을 생성하지 않고, hardware description language (HDL)[26]을 사용하여 하드웨어의 동작을 기술한 화일로부터 레이아웃을 자동적으로 생성하는 시스템이다. 실리콘 컴파일러는 받아들이는 설계기술의 수준에 따라 셀, 모듈, 프로세서, 시스템 컴파일러로 분류되고 있다.[9] 셀 컴파일러와 모듈 컴파일러는 많은 연구가 수행되어 실제 칩의 제작에 사용되고 있으나 입력 기술의 수준이 한 단계 높은 프로세서 컴파일러와 시스템 컴파일러는 현재 대학과 연구실 수준에서 연구가 진행되고 있으며 유수의 CAD 툴 벤더들이 90년대 프로젝트로 계획하고 있는 분야이다.[1] 프로세서 컴파일러가 일반적으로 말하는 상위수준 합성에 해당하고, 시스템 컴파일러는 이보다 한단계 더 높은 수준으로 프로세서 컴파일러의 수행작업의 복잡도를 줄이기 위해 받아들인 알고리듬 수준의 기술을 여러개의 프로세스로 작업을 분리하고 그들간의 통신을 생성하는 분야(algorithmic partitioning)로 아직은 먼 장래에 사용이 가능할 것으로 보인다. 본고에서는 현재 연구의 필요성이 부각되고 있는 상위수준 합성에 대해 설명한다.

Ⅱ. 상위수준 합성

상위 수준 합성은 설계하고자 하는 하드웨어의 동작을 알고리듬 수준에서 받아들여 데이타패스와 콘트롤러로 구성된 레지스터 트랜스퍼 수준의 설계표현을 생성하는 과정이다.[2] 주어진 하나의 동작기술로부터 그 기능을 수행하는 구조는 여러가지가 존재할 수 있어 상위수준 합성 과정에서는 설계 공간내의 구조중에서 주어진 제약조건인 클럭 싸이클, 칩 면적, 파워소모 등을 만족하면서 특정 비용을 최소화하는 구조를 찾아내는 것을 목표로 한다.[20]

상위수준 합성을 수행함으로써 설계과정이 자동화되어 칩의 설계시간을 줄일 수 있어 급변하는 반도체 시장에 대처할 수 있게 되고, 칩 개발에 소요되는 비용중 대부분의 비용이 개발에 소용된다는 점을 감안할 때 비용을 효과적으로 줄일 수 있다. 또한, 오류를 줄일 수 있고, 상위수준 합성 시스템을 사용하여 설계가 가능한 구조를 넓게 탐색함으로써, 최적의 하드웨어 구조를 찾을 수 있다. 설계과정의 문서화는 향후에 필요한 노우하우 축적에 중요한 역할을 하며, 설계과정에서 내린 결정이 자동적으로 기록되는 장점이 있다. 그리고, 칩 설계에 필요한 숙련된 기술을 시스템으로 구현함으로써 설계기술을 직접적으로 접하기 힘든 사람에게도 칩을 설계할 수 있는 환경을 조성해 준다.[20]

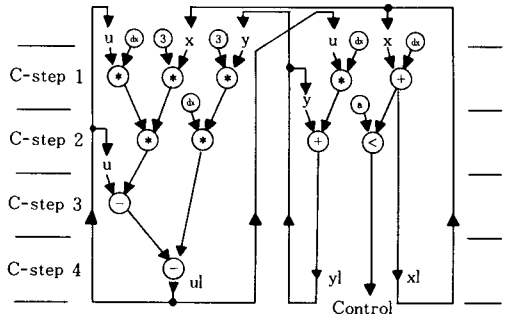
상위수준 합성의 과정은 다음과 같다. 합성의 첫 단계로 입력 HDL을 컴파일하여 합성에 편리한 중간형태를 만든다. 중간형태로는 그래프 형태의 표현을 주로 사용하고, 그 종류로 Value Trace[19], Sequenc-

ing Graph[12], Control Data Flow Graph(C/DFG)[3,4] 등이 있다. 중간형태의 하나인 C/DFG는 제어흐름(control flow)과 데이타흐름(data flow)을 동시에 표현한다. 제어흐름은 연산식의 우선순위와 분기나 루프등으로 제어흐름을 바꾸는 문에 의해 결정되고, 데이타흐름은 지정문(assignment statement)간의 데이타 의존성(data dependency)에 의해 결정된다. 다음 단계로 상위수준 합성에 있어서 핵심적인 부분인 스케줄링 과정과 모듈할당 과정이 수행된다. 스케줄링은 중간형태로 저장되어 있는 각 연산을 제어구간에 할당하는 과정으로 제어구간은 동기화된 시스템에서 기본적인 시간단위로 클럭 싸이클에 해당한다. 모듈할당은 연산과 변수를 레지스터, 연산자, 연결구조로 구성되는 하드웨어에 할당하고 그 사용을 결정하는 과정이다. 각 과정에서 사용되는 자세한 알고리듬은 3장에서 다룬다.

