개선된 시뮬레이티드어닐링 기법에 의한 디지탈필터 설계의 고찰

송 낙 운† 윤 복 식††

요 약

본 연구에서는 스케쥴링과 하드웨어 할당에 관련된 상위단계합성에서 최적설계방법론을 효과적으로 변형된 시뮬레이티드 어닐링 기법을 사용하여 정립한다. 또한 정립된 기법을 디지탈필터(DF : digital filter)의 설계에 적용하여 파이프라인 DF의 경우 최적설계시에 속도와 하드웨어의 최적의 절충 문제와 어레이 DF에서의 해석에 관련된 문제점을 검토한다. 이러한 적용사례를 통해 제안된 방법이 보다 빠른 시간에 향상된 비용함수값을 줄 수 있음이 확인되고 복잡한 디지탈필터 설계에 이용될 수 있음이 입증된다.

Investigation of Digital Filter Design using Improved Simulated-Annealing Technique

Nag Un Song† and Bok Sik Yun††

ABSTRACT

In this work, the optimized design methodology in high-level synthesis related with scheduling and hardware allocation is developed by simulated annealing technique effectively modified. Applying this method to digital filter design, the optimized tradeoff problem of speed and hardware costs in pipelined digital filter case and array digital filter case are investigated. While, it is confirmed that the suggested method gives the improved cost function value faster and can be used in complicated digital filter design.

1. 서 론

최근 디지탈 시스템 구성이 고도로 고집적 VLSI화 되면서 톱다운설계(top-down design)를 위한 자동화 설계툴의 개발에 관한 연구가 활발히 진행되고 있다. 이러한 설계과정의 상층부는 일반적으로 상위단계합성(HLS: high-level synthesis)과 로직합성(logic synthesis)으로 나눌수 있는데, 이중 VLSI의 알고리듬에서 아키텍처에 이르는 행위의 설계 및 합성에 해당하는 HLS의 분야에 최근 많은 관심과 연구가 있어 왔다[1, 2].

HLS의 과정은 크게 데이타경로 설계, 콘트롤 경로 설계의 두 가지로 분류해 볼 수 있다. 이때 HLS의 효율적인 수행에 가장 핵심이 되는 것은 데이타 경로를 최적의 콘트롤 스텝에 배정을 하여주는 스케쥴링과 이에 따른 하드웨어(HW: hardware)의 할당(allocation)의 문제라고 볼 수 있다. 이러한 문제들은 전형적인 NP-hard 문제가 되므로 적당한 시간내에 최적의 해를 구하기가 매우 힘들어지므로 여러가지 근사적인 최적화 기법이 요구된다. 더우기 전체 시스템의 복잡성을 최소화하면서 주어진 임무의 처리 속도를 최대화하는 최적의 디자인을 구현하기 위해서는 스케쥴링과 하드웨어 할당의 문제를 상호 연관시켜 고려하여야 할 필요가 있다. 이를 위해 기존의 연구에서 사용된 기법은 크게 두 가지로 나누어

† 정 회 원 : 홍익대학교 전자공학과 부교수
†† 정 회 원 : 홍익대학교 기초과학과 조교수
논문접수 : 1994년 9월 8일, 심사완료 : 1994년 1월 11일

볼수 있다. 첫번째 접근 방법은 먼저 스케쥴링을 적절히 임의로 설정하여 그 상황하에서의 할당 문제의 최적해를 구해내어 이 최적해로 할당을 하여 놓고 그안에서 최적 스케쥴을 구하는 방식을 반복적(iterative)으로 수행하며 개선해가는 방법이다[3, 4]. 이러한 반복적, 교대적(alter-native)인 방법에 반해, 두번째 접근 방법은 스케쥴링과 할당을 동시에 고려하여 전체적인 비용함수를 만들어 이를 최소화 시키는 스케쥴과 하드웨어 할당을 찾아내는, 보다 직접적인 방법으로 최근에 많이 발표되어 그 효율성을 인정받고 있는 방법이다. 이의 대표적인 예로는 HAL[5], MAHA[6]등의 발견적인(heuristic) 방법, SA (simulated annealing)를 이용한 global search 방법[7], 정수 계획법을 이용한 방법[8] 등이 있다. 물론 후자의 방법이 최적화의 관점에서 바람직하나 일반적으로 계산의 복잡성 때문에 적용가능한 문제의 규모에 제한이 있을 수 있다. 이중, 정수계획법을 이용한 접근 방법은 적용할 수 있는 문제의 규모가 매우 제한되기 때문에 실용성이 상당히 제한되는 편이며, 발견적인 기법은 계산상의 신속성이 큰 잇점이나 근본적으로 최적해를 보장해 주지 않기 때문에 적용에 유의해야 한다. 이에 비하여 SA는 속도가 느리고, 해가 비용함수의 설정에 크게 의존하게 되는 단점이 있으나, 최적해에 가까운 해를 얻을 수 있고 거의 모든 조합적 최적화 문제(combinatorial optimization problem)에 적용할 수 있다는 범용성 때문에 여러 분야에서 많이 사용되고 있다[9]. 즉, 발견적 기법인 FDS(force-directed scheduling)는 최종 결과가 항상 의도한 최적성을 갖기는 기대하기 힘드나, 이에 비해 SA기법은 본래적인 최적해에의 수렴성과 범용성때문에 보다 실용적인 큰 규모의 문제에 적합한 기법이라고 볼 수 있다.

이러한 HLS의 기법을 포함하여 전반적인 설계과정의 자동화 환경을 구축하는 실리콘 컴파일러에 관하여, 특히 디지탈 신호처리(DSP: digital signal processing)에 관련된 시스템 칩의 자동화 설계에 관해 많은 연구가 있어 왔는데, 이중에서도 디지탈 필터(DF: digital filter)의 설계는 매우 실용적인 가치가 있다. 이러한 DF의 구현은 CAD의 도움을 받아 컴파일레이션 개념에 의해 설계하는 것이 일반적인 추세이다.

Broderson et al.[10]은 상위단계 필터기술을 하위단계 레이아웃으로 전환하는 작업을 통하여 각종 통신 시스템에 응용되는 DF 뱅크를 구현하였으며, 아울러 이를 UC, Berleley대학에서 개발한 CAD 툴인 OCT 툴의 일부와 결합하여 체계화된 DSP용 실리콘 컴파일러 LAGER를 완성하였으며[11], Man et al.[12]은 CATHEDRAL 소프트웨어의 개발을 통하여 톱다운 설계를 수행하였다. Hwang et al.[13]은 그들의 컴파일러인 PARSIFAL을 사용하여 각종 DSP-IC를 개발하였으며, 아울러 파이프라인 구조에서의 스케쥴링, 하드웨어 공유에 의한 DF의 설계에 관한 연구도 수행하였으며, Haroun and Elmasry[14]는 분산병렬 구조에서의 DSP 알고리듬의 매핑 및 구현에 관한 연구를 수행하였다.

