論文96-33A-7-29

하드웨어-소프트웨어 통합 설계를 위한 분할

(Partitioning for Hardware-Software Codesign)

尹 坰 老 *, 朴 東 河 **, 申 鉉 哲 ***

(Kyungro Yun, Dongha Park, and Hyunchul Shin)

요 약

하드웨어와 소프트웨어의 소자들을 혼합하여 효율적으로 원하는 동작을 하는 시스템을 설계하기 위한 하드웨어-소프트웨어 통합 설계의 중요성은 증가하고 있다. 본 논문에서는 하드웨어와 소프트웨어가 혼합된 시스템의 동작 기술을 하드웨어 부분과 소프트웨어 부분으로 분할하는 새로운 하드웨어-소프트웨어 분할 알고리듬을 기술하였다. 분할 알고리듬은 하드웨어 리소스 또는 수행 시간의 제약 조건하에서 주어진 비용 함수를 최소화하고자 하며, 하드웨어 소프트웨어간에 오퍼레이션의 반복적인 이동으로 최적에 가까운 분할을 찾는다. 하드웨어 비용과 수행 시간은 리스트 스케듈링(list scheduling) 방법을 사용하여 추정하였다. 메모리가 하드웨어의 상당 부분을 차지하므로 하드웨어 비용에 메모리 비용을 포함시킨다. 실험 결과는 본 논문의 알고리듬이 효과적임을 보여 준다.

Abstract

Hardware-Software codesign becomes important to effectively satisfy performance goals, because designers can trade-off in the way hardware and software components work together to exhibit a specified behavior. In this paper, a hardware-software partitioning algorithm is presented, in which the system behavioral description containing a mixture of hardware and software components is partitioned into hardware part and software part. The partitioning algorithm tries to minimize the given cost function under constraints on hardware resources or latency. Recursive moving of operations between the hardware and software parts is used to find a near optimum partition and the list scheduling approach is used to estimate the hardware area and latency. Since memory may take substantial portion of the hardware part, memory cost is included in the hardware cost. Experimental results show that our algorithm is effective.

I. 서 론

최근 컴퓨터와 제어 및 통신 기기를 포함하는 대부분의 전자 시스템은 하드웨어와 소프트웨어를 모두 포함한다. 따라서 하드웨어와 소프트웨어가 복합된 복잡한 시스템을 체계적이며 효율적으로 설계하기 위한 하드웨어-소프트웨어 통합 설계의 중요성이 부각되고 있다. 하드웨어-소프트웨어 통합 설계의 목표는 디지탈 시스템을 주문형 반도체(ASIC), 표준 부품 또는 FPGA등에서 동작하는 하드웨어 부분과 CPU에서 동작하는 소프트웨어 부분으로 설계할 때, 각각의 구성 부분의 특성과 상호작용, 그리고 trade-off 등을 고려하여 원하는 기능의 설계를 주어진 제약조건과 구현 기술 하에서 최소의 비용으로 구현하는 것이다.

광범위한 시스템 구조와 설계 목표 때문에 하드웨어-소프트웨어 통합 설계 문제는 다양한 형태를 갖는다[1]

* 正會員, 三星電子株式會社
(Samsung Electronics Co., LTD.)
**準會員, ***正會員, 漢陽大學校 電子工學科
(Dept. of Elec. Eng., Hanyang Univ.)
※ 이 논문은 한국 과학 재단 특정연구(과제번호 94-0100-02-02-3)에 의해 연구되었음.
接受日字: 1995年8月10日, 수정완료일: 1996年5月13日

[14]. 하드웨어-소프트웨어 통합 설계의 대표적인 응용 예로는 마이크로 프로세서 설계, 디지탈 신호 처리용 IC 설계, 그리고 embedded system설계 등이 있다[13]. 전체적인 하드웨어-소프트웨어 통합 설계의 흐름을 그림 1에 보였다.