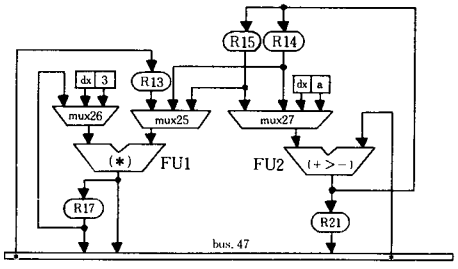
그림 1은 HAL 시스템[23]이 입력으로 사용한 HDL과 합성된 데이타패스를 보이고 있다. 그림 1의 (a)는 미분 방정식의 해를 구하는 하드웨어를 HDL을 사용하여 기술한 것이고, (b)는 중간형태, (c)는 합성된 데이타 패스이다. 사용자는 그림 1의 (a)와 같이 원하는 하드웨어의 동작을 기술하고 합성에 필요한 제약조건을 주면 상위수준 합성 시스템이 자동적으로 부가된 제약조건에 만족하면서 주어진 동작을 수행하는 데이타패스를 그림 1의 (c)와 같이 합성한다.[10]

```
while(x<a) repeat :
   x1=x+dx;
   u1=u-(3*x*u*dx)-(3*y*dx);
   y1=y+(u*dx);
   x =x1;  u=u1;  y=y1;
end;
```

(a) 동작기술



(b) 중간형태



(c) 데이타 패스

그림 1. 상위수준 합성의 과정

Ⅲ. 상위수준 합성 알고리듬

1. 스케줄링 알고리듬

스케줄링의 목적은 연산을 제어구간에 할당하되 제약조건을 만족하는 범위내에서 주어진 목적함수를 최소화하는 것이다. 목적함수는 필요에 따라 다양하게 정의하여 사용되고 있으며, 그 예로 사용하는 제어구간의 수, 자연시간, 전력의 소모, 하드웨어 자원 등이 될 수 있다. 여러 제약조건을 동시에 만족하는 데이타패스를 합성하는 것이 상위수준 합성의 목적이나, 시간과 면적 두 조건만을 고려하여도 설계공간이 너무 방대하여 두 조건을 동시에 만족하는 데이타패스를 합성하기란 매우 어렵다. 시간 혹은 면적중 하나의 제약조건을 입력으로 다른 하나를 최소화하는 것을 목적으로 하는 알고리듬이 대부분이다.[2,20]

스케줄링 알고리듬으로 iterative/constructive 방식이 주로 사용되고 있다. Iterative/constructive 알고리듬은 정해진 우선순위 함수에 따라 한번에 하나씩의 연산을 스케줄링하여 모든 연산이 스케줄될 때까지 반복하는 방법이다. 이 경우 우선순위를 결정하는 것이 핵심이고 면적이나 지연시간을 적절히 모델링하는 것이 필요하다. 표 1은 상위수준 합성 시스템과 사용하는 우선순위 함수를 나타낸다.

ASAP(as soon as possible)스케줄링은 가장 간단한 방법으로 연산을 가장 빨리 스케줄될 수 있는 제어구간에 할당하는 방식이다. Urgency와 freedom은 연산이 스케줄될 수 있는 제어구간의 범위를 말하며, 스케줄링 과정에서는 urgency와 freedom이 작은 연

표 1. 스케줄링의 우선순위 함수와 시스템

우선순위 함수	시 스 템
ASAP	CMUDA, MIMOLA, Flame I
Urgency	Elf, ISYN
Force	HAL
Freedom	MAHA

산을 우선적으로 한다.[17,21] 즉, 임계경로에 위치한 연산이 우선순위가 가장 높다.

FDS (force directed scheduling) 알고리듬[22]은 iterative/constructive 부류의 알고리듬의 대표적인 것으로 HAL 시스템에서 구현되었다. FDS 알고리듬은 서로 다른 제어구간에 사용되는 하드웨어는 공유될 수 있다는 개념하에 연산자, 레지스터, 연결 구조의 사용을 제어구간에 균등히 배분하는 것을 목표로 한다. Force의 결정을 위하여 time frame과 distribution graph (DG)가 계산된다. Time frame은 각 연산에서 정의되며 ASAP와 ALAP 스케줄 결과로부터 정해지고 임의의 연산이 스케줄될 수 있는 제어 구간의 범위를 의미한다. Time frame이 구해졌을 때 한 연산이 제어구간에 할당될 확률은 time frame 구간 거리의 역수가 된다. DG는 비슷한 연산의 병렬성을 의미하고 그 값들은 특정 종류의 연산이 각 제어구간에 할당될 확률을 더한 것이다. DG는 각 연산자에 대하여 각 제어구간에서 정의되며, 제어구간 i에서의 DG는 식 (1)과 같이 계산된다.

$$DG(i) = \sum Prob(Opn, i) \qquad (1)$$

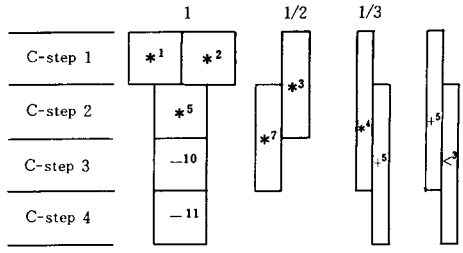
Prob (Opn, i) : 연산 Opn이 제어구간 i에 할당될 확률

그림 2의 (a)는 그림 1의 (b)에 대한 time frame을 보인 것이고 그림 2의 (b)는 이에 대한 DG이다.

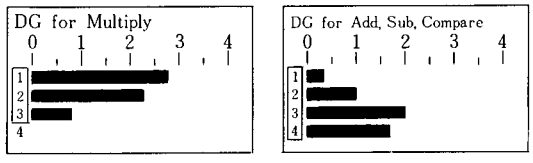
한 연산을 특정 제어구간에 할당하면 time frame에 변화가 발생하고, 그 결과로 전체적인 병렬성이 변하게 된다. 이 time frame의 변화의 영향을 정량적으로 표현한 것이 force이다. Force는 스프링에 적용하는 탄성계수 K와 변위 x로 이루어진 Hooke의 법칙을 적용한 것으로, 각 연산자에 대한 확률 분포인 DG를 탄성계수로, 연산이 특정 제어구간에 할당될 때 확률의 변화를 변위로 보고 있다. 기본적인 제어구간 i에 대한 force는 식(2)와 같다.