그러나 많은 HLS 연구에도 불구하고, 실제 DF의 자동화 설계시 발생되는 문제들을 효용성의 측면에서 고찰한 연구는 많지 않으며, HLS의 수행에서 스케쥴링 방법을 약간씩 변형하여 다양한 접근을 시도하였을 경우의 결과를 비교 검토한 연구도 많지 않은 편이다. 본 연구에서의 주안점은 우선 융통성(flexibility)이 매우 뛰어 나기 때문에 다양한 접근방법을 고려하는데 알맞은 알고리듬인 SA를 기본으로 하여, 우선 이것을 충분한 효율성을 갖도록 개선하는 방법을 찾아보는 것이고, 다음으로 얻어진 알고리듬을 이용하여 HLS의 스케쥴링및 하드웨어 할당문제의 효과적인 해결방법을 찾는 것이다. 그 후 이것을 실제적인 DF 아키텍처 설계 문제에 적용하여 파생되는 제반문제를 검토하여 향후 DF의 자동화 설계를 위한 CAD 환경구축을 시도한다. 본 서론에 이어 2장에서 전체적인 최적해를 줄 수 있는 SA 방법을 보다 빨리 수렴할수 있도록 개선하여 스케쥴링과 할당문제를 동시에 해결하는 구

현방법을 제시하고 3장에서는 이를 DF의 다양한 구조설계 문제에 적용하며 실제의 구조에서 일어나는 제반문제를 HLS에 의한 최적화설계의 관점에서 검토한다. 끝으로 4장에서 결론과 향후 연구방향이 언급된다.

2. 알 고 리 즘

조합적 최적화 문제의 한 해법으로서 Kirkpatrik[15] 등에 의해 제안된 SA는 기존의 반복적인 개선에 근거한 발견적 기법들이 국부적 최소점에 빠져버리는 단점을 개선한 범용화된 최적화 기법으로 현재까지 CAD를 비롯한 많은 분야에서 응용되고 있다[9, 15, 16]. 기본개념의 단순성과 범용성이 두드러지는 SA는 전체 최소점에로의 수렴성이 이론적인 증명[17, 18]과 같은 많은 실험결과 들[9, 19]을 통해 타당성이 확인되어 특별한 대안이 없을때 편리하게 사용할 수 있는 최적화 알고리듬으로써 일반적으로 받아들여지고 있다. 그러나, 아직까지 해결되지 못하고 있는 문제점들(예를 들면 쿨링 스케쥴, 종료기준 등)이 있어서 충분히 효율적인 SA의 적용이 이루어지지 못하고 있다. HLS의 경우에는 Devadas[7]에 의해 적용이 시도되었는데 만족할만큼 효율성 및 실험이 추구되지 못하였다. 즉, 문제의 모형화 방식과 알고리듬 적용방법에 개선할 여지가 많은데 본 연구에서는 우선 효과적으로 변형된 SA를 제시하고 실제 적용에 있어서 파이프라인 구조등 다양한 설계 방식에 적용될 수 있도록 개선한다.

2.1 문제의 형태

일단 입력 프로그램이 CDFG(control data flow graph) 형태로 변환되어 있다고 가정하자. HLS에서의 스케쥴링 및 하드웨어 할당 문제는 세로 방향을 콘트롤 단계의 시간슬롯 축으로 하고 가로 방향을 하드웨어 축으로 하는 2차원 격자형을 만들어 CDFG에서 주어진 오퍼레이션을 배치하는 문제로 정립할 수 있다. 즉 이 주어진 문제의 기본 연산을 O_i, $i=1,...,n$ 이라고 하고 해집합을 S 라고 하면 하나의 해, $X \in S$ 는,

$$X=[(TimeSlot_i, ALU_i)_{i=1,...,n}]$$
subject to: CDFG, 하드웨어 제약, 시간제약 (1)

과 같이 표현될 것이다. 이때 S는 기본적으로 2차원 격자구조의 형태지만, CDFG와 파이프라인 설계에서의 다양한 잠복주기(latency), 하드웨어 그룹의 모듈화등에서 발생하는 배후구조를 가지고 있는 해공간이다.

해(스케쥴링과 하드웨어 할당이 이루어진 상태) X의 시간과 하드웨어를 고려한 비용함수는,

$$\begin{aligned} C(X) = {} & p1 * Number\_Alu(X) + p2 * Number\_Register(X) + p3 * Number\_Bus(X) + \\ & p4 * Number\_Mux(X) + p5 * Number\_TimeSlot \end{aligned} \quad (2)$$

로 놓을 수 있다. 여기서 p1, p2, p3, p4는 각각 해당소자의 단위면적에 해당하는 하드웨어비용이며, p5는 단위 시간슬롯(timeslot)에 의한 수행시간(execution time)비용을 의미하며 파이프라인 구조시에는 잠복주기에 따라 시간슬롯 숫자가 정해지게 된다. 본 연구에서는 연산의 특성에 따라 걸리는 다르게 줄 수 있도록 허용하여 보다 일반적인 상황에서의 스케쥴링을 행할 수 있도록 한다. 예를들어 덧셈기는 1 시간슬롯, 곱셈기는 2 시간슬롯등으로 ALU에서의 지연시간을 다르게 줄 수 있다. (2)에서 Number_Alu(X), Number_Register(X), Number_Bus(X)를 구할때는 [5, 7]에서 사용하고 있는 각 시간슬롯에서의 이들의 값을 구하여 그 중 최대값을 선택하는 방법을 개선시켜서 파이프라인 방식과 모듈연결이 고려될 수 있도록하여 보다 정확한 하드웨어 비용을 반영될 수 있도록 한다.

식 (1), (2)에서 볼 수 있듯이 본 연구에서 고려되는 최적화 문제는 매우 융통성이 있다. 즉, SA에서 해의 이동범위를 (1)의 제약조건에 따른 가능한 해의 집합(feasible solution set) 범위

내부로 제한함으로써, 설계자가 의도하는대로 하드웨어 제약적인 문제, 또는 시간 제약적인 문제를 풀어서 최선의 디자인을 얻어 낼 수 있다.

2.2 기본 SA 알고리듬 일반적인 SA 알고리듬의 형태는 다음과 같다.

```
ALGORITHM SA0
INITIALIZE(X,T,M);
repeat
  for i=1 to M do
      Y = PERTURB(X);
      if (C(Y)≤C(X)) or (exp((C(X)−C
        (Y))/T)>random(0,1)) then
      X=Y; {accept the movement}
      endif;
    endfor;
    UPDATE(T,M);
until(Stop_Criterion);
```

SA는 해(X)로부터 적절한 방법으로 새로운 해(Y)를 만들어서 새로운 해의 목적함수의 값이 현재 해의 그것보다 작으면 새로운 해를 최적해 후보로 받아들이고, 그렇지 않을 경우에도 무조건 버리지는 않고 적절한 확률로 새로운 나쁜 해를 받아들여서 국부최소점에서 빠져나올 수 있는 가능성을 주어 전체 최소점을 찾아가게 하는 반복적 알고리듬이다. 이 기본적인 방법을 SA0로 칭하며 이의 변형형태는 다음절에 보인다.