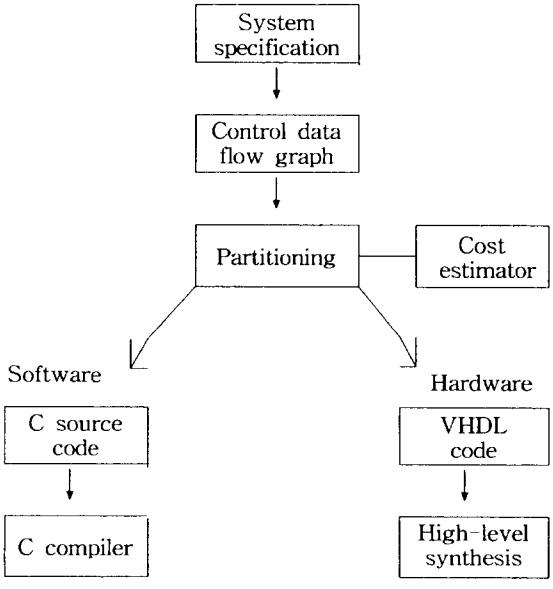


그림 1. 하드웨어-소프트웨어 통합 설계의 전체 흐름
Fig. 1. An example flow of the hardware-software codesign.

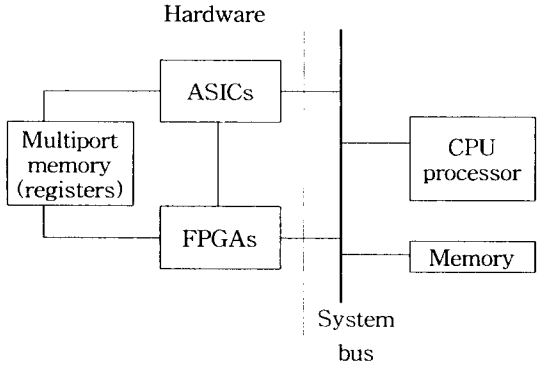


그림 2. 통합 설계의 기본 구조
Fig. 2. The target architecture of our codesign.

시스템 사양은 C 언어나 VHDL등으로 기술될 수 있다. 시스템 사양은 CDFG(control data flow graph)로 변환되고, 각 오퍼레이션(operation)은 하드웨어 또는 소프트웨어로 분할된다. 하드웨어 부분은 VHDL로 기술되고 ASIC이나 FPGA등으로 합성된다. 소프트웨어 부분은 C언어로 기술되고 주어진 마이크로 프로세서에서 실행된다. 본 논문에서는 그림 2와 같은 기본 구조를 사용하여 하드웨어-소프트웨어 분할 알고리듬을 개발하였다. 그림에 보인 바와 같이, 하드웨어 부분과 소프트웨어 부분간의 통신은 시스템 버스를 사용하여 수행된다.

통합 설계에서 하드웨어-소프트웨어 분할은 시스템의 사양으로부터 하드웨어와 소프트웨어 부분을 나누어주는 것이다. 어떤 기능을 하드웨어로 구현하게 되면 하드웨어 비용은 증가하지만 병렬 처리 등의 이점으로 수행 시간을 빠르게 할 수 있다. 반면 소프트웨어로 구현할 경우 대량 생산되는 고성능의 마이크로 프로세서에서 프로그램을 수행할 수 있으므로 하드웨어 비용은 줄어들지만 오퍼레이션들을 순차적으로 실행해야 하기 때문에 성능이 떨어지게 된다. 따라서 통합 설계에서 하드웨어-소프트웨어 분할 알고리듬은 시스템의 성능, 면적, 지연시간, 그리고 하드웨어와 소프트웨어간의 통신으로 인한 오버헤드(overhead)등을 고려하여 여러 기능들을 하드웨어와 소프트웨어로 적절히 나눔으로써 비용을 최소화할 필요가 있다. 보통 하드웨어-소프트웨어 분할은 상위 레벨(behavioral level)에서 수행되기 때문에 최종적으로 구현한 시스템의 성능에 크게 영향을 미치게 된다. 지금까지 발표된 하드웨어-소프트웨어 분할 논문들은 [2, 6, 7, 11, 15] 등이 있다.

본 논문에서는 하드웨어 면적의 제약조건 또는 수행 시간의 제약 조건하에서 비용 함수를 최소화하도록 시스템을 하드웨어와 소프트웨어 부분으로 분할하기 위한 알고리듬을 기술하였다. 시스템의 수행 시간과 하드웨어의 면적은 하드웨어와 소프트웨어 부분을 동시에 스케줄링하여 추정하였다.