$$Force(i) = = DG(i) * x(i) \qquad (2)$$

C/DFG상의 한 연산노드 OP가 특정 제어 구간에



(a) Time frame



(b) Distribution graphs

그림 2. 그림1 (b)의 중간형태에 대한 time frame과 distribution graphs

할당될 때 OP 자신의 time frame이 변화함에 따라 발생하는 force를 self force라 하며, 이는 time frame의 모든 제어구간에서 발생하는 force를 더한 것으로, time frame이 t에서 b일때 연산을 time frame 내부의 제어구간 j (t≦j≦b)에 할당할 경우 self force, SF (j)는 식 (3)과 같다.

$$SF(j) = \sum_{i=t}^{b} [Force(i)] \qquad (3)$$

Self force는 스케줄되는 노드 자신에서 발생하는 force이며, 스케줄되는 노드의 predecessor와 successor의 time frame도 변화가 생겨 force가 발생하게 되어 이를 스케줄링에 고려하기 위해 각각 predecessor force와 successor force라고 정의하고 force에 첨가하게 된다. FDS에서 이 3가지 force의 합을 우선순위 함수로 사용한다.

Repeat until (all operbtions scheduled)
  Step 1: Evaluate time frames;
    1.1 Find ASAP schedule.
    1.2 Find ALAP schedule.
  Step 2: Update DG.
  Step 3: Calculate self force for every feasible control step.

Step 4: Add predecessor and successor forces to self force.

Step 5: Schedule the operation with lowest force at the corresponding control step (c-step)
Set the time frame of the selected operator to c-step

FDS 알고리듬은 위와 같이 스케줄할 모든 연산에 대해 스케줄 가능한 구간에 스케줄할 때의 force를 계산하고 그 중 가장 작은 force를 갖는 스케줄을 실행에 옮긴다. 이와 같은 작업을 모든 연산이 스케줄될 때까지 반복하는 iterative/constructive 방식을 취하고 있다. 이러한 iterative/constructive 방식의 알고리듬은 근본적으로 heuristic 정보를 사용하고 있으므로 어떠한 우선순위 함수를 사용하여도 모든 경우에 최적의 해를 보장할 수 없다. 이러한 문제점을 해소한 것으로 integer linear programming (ILP) 기법을 사용한 수학적인 접근방식이 있다. ALPS[7]는 ILP를 사용하고 있는 대표적인 시스템으로 먼저 사용하는 변수를 다음과 같이 정의한다.

(1) M_{ti}는 t_i 종류 연산을 수행하는 연산자의 수를 나타내는 정수

(2) o_i를 i번째 연산이라고 하고 $x_{i,j}$를 0 혹은 1을 갖는 변수라 할 때, 만약, o_i가 제어구간 j에 스케줄되면 $x_{i,j}=1$, 그렇지 않으면 0

이때 식 (4)에서 식 (6)까지의 조건을 만족하는 범위내에서 식 (7)을 최소화 한다.

$$\sum_{\substack{i=1 \\ o_i \in FU\,tk}}^{n} x_{i,j} - M_{tk} \leqq 0, \quad for\ 1 \leqq j \leqq s,\ 1 \leqq k \leqq m \qquad (4)$$

$$\sum_{j=Si}^{Li} x_{i,j} = 1, \quad for\ 1 \leqq j \leqq n \qquad (5)$$

$$\sum_{j=S_i}^{L_i} j * x_{i,j} - \sum_{j=S_k}^{L_k} j * x_{k,j} \leqq -1, \quad for\ all\ o_i \rightarrow o_k \qquad (6)$$

$$\sum_{i=1}^{m} C_{ti} * M_{ti} \qquad (7)$$

n : 연산의 갯수, s : 시간제약조건,
m : 연산자의 종류, L_i : o_i의 ASAP,
S_i : o_i의 ALAP

식 (4)는 어떠한 제어구간에서도 사용하는 t_k 종류의 연산자의 수가 M_{tk}를 넘을 수 없다는 면적 제약조건을 나타내고, 식 (5)는 한 연산 o_i는 ASAP 스케줄 S_i와 ALAP 스케줄 L_i내의 제어구간중에서 한군데만 스케줄된다는 제약이고, 식 (6)은 연산 o_i가 o_k보다 먼저 수행이 되어야 한다는 선행 관계를 나타낸다. 식 (7)은 최소화 하고자 하는 목적함수로 사용하는 연산자 면적의 총 합을 의미한다. 이들 제약조건과 목적함수를 입력으로 ILP를 수행시키면 주어진 제약조건을 만족하는 최적의 스케줄링 결과가 생성된다. 이 방법의 문제점은 스케줄할 연산의 수와 시간 제약 조건이 증가함에 따라 ILP에 사용하는 변수의 수가 증가하고 그에 따라 해를 구하는 시간이 많이 걸린다는 단점을 가지고 있다. 그러나 현재 컴퓨터의 성능이 날로 향상되고 있는 추세이며 파이프라인 데이타패스, 멀티싸이클링, 체이닝 등 상위수준 합성에서 고려사항을 간단하게 수식으로 모델링하여 해결할 수 있고, 항상 최적의 해를 제공한다는 장점을 가지고 있어 각광을 받고 있는 방법이다.