2.3 SA의 효과적인 변형

SA의 개선의 목표를 첫째 최적에 가까운 해에 빨리 도달하게 하고, 둘째 사용자가 미리 설정해 주어야 하는 파라미터의 수를 되도록 줄여 새로운 문제에의 적용이 간편하게 하고 셋째 되도록 SA의 수렴성에 대한 이론적인 결과를 수용하여 수행의 신뢰성을 유지하는 것으로 설정하고 다음과 같은 변형을 행한다.

(1) 안쪽 for 루프 내부에서 그때까지의 최소의 목적함수 값을 준해를 기억하여 최종해로 준비한다. 이것은 보다 최적에 가까운 해를 구하는 방법이다.

(2) acceptance criteria : 기본 알고리즘에서는 해 변동 결과 목적함수 값이 커졌을 경우에 Metropolis 기준을 적용하여 확률 exp(−D/T), D=(변동후의 목적함수 값)−(변동전의 목적함수 값)으로 변동된 나쁜 해를 받아들이고 있는데 알고리즘의 수렴성을 위해서는 꼭 이 기준을 따를 필요는 없는 것이 경험적으로[26] 또는 이론적으로[27] 관찰되고 있다. 이 기준을 T가 클 때 exp(−D/T) 1−D/T 임을 이용하여 확률 (1−D/T)로 받아들이도록 수정하여 계산의 효율성을 높일 수 있다(SA2에서만 적용).

(3) 냉각스케쥴 : 수렴의 촉진을 위하여 기본적으로 기하적인 스케쥴을 사용하되 내부 for루프의 반복 횟수 L을 줄이기 위하여 매회 내부 루프를 끝낸 후 만약 비용 감소가 없으면 T를 그대로 두고, 있으면 a*T (a=0.9 또는 0.95로) 감소시킨다. 이렇게 하는 이유는 이론적으로 고정된 T에 대해 내부 루프의 전이를 L번 일으킨 마코프체인이 안정상태에 도달해야 하는데 안정상태에서는 목적함수의 평균값이 T값이 작아짐에 따라 낮아져야 한다. 따라서 내부 루프를 돌고 난 후에도 목적함수 값에 변화가 없으면 아직도 온도 T에서의 안정상태에 도달하지 못했다고 간주할 수 있으므로 T를 고정시킨 채 내부루프를 계속 돌린다. 이렇게 함으로써 L을 최소로 줄일 수 있어서 내부루프를 필요 이상 많이 돌리는 데 따른 시간의 낭비를 방지할 수 있고 개별 문제에 따라 적절한 L을 설정해야 하는 번거러움을 예방할 수 있다. 실제로 이러한 방법을 통해 많은 계산상의 이득을 볼 수 있었다

이와 같은 고려에서 다음과 같은 변형된 알고리즘 SA2(변형 1,2,3 적용)를 얻을 수 있는데 이때 중간단계를 SA1(변형 1,3 적용, 변형 2의 acceptance 기준은 Metropolis 기준을 적용)으로 놓고 성능을 비교한다.

```
ALGORITHM SA2:
INITIALIZE(X,T,L);
XBest=X;
Counter=0;
repeat
  CostOld=C(X);
  for i=1 to L do
    Y=PERTURB(X);
    if(C(X)-C(Y))RANINT(-T,0))then
          X=Y; {accept the movement}
      endif;
    endfor;
  CostNew=C(X)
  UPDATE(T, CostOld, CostNew)
  if (C(X)<C(XBest))then
    XBest=X;
  endif;
  if(CostNew=CostOld)then
    Counter = Counter+1;
  else
    Counter=0;
    endif;
  until(Counter>M);

  procedure UPDATE(T, CostOld, CostNew)
  if (CostOld>CostNew) then
      T=a*T
  endif;
```

2.4 구현 방법

앞 절의 알고리즘들을 실제문제에 적용하기 위하여 아직 미정인 과정들을 확정해 주어야 하는데 본 연구에서는 아래와 같은 방법을 택하였다.

(1) INITIALIZE(X,T,M)에서 초기해(X)를 ASAP 스케쥴과 ALAP 스케쥴중에서 비용이 적은 스케쥴링으로 두고 초기온도 T, 내부루프 수 M은 적당히 큰 값으로 준다. HLS에의 적용에서는 T=100, M=100-200(SA0)(예제의 연산 갯수에 따라 증가), 20-100(SA1,SA2)로 두고 실험을 하였다.

(2) PERTURB(X)에서는 다음과 같은 2가지 기본 이동을 고려한다.

a. 1개의 연산의 위치를 이동

b. 2개의 연산의 위치를 상호 교환

이중에 a는 원래 해와 유사한 근방의 해로의 이동이라고 볼 수 있으며, b는 현재의 해로부터의 비교적 급격한 이동이라고 볼 수 있다. 이동을 무작위적으로 만들어 내기 위해서 PERTURB(X)에서는 우선 난수를 연산 갯수의 범위내에서 1개 발생시켜 기준연산의 번호를 정한 후에, 전체 연산 갯수의 5배 정도의 범위에서 새로운 난수를 발생시켜 난수가 총 연산의 갯수보다 클 경우는 이동 a를, 그밖의 경우는 b를 선택하는 방식으로 하여 실제로 a에 의한 이동을 더 많이 일어나게 조정한다. 이때 a의 경우 이동 위치는 가능한 범위내에서 무작위적으로 정하고 b의 이동의 경우에는 두번째 난수가 기준연산과 상호교환될 번호가 된다. 본 연구에서는 연산의 지연시간을 임의로 줄 수 있게 하였으므로 b에서 선택된 두개의 연산들이 차지하는 시간슬롯의 수가 다를 경우가 발생하는데 이때에는 이동하고자 하는 연산의 시간 슬롯의 크기를 고려하고 동시에 그 연산의 전후 관계를 살펴 가능한 이동의 범위를 설정한다.

(3) UPDATE(T,M)에서 온도(T)를 매 반복마다 약간 내려주어야 하는데(cooling schedule), 0.95 또는 0.9를 곱해 주고 M은 고정을 시킨다.

(4) Stop_Criterion은 반복루프 수행시에 5회 변할 때까지 비용함수가 변하지 않을 때 중단하는 방식을 따른다.