2장에서는 전체적인 하드웨어-소프트웨어 분할 알고리즘을 기술하고, 3장에서는 메모리 비용을 고려한 분할에 대해서 설명한다. 4장에서는 여러 가지 예제에 대한 실험 결과를 보여준다. 그리고 5장에서 결론을 기술한다

II. 하드웨어-소프트웨어 분할 알고리듬

하드웨어-소프트웨어 분할은 테스크(task)나 함수(function), 그리고 기본 블럭(block)등의 대단위

(coarse-grain)로 할 수도 있고, 오퍼레이션 등의 소단위(fine-grain)로 할 수도 있다. 대단위로 분할할 경우, 분할이 간단해지고 소프트웨어 코드가 분산되는 것을 막을 수 있으나, 하드웨어 부분에서 리소스 쉐어링 등을 고려하기 힘든 단점이 있다. 반면 소단위로 분할할 경우, 리소스 쉐어링을 고려하기가 쉬워 하드웨어의 비용을 줄일 수 있지만, 소프트웨어 코드가 잘게 분할되어 분산되는 단점이 있다.[4]

하드웨어-소프트웨어 분할에서 여러 해에 대한 전체 시스템의 성능과 비용을 정확하게 추정하는 것은 매우 중요하다. 왜냐하면 그 추정 결과에 따라 분할 결과가 크게 달라지기 때문이다. 소프트웨어의 경우 사용하는 컴파일러에 따라 코드 최적화에 차이가 많이 나기 때문에 정확히 소프트웨어 성능을 추정하기는 어렵다[7, 16]. 하드웨어의 경우 스케듈링 및 resource allocation 등을 통해 성능과 비용을 추정할 수 있다.

본 논문의 분할 알고리듬은 오퍼레이션 레벨에서 분할을 수행한다. 분할 전에 DFG의 각 노드(하나의 오퍼레이션)를 하드웨어로 구현할 경우의 하드웨어 면적과 수행 시간, 소프트웨어로 구현할 경우의 수행 시간 등의 정보를 입력으로 준다. 그리고 하드웨어-소프트웨어 스케듈링을 통하여 전체 시스템의 성능 및 비용을 계산하여 최적에 가까운 해를 구할 수 있도록 분할을 수행한다.

하드웨어 합성 툴을 통해 합성되는 하드웨어 부분과 컴파일러를 통해 합성되는 소프트웨어 부분이 같은 클럭을 사용한다고 가정한다. 그리고 하드웨어와 소프트웨어간의 통신을 위해서는 시스템 버스를 사용한다고 가정하고 통신에 걸리는 시간을 고려하였다.

1. 리소스 제약 조건하에서의 분할
(Resource constrained partitioning)

하드웨어의 리소스 즉 덧셈기의 갯수, 곱셈기의 갯수 등이 제약조건으로 주어진 경우에는 주어진 제약조건하에서 전체 수행 시간을 최소화하도록 분할한다. 각 기능 블럭(functional unit)의 하드웨어 및 소프트웨어 수행 시간은 사용자가 입력으로 정해 준다.

비용 함수는 식 (1)와 같이 전체 수행 시간으로 주어진다.

$$\text{cost} = \text{Texe} \qquad (1)$$

여기서 Texe : Total execution time. (control steps)

전체 수행 시간은 하드웨어-소프트웨어 리스트 스케듈링을 통하여 추정하였다. 하드웨어 면적은 전체 시스템을 스케듈링한 후, 한 cstep에서 동시에 수행해야 하는 연산의 양에 따라 추정하고, 수행 시간은 스케듈링에 의해 latency가 결정되므로 추정 가능하다. 스케듈링은 기본적으로 리스트 스케듈링 알고리듬(list scheduling algorithm)을 사용한다.[5]