2. 모듈할당 알고리듬

스케줄링의 결과로서 결정된 연산자의 제어구간에 대한 정보는 특정한 중간형태로 변환되고, 모듈할당의 입력으로 주어진다. 스케줄링의 결과외에도 설계자가 제공할 수 있는 시간이나 면적의 제약조건을 참조하며 주어진 동작기술을 만족시키는 방향으로 모듈할당이 수행된다. 모듈할당의 목적은 사용하는 연산자, 메모리, 연결구조 등의 RTL 구성요소를 최소화 하는 것이다.[2] 이들 요소를 동시에 최소화하는 것이 궁극적인 목표이나 이들을 동시에 고려한다는 것은 복잡한 문제로, 대부분의 시스템들이 최적의 결과를 생성하지는 못하지만 이를 분리하여 최소화 하는 방식을 택하고 있다. 연산자의 경우 사용하는 제어구간이 서로 다른 경우 하나의 연산자를 사용하고 그의 입력과 출력을 먹싱하여 공유함으로써 면적을 줄일 수 있다. 레지스터의 할당에 있어서 입력 동작기술에서 사용한 변수의 생존 구간(lifetime)이 겹치지 않을 경우 그 변수들은 하나의 변수로 치환될 수 있고, 따라서 하나의 레지스터로 구현이 가능하다. 연결구조의 할당에 있어 연산자나 레지스터간의 통신경로가 같은 제어구간에 사용되지 않는 부분은 MUX나 BUS를 통하여 공유가 가능하다. MUX만을 사용하여 연결구조를 구성할 경우 알고리듬이 간단하지만 통신선로의 낭비를 유발할 수 있다. BUS를 사용하면 통신선로의 면적을 줄일 수 있는 반면 와이어의 케패시턴스가 증가하고 삽입된 tri-state 버퍼의 영향으로 동작속도가 감소한다. 따라서, 응용에 따라 MUX와 BUS를 혼합한 형태를 주로 사용한다.[20]

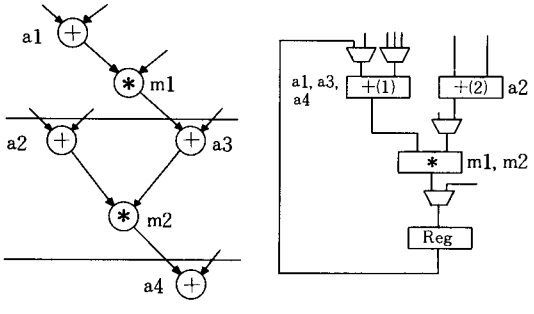
모듈할당은 각 RTL 구성요소를 동시에 고려하지 못하고 분리하여 최소화하므로 그들간의 상호작용을 고려해야 하고 각 단계에서 그 순서에 맞는 목적함수를 사용해야 한다. 일반적으로 레지스터 할당, 연산자 할당, 연결구조 할당의 순을 주로 사용하고 있다. 예를 들어, 레지스터 할당의 경우 사용하는 레지스터의 갯수를 최소화 하는 것도 중요하지만 그 후에 수행될 연결구조의 할당을 고려하여 레지스터의 사용을 결정하는 것이 필요할 것이다.

모듈할당에 사용하는 알고리듬은 크게 iterative/constructive 알고리듬과 광역적 알고리듬의 두 부류로 구분된다.[2] Iterative/constructive 방식은 할당할 레지스터, 연산자, 연결구조를 하나씩 찾아 할당하는 과정을 모든 RTL 구성요소가 할당될 때까지 반복하며, 광역적 알고리듬은 모든 RTL 구성요소를 그래프나 수학적 방정식 등으로 모델링하여 한번에 여러개의 하드웨어를 할당하게 된다. Iterative/constructive 알고리듬은 모듈할당의 수행시간이 빠르다는 장점이 있지만 지엽적인 공간의 탐색으로 인하여 최적화된 결과를 얻기가 어렵다는 단점이 있다. 광역적 알고리듬은 할당이 가능한 모든 가능성을 탐색하므로 많은 수행시간이 소요되지만 최적화된 할당 결과를 얻을 수 있다.

CMUDA의 모듈할당기[27]는 지엽적인 선정기준을 사용하였고, DAA 시스템[18]은 할당할 순서의 결정에서 지엽적인 탐색을 수행하였고 할당시킬 장소의 결정은 expert 시스템을 사용하였다. EMUCS는 할당할 순서와 장소의 결정에 있어 광역적인 탐색을 수행하였다.[10] HAL 시스템은 스케줄링된 데이타의 흐름에 대한 연산모듈을 선택하기 위해 rule-based expert 시스템을 사용하였고, iterative 알고리듬을 사용하여 모듈할당을 수행한다. MIMOLA 시스템[24]은 연산모듈의 할당에는 광역적 알고리듬을 사용하고 나머지 모듈에 대해서는 iterative 알고리듬을 사용하며, Chippe[11]와 DAA 시스템은 rule-based expert 시스템으로 iterative 알고리듬을 이용한다.

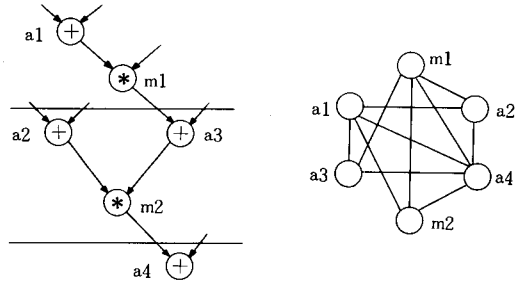
Iterative/constructive 모듈할당의 예를 그림 3에 보였다. 모듈할당의 순서는 제어구간이 작은 것부터 할당을 수행하고, 목적함수는 연결구조를 최소화 하도록 하고 있다. 먼저 첫번째 제어구간에서 a1과 m1이 각각 서로 다른 모듈로 할당되고, 두번째 제어구간의 연산에 대하여 곱셈기의 입력의 먹싱 비용을 줄이기 위하여 a2가 덧셈기 2에 할당되고 a3가 덧셈기 1에 할당되었다. 세번째 제어구간에 대하여 레지스터로 부터 연결 경로가 존재하는 덧셈기 1에 할당되어 연결구조 비용의 증가를 최소화하도록 한다.