위와 같은 구현방법을 기본으로 하여 각 문제마다 비용함수의 계산시에 약간의 변형을 두어 실행을 한다. 예로서 파이프라인의 비용함수 산출시에 이를 pipe 0, 1, 2로 구분하여, 단일 시간

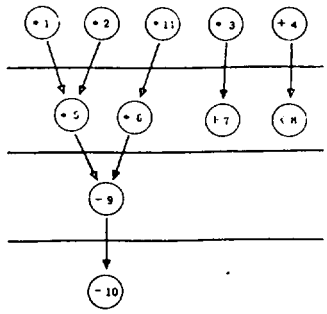
슬롯만 주로 고려하며 파이프라인의 가중치를 준 경우(pipe0), 해당시간슬롯의 H/W를 모두 고려하는 경우(pipe1), 전체 시간슬롯의 H/W를 모두 고려하는 경우(pipe2) 등으로 나누어 필요시 점진적으로 확장하여 수행하였다. 즉, 스케쥴링과 H/W 할당은 초기에는 기본적으로 동시에 이루어지나, 시스템구조의 다양화(예를 들면, 파이프라이닝 : pipelining 등) 혹은 총체적인 H/W의 정도에 따라 H/W할당시에 고려해야할 전체적인 모쥴연결을 보다 정확하게 계산하여갔다. 여기에는 2.1 절에서의 비용함수계산시의 H/W 항목 이외에도, ALU 및 레지스터 등의 공유시에 먹스(mux : multiplxer)의 H/W 비용을 추가하였다.

본 연구의 알고리즘은 C언어로 코딩되었으며, IBM-PC 486 DX에서 실험이 수행되었다.

3. 디지탈필터 설계에의 적용

3.1 미분 방정식 문제

참고문헌 [5]에서 언급되었던 HLS의 대표적인 벤치마크인 미분방정식 문제(그림 1)에 앞서의



(그림 1) 미분방정식 문제[5]의 CDFG의 ASAP(as soon as possible)에 의한 시간슬롯 배치
(Fig. 1) Timingslot allocation of CDFG by ASAP on differential equation problem[5]

SA 스케쥴링을 적용하면 전형적으로 (〈표 1〉 (a))와 같은 결과를 얻게 되는데, 이를 문헌 [5]의 FDS에 기반을 두어, 본 저자들이 개발한 소프트웨어를 사용한 결과[22](〈표 1〉 (b))와 비교해 볼 때 두 방법이 비슷한 결과, 즉, 각 시간슬롯에서 ALU의 숫자가 비교적 균등하게 분포, 즉, 평준화(leveling)됨을 알수가 있었다. FDS에서는 비용 함수에 해당하는 force를 줄이는 방향으로 발견적인 기법으로 최적화를 수행하였으며, 해당 시간슬롯에서의 ALU수와 수행시간만을 고려하였으며, 모든 연산기와 시간스텝의 비용은 동일하게 가정하였다.

다음에 이를 파이프라인 구조로 바꾸었을 경우, 잠복주기 L(latency : 문헌 [13]에서의 DII (data introduction interval : 데이타 진입간격, 또는 줄여서, 데이타 간격)에 해당)를 바꾸어가며 비용함수를 구하였다. L값이 커질수록 수행

〈표 1〉 미분방정식 문제의 스케쥴링 결과
〈Table 1〉 Scheduling results of differential equation

(a) SA 스케쥴링의 결과 예
(a) SA scheduling result example

C-step	ADD(+)	SUB(-)	MULTI(*)	etc.
0	4		1, 2	
1			3, 5, 11	
2		9	6	8
3	7	10		

(b) FDS 결과 예
(b) FDS result example

C-step	ADD(+)	SUB(-)	MULTI(*)	etc.
0	4		1, 2	
1			5, 11	8
2		9	3, 6	
3	7	10		

(c) 파이프라인 스케쥴링 결과
(c) Scheduling results of pipeling

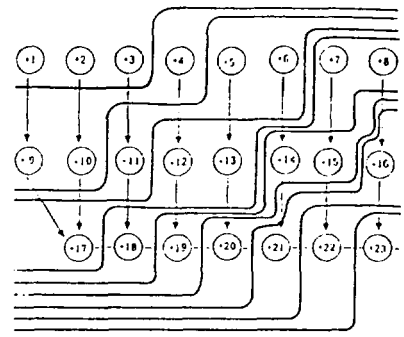
	SA 분류 (0,1,2)	latency			
		1	2	3	4
비용값	SA0	244	145	120	104
	SA1	"	145	119	"
	SA2	"	146	119	"
H/W(PE) 수	SA0	11	6	4	3
	SA1	"	"	"	"
	SA2	"	"	"	"
CPU 시간 (분,초,.01초)	SA0	0,0,22	0,2,85	0,1,98	0,2,64
	SA1	0,0,5	0,1,31	0,1,15	0,1,20
	SA2	0,0,6	0,0,66	0,0,60	0,0,77

시간 증가에 따라 해당 비용함수값이 증가된다. 관련 HW 비용은, ALU : 5, 레지스터 : 1, 버스 : 5, 잠복주기(L) : 10,으로 하였으며 pipe1의 계산에 의하여 앞의 방법(SA0, SA1, SA2)들을 적용하였으며, 이의 실험 결과는 〈표 1〉 (c)에 보였다. SA0, SA1, SA2의 순으로 계산시간이 단축되었으며 비용함수, PE결과는 비슷하였다.

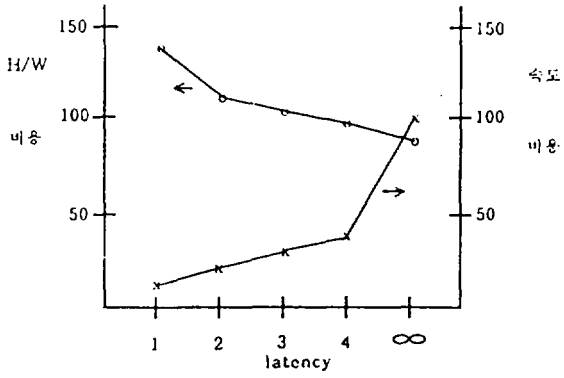
3.2 디지탈 필터

본 절에서는 SA를 변형한 알고리즘을 이용하여, 먼저 HLS의 벤치마크인 16차 FIR 필터의 문제[4]에 적용하여 다양한 변형을 가하여 최적 설계를 시도하였다. 이때 비용함수는 2.1절에서처럼 하드웨어, 속도 등을 고려하여 구한다. 단, 동일 시간슬롯에서 동기신호없이 연속되는 연산요소(PE : processor element, processing element)의 진행은 타이밍 제어 등에 문제점을 야기하게 됨으로 본 논문에서는 이러한 구조는 경우(〈표 2〉 (c))에 따라 적용하였다.