전체 분할 알고리듬을 알고리듬 1에 보였다. 분할 알고리듬은 데이터 플로우 그래프를 입력으로 받는다. 현재 각 노드는 하나의 오퍼레이션이고 각각의 노드는 그 오퍼레이션을 하드웨어나 소프트웨어로 수행할 때의 지연시간 등을 데이터로 갖고 있다. 우선 시작 그룹을 하드웨어나 소프트웨어 중의 하나로 선택한다 (Select_From_Group()). 초기 비용을 계산하고 from 그룹에 속한 각각의 노드를 다른 그룹으로 옮겼을 경우 비용의 증감을 계산하는데 (Update_Gain()) 이것이 각 노드의 이득이 된다. 즉 이득이 양의 값일 경우 비용이 감소하는 경우이고 이득이 음의 값일 경우에는 비용이 증가하는 경우이다. 여기서 비용은 설계하고자 하는 시스템의 전체 수행 시간이 되고, 이것은 하드웨어-소프트웨어 스케듈링으로 계산한다. 어떤 노드를 하드웨어로 구현하면 리소스 제한을 초과할 경우에는 이득은 음의 무한대 값을 갖는다. 즉 소프트웨어로 구현하도록 한다.

본 연구에서는, [7]에서와 같이, 덧셈기는 하드웨어와 소프트웨어에 대하여 각각 수행 시간이 1 cstep(control step)이라고 가정하고, 곱셈기는 하드웨어로는 2 csteps 소프트웨어로는 28 csteps걸린다고 가정하였다. 그리고 통신에는 [11]에서와 같이 2 csteps 걸린다고 가정하였다. 또 통신이 일어날 경우 한 변수씩 차례로 전송하여 버스 contention이 일어나지 않도록 한다. 이것은 하드웨어와 소프트웨어 부분이 통신을 위하여 하나의 버스를 공통으로 사용하기 때문이다.

DFG의 각 노드들 중에서 가장 큰 이득 값을 갖는 노드부터 이득이 threshold 값보다 크면 상대편 그룹으로 옮긴다. 이 때 threshold값은 사용자가 주게 되어 있는데, 이 값을 음수로 정하면 hill climbing 효과를 얻을 수 있다. 왜냐하면 한 노드가 음의 이득을 갖더라도 그 노드를 상대편 그룹으로 옮김으로써 다음의 다른 노드가 양의 이득을 갖게 되어 전체적으로 비용이 감소되는 경우가 있기 때문이다[13]. 이 과정을 반복하

면서 가장 최소의 비용을 갖는 분할을 저장한다 (Save_Partition()). 한 쪽 그룹에서 다른 쪽 그룹으로의 이동이 다 끝났을 때의 비용이 그 전까지의 최소 비용보다 적으면 그룹을 바꾸어(Change_From_Group()) 위의 과정을 반복하고 줄어들지 않으면 분할이 끝나게 된다.

Algorithm 1. HW-SW partitioning algorithm.

```
HW_SW_Partition(DFG)
/* DFG = G(V, E) : Data flow graph */
{
   from = Select_From_Group(); /* either HW or
          SW */
   min_cost = Calc_Cost(DFG); /* calculate
              current cost */
   cur_cost = ∞;
   while ( min_cost < cur_cost ) {
      Update_Gain(from); /* find gain of moving
                  each operation from "from"
                  group to the other group */
         cost = cur_cost = min_cost;
         while ((NODE = Largest_Gain_Node
                (from)) is not empty ) {
            if (NODE's gain < THRESHOLD)
               break;
            Move_Node(NODE);
            cost = cost - NODE's gain;
            if ( cost < min_cost ) {
               min_cost = cost;
               Save_Partition();
            }
            Update_Gain(from);
         } end while;
         from = Change_From_Group(from);
   } end while;
}
```

2. 수행 시간 제약 조건하에서의 분할
(Latency constrained partitioning)

설계하고자 하는 시스템의 전체 수행 시간이 제약조건으로 주어진 경우에는 주어진 제약 조건하에서 하드웨어 면적을 최소화하도록 분할한다. 분할 알고리듬의 대부분은 앞의 2.1절의 알고리듬 1과 같으나, 비용 함수와 스케듈링 방법이 다르다.