광역적 모듈할당 알고리듬에는 그래프 이론에 바탕을 둔 알고리듬과 branch-and-bound 알고리듬 그리고 수학적 접근방식인 ILP 기법이 있다. 그래프이론 중 클리크 파티셔닝 알고리듬에 바탕을 둔 Facet 시스템[8]에서는 연산자, 레지스터, 연결구조의 할당에 있어서 할당하고자 하는 하드웨어를 노드로, 그들간의 공유가 가능한 경우를 간선으로 연결한 그래프를 구성한다. 그래프로 모델링된 경우 모듈할당의 문제가 그래프에서 노드의 집합을 찾아내는 것으로 변환된다. 클리크란 전체 그래프에서의 노드의 집합으로써 모든 노드들 사이에 간선이 존재하는 complete 그래프를 말하고, 하나의 클리크에 포함된 연산은 하나의 연산자로 동시에 공유가 가능하다. 모듈할당 과정의 목적이 사용하는 하드웨어의 최소화이므로 모듈할당 과정은 주어진 그래프를 최소의 클리크로 분리해 내는 문제로 해결할 수 있다. 클리크 파티셔닝의



(a)스케줄링 결과 (b) 합성된 데이타패스

그림 3. Iterative/constructive 알고리듬의 예



(a) 스케줄링 결과 (b) Compatibility graph

그림 4. 클리크 파티셔닝을 사용한 연산자 할당

의 문제는 NP-problem으로 휴리스틱 알고리듬을 이용한다. 그림 4는 클리크 파티셔닝으로 연산자 할당을 수행한 예이다.

그림 4에서 서로 다른 제어구간에 스케줄된 덧셈기 및 곱셈기는 서로 공유가 가능하므로 그림과 같은 그래프가 형성된다. 위의 경우는 덧셈기와 곱셈기를 하나의 연산으로 동일하게 취급함으로써 각 노드 사이에 간선이 존재하는 예이다. 그러나 일반적으로 효율적인 연산모듈의 할당을 위해서 수행시간, 면적, 기능에서 차이가 나는 연산자간의 공유는 간선을 제거하거나 우선순위를 낮추는 방향으로 할당이 수행된다.

Branch-and-bound 기법은 연산을 연산자에, 변수를 레지스터에, 연결을 BUS나 MUX에 할당할 때 발생하는 모든 경우를 시도해 봄으로써 최적의 결과를 얻을 수 있다. 이 기법은 모든 할당 결과를 탐색할 수 있으므로 효율적인 결과를 제공하지만 탐색공간이 너무 방대하기 때문에 휴리스틱 정보를 사용하여 경우의 수를 제한하게 되고 결국 최적의 결과를 보장받지 못하게 된다. Splicer[5]와 MIMOLA시스템[24]이 이 방법을 사용하였다.

수학적인 접근방법으로 ILP 기법이 있다. 이 기법은 각 RTL 구성요소의 가능한 모든 할당을 변수로 선언하고, 그 변수에 해당하는 할당이 이루어질 경우 1로, 할당되지 않을 경우 0으로 지정한다. 그리고 각 할당은 한 하드웨어로 이루어진다는 것과 같은 제약조건들을 가하여 정한 목적함수를 최소화 하도록 ILP로 표현하여 변수의 값을 구하는 방법으로 모듈 할당을 수행한다.[2,20]

Ⅳ. 상위수준 합성의 추세

1. 입력 HDL

현재까지 개발된 상위수준 합성 시스템들은 다양한 입력 HDL을 사용하고 있으며, 디지탈 시스템의 상위수준 행위는 하나의 알고리듬이므로 단순히 C, Pascal, Fortran 등의 일반 프로그래밍 언어를 사용하여 기술할 수 있다. 예로서, Flamel 시스템[15]은 Pascal을, HARP 시스템[28]은 Fortran을 입력으로 받아들인다. 그러나, port나 bit등의 하드웨어 특정 요소들은 일반 프로그래밍 언어로 기술하기 어렵다는 단점이 있어 하드웨어 기술 전용 언어의 사용이 불가피하다. 일반적으로 하드웨어 기술 언어는 일반 프로그밍 언어의 장점을 유지하면서 하드웨어 특정 요소에 대한 기술이 가능해야 한다.[26]

사용된 HDL로써는 Yorktown Silicon Compiler (YSC)의 YLL,[26] Emerald 시스템의 ISPS,[8] HARP 시스템의 Fortran, Olympus 시스템의 HardwareC[13]등을 예로 들 수 있다. 그러나 현재까지 발표된 많은 합성 시스템은 각기 다른 하드웨어 기술 언어를 사용함으로써 시스템 사이에 설계 데이타의 호환성이 결여되어 설계 데이타를 공유할 수 없는 비효율성을 초래하였다. 실리콘 컴파일러로 대변되는 설계 자동화 기술이 도입됨에 따라 설계 데이타의 호환성은 필수적으로 되었고 입력 하드웨어 기술 언어의 표준화가 요구되었다.