이제 적용 결과중에서 우선 L에 무관한 경우(∞ : 여기서는 L=10에 해당)의 한 예를 (그림 2) (a)(pipe0)에 보였다. 이 경우는 직렬연산에 해당하게 되는데, 그림에서와 같이, CP(critical path)를 따라 하드웨어 할당이 우선적으로 이루어짐과, 시급도 시간(urgency time)이 작을수록 초반에 기다리는 시간이 길게 배열되는 것을 알 수 있다. 이제 잠복주기가 있는 경우의 결과를 다음에 구하였으며, 이때 파이프라인에 의한 HW 가중치를 비용함수에 첨가하였다. 해당 비용변수계산에서, 곱셈연산은 덧셈연산에 비해 시간(비용)이 2배 걸리는 것으로 가정하며, HW 비용은 3.1절에서와 동일하였으며, PE 숫자는 5로 제한하였다. 관련 데이타경로 구현에 있어서는 계산을 단순화하기 위하여 각 변수의 최대치를 주어진 상태에 대한 변수의 값으로 대체 합산하여 계산하였으며, 제약조건이 있는 경우에는 벌점(penalty) 개념을 도입하여 고려하였다. 여기에서 곱셈기의 HW 비용을 덧셈기에 비해 2배로 했을때에도 총비용은 증가하나, 타이밍 슬롯 배치결과는 동일하였다. 한편, 곱셈연산에 필요한 시간은 덧셈연산의 2배에 해당하므로 시간슬롯을 2배로 점유하게 되며, 덧셈연산은 한 슬롯을 차지하게 된다. L을 가변한 경우, 즉, L=1,2,3,4,∞ 등,의 결과치를 그림 2.b(SA0, pipe0)에 나타내었다. 각각의 경우 연산기(+/*)의 숫자는 (15/8, 8/6, 7/5, 6/3, 4/2)로 얻어졌으며, 총 PE 숫자로는 약간 감소하였다. 여기서 L이 작아질수록 하드웨어 비용은 커지며 속도는 빨라졌으며, L이 커질수록 속도비용이 커지게 되어, 최적설계점이 존재함을 알수있다. 이때, L이 정해지면 그에 상응하는 최적 PE 수 및 이의 배열



(a) DF의 타이밍 할당의 한 예
(a) A timing allocation result of DF



(b) 파이프라인 구조시의 비용계산 결과(10슬롯, pipe0)
(b) Cost calculation results of pipelined structure(10 slot, pipe0)

(그림 2) FIR DF[4]의 스케쥴링 결과
(Fig. 2) Scheduling results of FIR DF[4]

상태가 자원 평준화의 형태로 배치되게 되어, PE 숫자에 제한이 있는 경우에는 L이 변하면서 수행속도관계비용을 변화시키게 됨을 알수 있다. 이때, L의 속도비용과 HW 비용은 서로 묶여져 변하는 관계에 있게 됨으로, 제약조건 및 필요에 따라 최적설계점을 구해야 함을 알 수 있다.

다음에, 해당 시간슬롯에서의 H/W 비용을 전부 고려한 경우의 스케쥴링 결과를 (〈표 2〉(a))에 보였다. 단, 이때 PE의 제한은 없도록 가정하였다. PE 배치는 앞서와 대동소이하였으며, SA0에 비하여 SA1, SA2의 순으로 CPU time이 개선되었고 비용치, PE 결과는 비슷하였다. 다음으로는, 시간간격이 큰 경우, 즉, 연속되는 덧셈기가 동일 시간슬롯에 들어갈수 있는 경우의 스케쥴링 결과를 다음 (〈표 2〉(b))에 보였다. 역시 CPU time이 개선(SA0, SA1, SA2 순)됨을 알수가 있다. 이는 다음 pipe2의 방법 중 먹스와 레지스터의 비용을 무시하여 단순화한 경우로 볼수가 있다. 다음에는, 앞의 비용 변수에서 모든 시간 슬롯에서의 먹스를 포함한 H/W 모두를 고려하는 경우를 2.3절의 방법에 의하여 스케쥴링을 수행하였다. 해당 H/W의 비용은 앞에서와 약간 다르게, 즉, 다음(PE=10(+),20(*), mux=1, reg.=5, exe.time=10)과 같이 택하였다. 이때, 버스(링크 포함) 비용의 경우, 이를 공유하는 경우가 많으므로 이를 버스의 수(complexity)에 따라 시간 슬롯을 상충이 되지 않도록 배치 후 합쳐서 공유할수 있도록 할수는 있으나, 여기에서는 레지스터와 먹스에 이를 포함시켜 대체함으로 단순화하였으며[24], 즉, p3=0, 이때 먹스의 입력수(2,3,4)의 종류에 따른 결과도 보였다. PE의 경우, 파이프라인 L에 따라 먹스를 도입하여 PE 전후의 레지스터도 포함하여 HW를 가능한 공유할수가 있게 되므로 비용산출에 이를 고려하였다. 모의실험결과를, (〈표 2〉(c))에 보였다. L의 값 4 근처에 최적 비용함수값이 존재하였다. 모듈연결을 다 고려함으로 계산시간은 증가하였으나, 역시 SA0, SA1, SA2의 순으로 개선되었으며, PE의 총 숫자는 전체적으로 앞에서와 비슷한 값을 가졌다. CPU 계산시간의 경우 개선된 SA2의 방법의 경우에는, FDS 방법에 근간을 둔 문헌[13]의 결과와 비슷하였다. 나아가서, PE의 면적 가중치도 고려하면 배치배선(placement & routing)시에, 이들에 관련된 연결선버스의 가중치가 변화함으로 실질적인 상황의 계산이 가능할 것이다.

〈표 2〉 (그림 2)의 스케쥴링 결과
〈Table 2〉 Scheduling results of (Fig. 2)

	SA 분류 (0,1,2)	latency 1	2	3	4	5	6	7
비용값	SA0	532	292	213	182	166	171	185
	SA1	-	-	213	-	167	170	175
	SA2	-	-	219	-	172	170	-
H/W(PE) 수 (+,*)	SA0	15,8	7,4	6,4	6,3	4,2	4,2	4,3
	SA1	-	8,4	7,4	5,3	4,3	5,2	5,3
	SA2	-	8,4	7,5	6,3	5,3	5,2	-
CPU 시간 (분,초,.01초)	SA0	0,0,72	0,5,66	0,5,76	0,6,92	0,8,19	0,6,21	0,5,93
	SA1	0,0,22	0,2,64	0,3,74	0,3,40	0,3,95	0,3,84	0,4,23
	SA2	0,0,17	0,2,69	0,2,86	0,2,3	0,2,20	0,2,9	-

(a) 파이프라인 스케쥴링 결과(10 슬롯, pipe1)
(a) Scheduling results of pipeling(10 slot, pipe1)

	SA 분류 (0,1,2)	latency 1	2	3	4	5	6
비용값	SA0	484	260	199	180	176	180
	SA1	-	-	197	180	176	175
	SA2	-	-	198	172	165	175
H/W(PE) 수 (+,*)	SA0	15,8	8,4	6,4	5,3	6,3	5,3
	SA1	-	-	7,3	5,3	6,3	-
	SA2	-	-	6,3	5,4	5,2	-
CPU 시간 (분,초,.01초)	SA0	0,0,50	0,4,34	0,4,45	0,4,12	0,3,79	0,3,90
	SA1	0,0,17	0,2,31	0,1,82	0,1,92	0,2,31	0,2,97
	SA2	0,0,11	0,1,70	0,1,9	0,1,82	0,1,65	0,2,25

(b) 파이프라인 스케쥴링 결과(6 슬롯, pipe1)
(b) Scheduling results of pipeling(6 slot, pipe1)

	SA 분류 (0,1,2)	latency 1	2	3	4	5	6
비용값	SA0	535	399	374	359	377	378
	SA1	-	-	364	359	377	376
	SA2	-	-	374	335	369	388
H/W 수 +,*,reg.,mux mux입력수종류 (2,3,4) H/W 수	SA0	15,8,41,0	8,4,40,19 (19,0,0)	6,3,38,14 (7,7,0)	5,3,39,14 (4,9,1)	6,3,39,12 (2,9,1)	3,2,43,13 (2,4,7)
	SA1	-	8,4,40,19 (19,0,0)	5,3,38,14 (5,9,0)	5,3,39,14 (4,9,1)	6,3,39,12 (6,5,1)	3,2,43,11 (0,4,7)
	SA2	-	8,4,40,19 (19,0,0)	6,3,38,14 (7,7,0)	5,2,39,10 (0,3,7)	5,3,39,12 (2,9,1)	4,2,43,13 (4,2,7)
	문헌[15]	15,8,57,0	8,4,42,40	5,3,43,42	4,2,50,41	4,2,46,39	3,2,43,37
CPU 시간 (분,초,.01초)	SA0	0,23,18	6,11,79	6,30,68	7,20,83	7,55,98	5,37,46
	SA1	0,7,9	4,34,46	4,7,39	7,33,85	4,49,51	3,46,13
	SA2	0,7,18	3,1,69	1,39,86	4,43,47	3,35,9	2,11,6
	문헌[15]	0,9,0	2,18,0	2,24,0	2,22,0	2,23,0	2,27,0

(c) 파이프라인 스케쥴링 결과(6 슬롯, pipe2)
(c) Scheduling results of pipeling(6 slot, pipe2)

다음으로 문헌[23]의 5차 wave elliptic DF를 SA기법에 의하여 HLS 최적화를 시도하였다. 일단 해당 소자의 단위비용을 (p1=10, p2=1, p3=5, p4=0, p5=10)으로 설정하고, 시간슬롯수

=17, pipe0의 방법을 택한 실험을 수행하였는데, SA0, SA1의 경우 CPU 시간의 경우, 193, 185(초)로 SA1의 경우가 약간의 개선이 있었으며, 이들은 문헌 [24]의 결과(2 min)와 비슷하였으며 PE의 수도 동일(+:3,*:2)하였다.

3.3 어레이 구조의 디지탈 필터

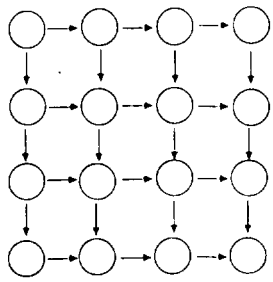
앞절의 구조를 실제의 DF에 그대로 적용하기에는 점검해야할 문제점들이 있다. 한 예로, 앞절의 스케쥴링에서는 회로구조가 지극히 불규칙하게 됨으로 HW 공유를 포함한 타이밍제어의 연결선비용이 증가하게 된다. 즉 이들의 타이밍 제어를 위한 제어기 및 이들을 위한 버스선, 배치배선을 포함한 연결선의 증가와 레지스터, 먹스, PE 등 데이타경로 등의 면적 관련 비용이 증가하게 된다. 아울러 지연래치에 관한 고려가 미흡하여 타이밍 동기에 무리가 있게 된다. 따라서 실제로는 규칙성 및 자원공유가 용이한 어레이 구조를 많이 이용하게 된다. 각종 어레이구조에서 설계의 제반적인 문제점(즉, 파이프라이닝, PE 가동도, PE 공유, 시간 스케일링, 스케쥴링 등)의 최적화의 고찰은 문헌[25]에 다양하게 제시되어 있으며, 아울러 다양화된 해석을 위하여는 노드가 합병 혹은 분할이 가능하며 각 연결선 및 노드에 관한 정보를 조절할수 있도록 해야하며, PE의 적절한 특성분석 및 변수화가 필요하다.

어레이구조에서의 최적화의 기준은 일반적으로 계산시간, 파이프라인주기, 블록 파이프라인주기, 어레이크기 등이 있으나, 본절에서는 앞절에서와 같은 비용함수를 택하였다. 이의 고려를 위하여 본 절에서는 지연래치와 시간 및 HW 제약조건을 삽입하나, 먹스는 무시하였다. 이러한 구조는 매우 규칙적이므로 시간제약조건이 허락하는 한, 주어진 블록주기(period x latency)의 잠복주기 안에서 PE간의 분담이 가능하게 되어 직렬처리가 용이하게 된다. 실제로 먹스를 포함하는 총모쥴연결을 고려한 구조에서는(예로, 〈표 2〉 (c)), PE간의 분담이 어느정도 가능해지나, 규칙성이 결여된 관계로 제어신호생성 관련비용이 실질적으로 증가하게 된다. 따라서 일반적으로 규칙적인 구조의 형태를 선호하게 되는데, 이에 따라 앞서의 1-D(일차원) FIR 필터를, 레지스터를 삽입한 일차원 어레이 구조로 가정하여, 진행방향에 직각이 되도록 파이프라이닝화하여 앞서에서와 같이 L의 속도비용과 H/W 비용의 조절문제, 즉, 관련 HW를 몇개를 묶어 1개의 PE를 갖는 구조로 볼수가 있으며, 이때 PE를 고려한 노드의 형태로 데이타흐름도를 대체하였을때, 임계경로는 1차원 선상에 위치하게 되어, 일차원 상의 최적화(스케쥴링 및 HW 할당) 문제로 볼수가 있게 된다. 이 경우에 보다 다양성을 부여하기 위해 비용함수를 파이프라인주기, 블록파이프라인주기, 어레이크기, 연산시간 등의 최적화에 맞추는 경우도 있게된다. 또한, 1 PE (*+)의 계산 주기가 앞서의 타이밍슬롯보다 더 커지는 문제가 생기므로 HW 공유가 커질수록(잠복주기 L 증가), 속도 범위에 주의를 기울여야 한다. 아울러 3.1절의 예에서처럼 계수의 대칭성에 의한 겹침(folding) 기법을 이용하여 HW를 줄일수가 있으며 이 경우 데이타 취득 시간이 보다 길어지게 되며, 덧셈기의 추가도 고려해야 된다. 전술한 바와 같이, 곱셈기의 시간이 기본적으로 많이 걸리므로, 1블록주기 안에서는 구조를 파이프라인 형태, 혹은 multi-rate 구조로 곱셈기에 필터계수 부분만을 가변입력이 가능하도록 배럴 쉬프터를 이용하거나, 혹은 ROM의 형태로 스위칭이 가능하도록 하는 것이 관련 비용을 줄이는 것이 된다. HW의 경우, PE 뿐만이 아니라 앞에서와 같이 연결선 면적도 고려해야 한다. 따라서 전체적인 타이밍을 고려한 총체적 구조를 택하느냐, PE를 채택한 어레이 구조를 택하느냐(차수는 어느 정도로 하나) 하는 것은 근본적으로 신호와 시스템 HW의 속도매칭에 의하여 결정한다. 이상의 논리는 transversal 형태의 FIR 필터의 구조에서도 동일하게 성립이 된다. 어레이 구조에서는 앞서와 같이 잠복주기와 PE(지연요소도 포함)의 절충이 매우 중요하게

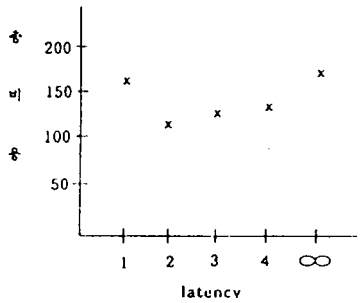
되며, 아울러 파이프라인 방향 및 주기를 보다 효율적으로 최적설계에 이용하여야 한다. 이러한 상황은 시스톨릭, wavefront 어레이 등을 포함한 다차원 어레이 구조에서도 공통적이며 앞서의 잠복주기, PE의 제약조건에 따른 비용함수의 최소화에 의한 DF의 최적설계 방법에 적용될수가 있다.