VHDL은 1981년부터 개발되기 시작하여 1987년 IEEE에 의해 공식 표준화된 하드웨어 기술 언어이다.[16] VHDL은 시뮬레이션과 문서화를 초기목적으로 개발되었으며 설계하고자 하는 하드웨어의 다층 수준 시뮬레이션이 가능하고 동기 및 비동기 시스템의 타이밍 표현도 지원된다. 또한 아키텍춰 단계부터 게이트 단계까지 행위 기술을 할 수 있어 계층적인 설계가 가능하여 하드웨어 기술언어로서의 모든 조건을 갖추었다고 볼 수 있다. 현재 VHDL을 입력언어로 사용하는 시뮬레이션 시스템과 논리회로 합성 시스템은 상용화되어 있고 행위 단계의 기술을 받아들이는 상위수준 합성시스템의 개발이 활발히 진행되고 있다.[34]

2. 아키텍춰 구동 상위수준 합성

고전적인 상위수준 합성의 경우 소프트웨어 공학에서의 컴파일러 개념을 그대로 도입하여 임의의 알고리듬 기술에 대하여 자동적인 합성을 수행 하고자 하였다. 이에 대한 많은 연구가 진행된 싯점에서 한계성이 드러나기 시작하였고, 한편에서는 DSP와 같은 특수한 목적으로 범위를 줄이고 타겟 아키텍춰를 선정하여 주어진 알고리듬 기술에서부터 타겟 아키텍춰의 데이타패스를 최적으로 찾기 위한 연구가 진행되고 있다.[14]

SPAID 시스템[6]은 아키텍춰 구동 합성시스템으로써 타겟 아키텍춰로 N개의 레지스터 파일에 저장된 데이타가 레지스터 파일 각각에 연결된 N개의 BUS를 통하여 연산자에 전달되는 구조를 타겟으로 하고 있고, CATHEDRAL 시스템[14]은 meet-in-the-middle 설계방식을 채택하고 있는 것이 특징이며 입력 행위 기술의 동작속도에 따라 타겟아키텍춰를 표 2와 같이 4가지로 분류하여 서로 다른 합성방법을 사용하고 있다.

표 2. CATHEDRAL 시스템

구 분	타겟 아키텍춰	샘플링 레이트	응 용 분 야
CATHEDRAL Ⅰ	Bit Serial Datapath	수백K	음성처리, Audio
CATHEDRAL Ⅱ	Microcoded Processor	1 M	Audio, 중저급영상처리
CATHEDRAL Ⅲ	Cooperating Datapath	수M	Video, 레이다
CATHEDRAL Ⅳ	Regular Array	수십 M	〃

3. 연구동향

국내에서도 세계적인 추세에 따라 실리콘 컴파일러 개발에 관한 연구가 진행중이다. 이에는 한국전자통신연구소에서 개발중인 EDAS(electronic design automation system)[37]와 서강대에서개발중인 SSC (Sogang silicon compiler)시스템[39]이 있다. EDAS는 공학용 디자이너, 완전주문형 디자이너, 게이트 어레이 디자이너, 셀 디자이너와 실리콘 디자이너로 구성되어 있으며, 상위수준 합성 시스템은 실리콘 디자이너에 포함될 예정이고 현재 개발중이다.

SSC 시스템은 크게 다음의 네가지 분야로 구분된 시스템들의 통합으로 framework를 구성하며, 이 SSC 시스템의 framework를 그림5에 보인다.

- HDL simulation 및 compile
- Behavioral synthesis(high level synthesis)
- Logic synthesis
- Layout synthesis 및 verification

SVSIM(Sogang VHDL simulator)[30]는 HDL 로서 VHDL을 입력으로 하여 AST(abstract syntax tree)를 구성한다. AST로부터 합성과 시뮬레이션을 위한 자료 구조를 가지는 C/DFG를 생성하고, C/DFG의 control 및 data flow를 따라 event-driven 시뮬레이션을 수행한다. 생성된 C/DFG는 합성에 사용되는 정보가 포함되므로 시뮬레이터와 합성기는 하나의 VHDL 분석기를 이용한다. 합성의 효율성을 높이기 위한 중간 형태 언어인 SBIF(Sogang behavioral intermediate format)[33]도 사용 가능하다.

SVSIM에 의해 동작이 검증된 동작기술은 상위수준 합성기인 SHSS(Sogang highlevel synthesis system)[39]로 합성하여 데이타패스를 SLIF(Sogang logic intermediate format)로 출력하며, 각 모듈의 제어를 위한 control signal을 제어 시간별로 결정하여 T-

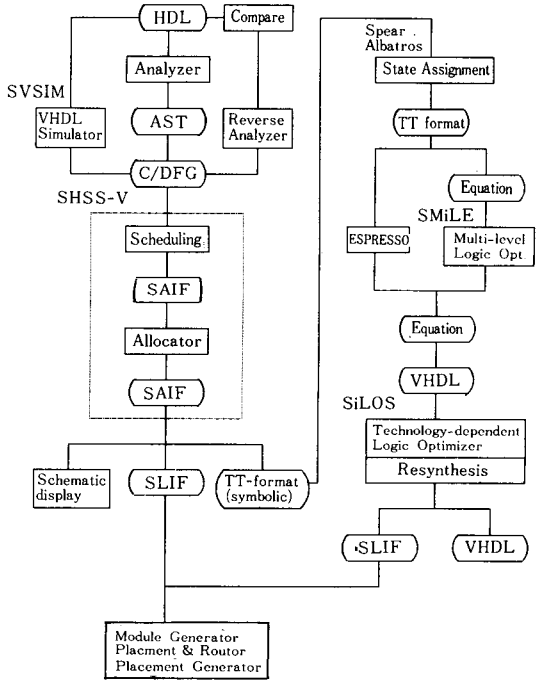


그림 5. 서강 실리콘 컴파일러 시스템

format으로 출력한다. 출력된 TT-format의 각 state는 symbolic 형태를 갖는다. SAIF(scheduling allocation interchange format)는 스케줄링과 모듈 할당의 원활한 연결을 위해 사용된 중간형태이다.