이제 이러한 다차원 어레이 구조를 DFG(data flow graph)형태로 바꾸어 앞에서 개발한 SA 기법을 적용할수가 있다. 그러나 이 경우 각 노드가 의미하는바는 앞절에서의 경우에서처럼 단순한 연산용 PE보다는 본절에서 이미 기술한 다양한 형태의 연산기(예로 multiply and add 등 마크로블록 모델포함)를 포함하며 각 노드간 연결선(arc)도 보다 복잡한 형태를 포함하게 된다. 한편으로는 거시적인 모델에 의한 이러한 방법을 이용하면, 예를 들어 2차원 DF의 구조를 DFG의 스케쥴링 문제로 대체하여 2D 어레이문제로 풀 수가 있게 된다. 실제적으로, 2D DF의 대표적인 2D FIR DF의 경우 필터계수의 대칭성에 의한 H/W의 효율을 고려하며, 1D DF와 line memory로 구현하는 것이 일반적이나[26], VLSI 제조기술의 발달에 따라 H/W 비용이 감소하게 되어 어레이 아키텍쳐 구조의 활성화가 이루어짐에 따라, 설계관련 HLS 연구도 활발하게되어, 본절의 앞 부분에서 기술한바와 같은 어레이구조에의 다양한 접근이 가능하게 되었다.

이와 같은 맥락에서 2D 어레이 DF의 문제를 풀기 위하여, 본절 앞부분에서 기술한 1D DF의 1차원의 어레이구조를 확장하여 2D FIR DF의 2차원적인 어레이구조의 DFG로 바꾼 모델을 ((그림 3) (a))에 보였다. 이때 각 노드와 연결선은 마크로블록의 의미를 갖고있다. 이를 대표적인 2D DF구조와 매핑시키기 위하여는 몇가지



(a) 어레이 구조
(a) Array structure



(b) 비용계산 결과(9 슬롯, pipe0)
(b) Cost calculation results(9 slot, pipe0)

(그림 3) 이차원 어레이 구조의 스케쥴링 결과
(Fig. 3) Scheduling results of 2-dimensional array structure

〈표 3〉 (그림 3)의 스케쥴링 결과(9 슬롯, pipe1)
〈Table 3〉 Scheduling results of (Fig. 3)(9 slot, pipe1)

	SA 분류 (0,1,2)	latency								
		1	2	3	4	5	6	7	8	9
비용값	SA0	573	309	234	188	187	175	185	195	205
	SA1	"	308	228	192	181	"	"	"	"
	SA2	"	309	228	191	181	"	"	"	"
H/W(PE) 수	SA0	16	8	6	4	4	4	4	4	4
	SA1	"	"	"	"	"	"	"	"	"
	SA2	"	"	"	"	"	"	"	"	"
CPU 시간 (분,초,.01초)	SA0	0,0,77	0,7,31	0,9,1	0,6,48	0,4,94	0,0,72	0,0,71	0,0,72	0,0,77
	SA1	0,0,22	0,3,84	0,0,16	0,0,77	0,0,33	0,0,22	0,0,17	0,0,22	0,0,17
	SA2	0,0,17	0,1,9	0,0,38	0,0,71	0,0,16	0,0,22	0,0,22	0,0,22	0,0,22

단순화가 필요하다. 즉, 신호데이타 입력과 출력은 각각 좌상에서 우하로 진행되며, scan data를 저장하는 line memory는 무시하였다. 한편, 2차원상의 각 필터계수는 노드내의 곱셈기의 한 입력으로 포함시켰다. 따라서 각 노드에 연결된 선은 각각 화살표의 진행방향에 따라 한 예로, (W : 입력화상데이타와 필터계수, E : 화상데이타와 계산데이타(carry)출력, N : 입력계산데이타(sum), S : 계산데이타(sum)출력), 등으로 정해지게되며, 아울러 노드내에는 예를 들어, 기언급한 multiply and add의 PE 소자가 있게 되며, 하드웨어 구현방식에 따라 다양한 방법이 있을수가 있다.

이 어레이구조를 속도에 최대역점을 두어 최적화하는 경우엔, 모든 경로가 임계경로상에 위치하게 되어 시간 및 HW의 재배치에 유연성이 없어진다. 따라서 시간 슬롯을 늘려가며 앞서의 SA 스케쥴링 기법에 의한 관련비용함수를 구한 결과(시간 슬롯 2 추가, PE를 3으로 제한, 관련비용: 레지스터(1), PE(5), 버스(5), 잠복주기(10) & pipe0 방법)를 ((그림 3) (b))에 보였다. 즉, L이 2인 경우에 최소비용이 들게 되었으며, 시간슬롯을 약간 증감한 경우에도 유사한 결과를 보였다. 이러한 해석은 일반 시스톨릭 구조에서의 규격화된 신호진행에 보다 유연성을 줄수 있게 된다.

다음으로는 해당시간 슬롯 전부를 고려한 pipe1의 방법에 의한 최적화결과를 〈표 3〉에 보였다. 단, 이때에 시간 슬롯을 9로 하였으며 PE의 숫자제한은 없게 하였다. 앞에서와 같이 SA0, SA1, SA2의 순으로 CPU시간이 개선되었으며, L이 6(최대 : 9)일때 비용함수가 최소가 되었다.