TT-format의 symbolic 형태로 정의된 각 state의 최적화된 bit coding을 위한 SPEAR[32], Albatros[31]는 상태 할당을 수행하여 그 결과를 TT-format으로 출력한다. 출력된 TT-format은 이단 논리식의 형태를 가지며 PLA의 최소화된 형태 생성을 위하여 ESPRESSO[25]를 거치거나, equation으로 변환하여 다

단 논리 합성과정을 거친다. Decomposition, extraction, collapsing, phase assignment 등의 방법을 algorithmic approach로 구현한 SMiLE(Sogang multi-level logic minimizer)[38]은 회로의 다단 논리 합성을 이루며 면적을 최소화하며, SiLOS(Sogang inteiligent logic optimization system)[35]는 다단 논리합성의 결과를 기술 의존 논리 최적화 과정을 통하여 동작 속도 및 면적 제약 조건을 고려한 최적화와 각 논리소자를 라이브러리에서 제공하는 게이트로 변환하는 technology mapping을 이룬다.

SHSS에서 합성된 데이타패스의 모듈 중 라이브러리에서 제공되지 않는 모듈은 논리 합성을 통하여 논리를 최적화 한 후, module generation 과정을 거쳐 회로의 레이아웃을 생성한다.[36] 생성된 각 레이아웃의 수정 또는 기능 향상을 위한 레이아웃 editor 및 레이아웃으로 부터 동작 기술의 추출을 위한 extraction을 수행하여 verification을 수행한다.[29] 상위수준 합성의 결과로 출력된 SLIF에서 지정하는 모듈의 레이아웃이 생성되면, SLIF내의 모듈간의 연결 관계를 입력으로 하여 배치 및 배선 과정을 수행하여 최종적인 레이아웃을 완성하게 되어 CIF(Caltech interchange format)로 출력된다.[40]

V. 결 론

본 고에서는 실리콘 컴파일러에서의 상위수준 합성에 대한 전반적인 문제를 설명하였다. 상위수준 합성 시스템은 세계 유수의 CAD 툴 제조 업체에서도 90년대 목표로 설정하고 있는 분야이고, 주어진 알고리듬 수준의 기술로부터 설계 스타일을 결정하는 문제를 포함하여 인터페이스 디자인, 시스템 레벨 합성, 상위수준의 변환 등 앞으로 해결해야 할 많은 문제를 가지고 있다. 이러한 문제의 해결을 통하여, 논리 합성 이후 RTL 합성을 거쳐 다음 단계인 진정한 의미의 상위수준 합성 시스템의 등장과 실용화를 위한 노력이 기대된다.

參 考 文 獻

[1] Proc. 1st CADENCE Design Automation Conf., Seoul, Korea, 1990.

[2] A.C. Parker, "Tutorial on High-Level Synthesis," in *Proc. of ACM/IEEE Design Automation Conf.*, pp. 330-336, June 1988.

[3] A.E. Casavant, D.D. Gajski, D.J. Kuck, "Automatic design with dependency graphs," in *Proc. of ACM/IEEE Design Automation Conf.*, pp. 506-515, June 1980.

[4] A. Orailoglu, D.D. Gajski, "Flow graph representation," in *Proc. of ACM/IEEE Design Automation Conf.*, pp. 503-509, June 1986.

[5] B.M. Pangrle, "Splicer: A heuristic approach to connectivity binding," in *Proc. of ACM/IEEE Design Automation Conf.*, pp. 536-541, June 1988.

[6] B.S. Haroun, M.I. Elmasry, "Architectural synthesis for DSP silicon compilers," *IEEE Trans. CAD*, vol. 8, no. 4, pp. 431-447, April 1989.

[7] C.T. Hwang, J.H. Lee, Y.C. Hsu, "A formal approach to the scheduling problem in high level synthesis," *IEEE Trans. CAD*, vol. 10, no. 4, pp. 464-475, April 1991.

[8] C. Tseng, D.P. Siewiorek, "Automated synthesis of data paths in digital systems," *IEEE Trans. CAD*, vol. 5, no. 3, pp. 379-397, July 1986.

[9] D.D. Gajski, *Silicon Compilation* , Addison Wesley; Reading, Mass., 1988

[10] D.E. Thomas, E.D. Lagnese, R.A. Walker, *Algorithmic and Register Transfer Level Synthesis: The System Architect's Workbench*, Kluwer Academic Publishers, Boston, Mass., 1990.

[11] F.D. Brewer, D.D. Gajski, "Knowledge Based Control in Micro-Architecture Design," *in Proc. of ACM/IEEE Design Automation Conf.*, pp. 203-209, June 1987.

[12] G.De Micheli, D.C.Ku, "HERCULES-A system for high-level synthesis," in *Proc. of ACM/IEEE Design Automation Conf.*, pp. 483-488, June 1988.

[13] G. De Micheli, "The Olympus Synthesis System," *IEEE Design & Test*, pp. 37-53, Oct. 1990.

[14] H. D. Man, F. Catthoor, "Architecture driven synthesis techniques for VLSI imple-

mentation of DSP algorithms," *IEEE Proceedings*, vol. 78, no. 2, pp. 319-335, Feb. 1990.

[15] H. Trickey, "Flame 1: A high level hardware compiler," *IEEE Trans. CAD*, vol. 6, no. 2, pp. 259-269, March 1987.

[16] *IEEE Standard VHDL Language Reference Manual*, IEEE Std. 1076-1987, IEEE, N. Y., March 1988.

[17] K.S. Hwang, A.E. Casavant, "Scheduling and Hardware Sharing in Pipelined Data Paths," in *Proc. of ICCAD*, pp. 24-27, Nov. 1989.

[18] M.C. McFarland, "Using Bottom Up Design Techniques in the Synthesis of Digital Hardware from Abstract Behavioral Descriptions," in *Proc. of ACM/IEEE Design Automation Conf.*, pp. 474-480, June 1986.