4. 결 론

본 연구에서는, VLSI 회로의 자동설계의 상위단계합성에서의 스케쥴링과 하드웨어 할당 문제를 효과적으로 변형된 SA 기법에 의하여 최적화 설계를 시도하였으며 이를 실제 DF 설계와 결부하여 문제점들을 검토하였다. 우선 HLS에서의 스케쥴링과 하드웨어 할당 문제를 변형 SA를 적용하여 총괄적으로 접근하는 방법을 제안하였다. 이 변형된 SA 기법을 간단한 미분방정식의 문제에 적용하여, 기개발된 FDS의 방법에 의한 결과와 비교하여 유사한 결과, 즉 관련 하드웨어가 비교적 균등하게 분포되는 최종 스케쥴링 결과를 얻었다. 이때 변형된 SA1, SA2가 SA0에서 얻을수 있는 수준의 해를 보다 빠른 시간에 얻을수 있음을 확인하였다. 본 연구의 후반부에서는 SA 기법을 DF 설계문제에 적용하였다. 먼저, 대칭계수 DF의 파이프라인 구조의 경우, 잠복주기의 값이 변화함에 따라 수행속도 비용과 하드웨어 비용이 변화하면서 최적 잠복주기 값이 존재함이 확인되었다. 이때, 잠복주기가 정해지는 경우 앞의 미방의 예에서처럼 자원 평준화 현상이 일어나게 되었다. 이때 구조의 다양한 변화에 따라 스케쥴링 방법을 바꾸어가며 적용한 결과 모든 경우에 공히 비슷한 PE배치가 이루어졌다. 또한 이들 SA 스케쥴링 기법을 어레이구조의 DF 설계에 적용하는 과정에서, 규칙성 및 자원공유가 용이한 어레이 구조를 택하여 스케쥴링 기법에 의하여 비용을 구해보았으며, 이때 이들 값은 파이프라인 방향과 주기 및 잠복주기에 따라 변하여, 최적설계시에 이들을 고려해야 함을 확인하였다. 이를 통해 다양한 형태의 문제를 SA를 기본으로 하는 알고리즘을 사용하면 손쉽게 해결할 수 있음을 입증하였고 아울러 이의 변형된 방법인 SA1, SA2의 우월성을 확인하였다.

앞으로의 연구로서는 첫째, SA의 적용에서 해의 이동 방식, 쿨링스케쥴 등 효과적인 변형을 통한 개선을 기하며, 둘째, 하드웨어 비용의 계산시에 실질적인 테크놀러지 매핑, 배치배선을 고려한 연결선비용의 계산(제어기 관련 포함) 등 정확한 하드웨어 비용을 구하여 본 알고리즘를 개선하고자 한다.

참 고 문 헌

[1] M. C. Mcfarland, A. C. Parker, and R.

Camposano, "The High-Level Synthesis of Digital Systems," Proceedings of the IEEE, Vol. 78, No. 2, pp. 301-318, Feb. 1990.

[2] D. Gajski et al., High-level synthesis, KAP, 1992.

[3] P. Marwedel, "A new synthesis algorithm for MIMOLA software design," Proc. of the 23rd Design Automation Conf., New York, NY : ACM/IEEE, pp. 271-277, 1986.

[4] N. Park and A. C. Parker, "Sehwa : A Software Package for Synthesis of Pipelines from Behavioral Specifications," IEEE Trans. on CAD, Vol. 7, No. 3, pp. 356-370, March 1988.

[5] P. G. Paulin and J. P. Knight, "Force-Directed Scheduling for the Behavioral Synthesis of ASIC's," IEEE Trans. on CAD, Vol. 8, No. 6, pp. 661-678, Jun. 1989.

[6] A. C. Parker, J. Pizarro, and M. Mlinar, "MAHA : A Program for Datapath Synthesis," Proc. of the 23rd Design Automation Conf., New York, NY: ACM/IEEE, pp. 461-466, 1986.

[7] S. Devadas and A. R. Newton, "Algorithms for Hardware Allocation in Data Path Synthesis," IEEE Trans. on CAD, Vol. 8, No. 7, pp. 768-781, Jul. 1989.

[8] M. Barbacci and P. Marwedel, "Integrated Scheduling and Binding: A Synthesis Approach for Design Space Exploration," Proc. of 26th Design Automation Conf., New York, NY : ACM/IEEE, pp. 68-74, Jun. 1989.

[9] P. J. M. van Laarhoven, E. H. L. Aarts, Simulated Annealing: Theory and Applications, Reidel, Dordrecht, 1987.

[10] P. R. Ruetz et al., "Computer generation of digital filter banks," IEEE T-CAD, Vol. 5, No. 2, pp. 256-265, April 1986.

[11] C. B. Shung et al., "An integrated CAD system for algorithm-specific IC design," IEEE T-CAD, Vol. 10, No. 4, pp. 447-463, April 1991.

[12] H. D. Man et al., "Architecture-driven synthesis techniques for VLSI implementation of DSP algorithms," Proc. IEEE, Vol. 78, No. 2, pp. 319-335, Feb. 1990.

[13] K. S. Hwang et al., "Scheduling and hardware sharing in pipelined data paths," ICCAD proceed., pp. 2427, 1989.

[14] B. S. Haroun, M. I. Elmasry, "Architectural synthesis for DSP silicon compilers," IEEE T-CAD, Vol. 8, No. 4, pp. 431-447, April 1989.

[15] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," Science, Vol. 220, pp. 671-680, May 1983.

[16] D. F. Wong, H. W. Leong, and C. L. Liu, Simulated annealing for VLSI design, Kluwer, Boston, 1988.

[17] D. Mitra, F. Romeo, and A. Sangiovanni - Vincentelli, " Convergence and finite-time behavior of simulated annealing," Advances in Applied Probability, Vol. 18, pp. 747-771, 1986.

[18] B. Hajek, "Cooling Schedule for Optimal Annealing," Math. Oper. Res., Vol. 13, pp. 311-329, 1988.

[19] D. S. Johnson, C. R. Aragon, L. A. Mcgeoch, C. Schevon, "Optimization by simulated annealing : An experimental evaulations; part 1-Graph partioning,"

Operation research, Vol. 37, pp. 865-892, 1989, "part 2-Graph coloring and number partioning," Operation research, Vol. 39, pp. 378-406, 1991.

[20] Y. G. Saab, V. B. Rao, "Combinatorial optimization by stochastic evolution," IEEE T-CAD, Vol. 10, No. 4, pp. 525-535, 1991.

[21] S. Anily, A. Federgruen, "Simulated annealing methods with general acceptance probilities," J. Appl. Prob., Vol. 24, pp, 657-667, 1987.

[22] ISRC 1991 report, 수율제고를 위한 MOS 최적화 설계에 관한 연구.

[23] S. Y. Kung et al., 'VLSI and modern signal processing,' pp. 258-264, 1985.

[24] P. P. Paulin, J. P. Knight, "High-level benchmark results using a global scheduling algorithm," in 'Logic and architecture synthesis for silicon compilers,' pp. 211-228, North-Holland, 1989.

[25] S. Y. Kung, VLSI array processors, PH 1988.

[26] P. Pirsch, 'VLSI implementations for image communications,' ch. 6(high throughput digital filters), Elsevier Science Publishers, 1993.

송 낙 운

1975년 서울대학교 전자공학과 졸업(학사)
1986년 Univ. Texas Austin (Ph.D)
1986년~89년 금성반도체 근무
1989년 홍익대 전자공학과 부교수
관심분야 : VLSI시스템 자동화 설계.

윤 복 식

1980년 서울대학교 산업공학과 졸업(학사)
1982년 서울대학교 산업공학과 대학원 졸업(석사)
1988년 UC, Berkeley Ph.D
1989년~현재 홍익대학교 기초과학과 조교수
관심분야 : CAD