[19] M.C. McFarland, "The value trace: A data base for automated digital design," Master's thesis, Carnegie Mellon Univ., Pittsburgh, Dec. 1978.

[20] M.C. McFarland, A.C. Parker, P. Camposano, "The high-level synthesis of digital system," *IEEE Proceedings*, vol. 78, no. 2, pp. 301-318, Feb. 1990.

[21] N. Park, A.C. Parker, "SEHWA: A software package for synthesis of pipelines from behavioral specification," *IEEE Trans. CAD*, vol. 7, no. 3, pp. 356-370, March 1988.

[22] G.P. Paulin, J.P . Knight, "Force directed scheduling for the behavioral synthesis of ASIC's," *IEEE Trans. CAD*, vol. 8, no. 6, pp. 661-679, June 1989.

[23] P.G. Paulin, J.P. Knight, "HAL: A Multi-paradigm Approach to Automatic Data Path Synthesis of ASIC's," in *Proc. of ACM/IEEE Design Automation Conf.*, pp. 263-270, June 1989.

[24] P. Marwedel, "A new synthesis algorithm for the MIMOLA software," in *Proc. of ACM/IEEE Design Automation Conf.*, pp. 271-277, June 1986.

[25] R. Rudell, A. Sangiovanni-Vincentelli, "ESPRESSO-MV; Algorithms for multivalued logic minimization," in *Proc. Custom Int. Circ. Conf.*, pp. 23-28, Oct. 1984.

[26] R.W. Hartenstein, *Hardware Description Languages*, Advances in CAD for VLSI, vol. 7, North Holland, 1987.

[27] S.W. Director, A.C. Parker, D.P. Siewiorek, D.E. Tomas, "A design methodology and computer aids for digital VLSI systems," *IEEE Trans. Circuits Syst.*, vol. CAS-28, no. 7, pp. 634-645, July 1981

[28] T. Tanaka, "HARP: Fortran to silicon," *IEEE Trans. CAD*, vol. 8, no. 6, pp. 649 660, June 1989.

[29] Y.J. Lee, I.H. Moon, S.Y. Hwang, "A Logic/timing Extractor from Transistor Schematic," in *Proc. KITE Int. Conf. VLSI CAD*, pp. 11.3.1-11.3.4, Nov. 1991.

[30] 오대일, 황선영, "VHDL Behavior 단계 시뮬레이터의 구현에 관한 연구," 대한전자공학회종합학술대회 논문집, 제13권 제1호, pp. 547-550, 1990년 7월.

[31] 유진수, 황선영, "상위수준 기술로부터 순차회로의 자동 생성," 전자공학회 논문지, 제8권 제12호, pp. 115-124, 1990년 12월.

[32] 이기중, 황선영, "다단 논리회로에 기반을 둔 유한 상태기의 효율적인 상태 할당 알고리듬," 한국정보과학회 논문지, 제18호 제2호, pp. 184-194, 1991년 3월.

[33] 이봉선, 황선영, "행위단계 VHDL 합성 시스템을 위한 중간 언어의 설계," 대한 전자공학회 종합 학술대회 논문집, 제14권 제2호, pp. 547-551, 1991년 11월

[34] 이영희, 황선영, "VHDL 설계 환경 구축을 위한 Front-end의 설계," 한국정보과학회 논문지, vol. 18, no. 1, pp. 93-103, 1991년 1월.

[35] 이재형, 황선영, "성능 구동 논리 회로 자동 설계 시스템," 전자공학회 논문지, 제28권 A편 제1호, pp. 74-84, 1991년 1월.

[36] 이종욱, 김상범, 황선영, "프로시저 레이아웃 언어를 사용한 모듈 생성기의 구현," 대한전자공학회 추계종합 학술대회 논문집, 제13권 제2호, pp. 607-610, 1990년 11월.

[37] 이철동, "자동설계 환경구축에 관한 연구(Ⅱ)," 1989년 과학기술처 특정연구 최종발표회논문집, pp. 3-7, 1990년 8월.

[38] 임춘석, 김태선, 황선영, "SMiLE : 다단 논리최적화 시스템," 한국 정보과학회가을 학술발표 논문집, vol. 17 no. 2, pp. 689-692, 1990년 10월.

[39] 전홍신, 이해동, 황선영, "서강 실리콘 컴파일러

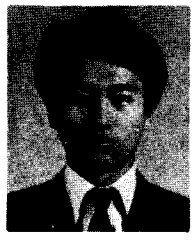
의 High Level Synthesis System의 설계," 전자공학회 논문지, 제 28권 제 6 호, pp. 82-93, 1991년 6월.

[40] 진훈상, 문인호, 황선영, "매크로 셀 배치를 위한 효과적인 알고리듬," 전자공학회 논문지, 제 28-A권 제 2 호, pp. 74-82, 1991년 2월.

筆 者 紹 介

黃 善 泳

1954年 5月 20日生
1976年 2月 서울대학교 전자공학과(학사)
1978年 2月 한국과학원 전기 및 전자공학과(석사)
1986年 10月 미국 Stanford 대학(박사)
1976年 3月～1981年 7月 삼성 반도체 연구원
1986年 7月～1989年 1月 Stanford 대학 CIS 연구소 연구원
1986年 12月～1988年 2月 Fairchild Semiconductor 기술 자문
1989年 3月～현재 서강대학교 전자공학과 교수
주관심 분야 : CAD시스템, Computer Architecture 및 System Design, VLSI 설계 등임.



田 弘 信

1967年 8月 22日生
1989年 2月 서강대학교 전자공학과(학사)
1991年 2月 서강대학교 전자공학과(석사)
1991年 3月～현재 동대학원 (박사과정 재학중)
주관심 분야 : High-level Synthesis, DSP Architecture, VLSI 설계 등임.