현재의 분할에 대해 스케듈링을 한 뒤, 각 기능 블럭의 갯수에 그 면적을 곱하여 더한 값이 사용하는 하드웨어의 면적이고, 비용 함수는 이 면적과 전체 수행 시간 제약조건 위반에 대한 함수로 정의한다. 이 때의 스케듈링은 수행 시간 조건하에서의 리스트 스케듈링을 이용한다.

면적 계산을 수식으로 표현하면 식 (2)와 같다.

$$area = \sum_i N_i \times A_i \tag{2}$$

여기서 N_i는 type i인 기능 블럭의 갯수,
A_i는 type i인 기능 블럭의 면적이다.

주어진 분할의 비용은 식 (3)과 같이 계산한다.

$$cost = \alpha \cdot area + \beta \cdot Vtime \tag{3}$$

여기서, Vtime = 0, if total delay < latency_constraint
Vtime = | latency_constraint - total delay |, otherwise.

최종적인 분할은 주어진 수행 시간 제약조건을 만족해야 하므로, $\alpha << \beta$ 인 값을 사용한다.

Ⅲ. 메모리 비용을 고려한 분할

하드웨어에서 메모리가 차지하는 면적이 상당히 큰 경우가 많으므로, 메모리 비용을 고려하면 좀 더 정확한 하드웨어 비용을 추정할 수 있다.

본 논문에서는 하드웨어의 메모리는 다중 포트 메모리(multiport memory)를 사용하였다. 다중 포트 메모리 모듈(multiport memory module)은 다수의 포트(read only, write only와 read/write port)를 공유하는 레지스터의 그룹이다. 레지스터는 다른 cstep에 같은 포트를 통하여, 또는 같은 cstep에 다른 포트를 통해 access될 수 있다. 이러한 포트들의 공유로 인해서 버스의 수, multiplexer의 수 및 tri-state buffer의 수를 감소시킬 수 있다. 그리고, 다중 포트 메모리는 random logic에 비해 규칙적인 layout구조를 가지기 때문에 좀 더 작은 칩 면적에 구현이 가능하다. 이러한 장점이 data path synthesis에 다중 포트 메모리를 사용하는 동기가 된다.

기존의 연구들은 하드웨어를 합성할 때의 다중 포트 메모리 모듈의 최적 합성에 관한 것이다. 우리의 관심은 하드웨어와 소프트웨어의 분할이 메모리 크기에 미치는 영향을 연구하여, 성능 사양을 만족하는 범위 내에서 다중 포트 메모리 모듈의 비용이 고려되도록 분할하는데 있다.

다중 포트 메모리의 구조는 MAP[11]과 Liu[8]와

같은 linear topology 구조를 가정하였다.

분할할 때 하드웨어에서 소프트웨어로 또는 역으로 CDFG의 노드를 옮길 때마다 스케듈링하여 전체 수행 시간과 하드웨어 비용을 계산한다. 이때 다중 포트 메모리 모듈 비용을 계산하여 하드웨어 비용에 포함시킨다. 그러나, 필요한 다중 포트 메모리 모듈들의 최적 설계는 ILP문제로 공식화되어 수행 시간이 오래 걸리게 되므로, 비교적 정확하면서도 간단한 계산 방법이 필요하다. 간단한 합성을 통한 비용 추정을 한다면 최적의 합성은 아니지만 효율적인 추정이므로 수행 시간이 오래 걸리지 않는다.

본 논문에서는 후자를 선택하여 분할을 수행하였다. 이 경우 다중 포트 메모리 모듈을 최종 합성하는 방법에 따라 분할시 메모리 비용 추정 방법이 달라져야 한다. 예를 들면 MAP[11]의 경우 먼저 다중 포트 메모리 모듈을 합성한 후 인터커넥션을 합성한다. 그러나 Liu[8]의 경우 먼저 레지스터와 기능 블럭간의 인터커넥션을 결정한 후 다중 포트 메모리 모듈을 합성한다. 그러므로 모듈 합성 방법에 따라 비용 추정 방법이 결정되어야 한다. 본 논문의 경우 MAP과 같은 방식으로 최종 합성한다고 가정하였다.

메모리의 비용은 다중 포트 메모리 모듈들의 비용의 합으로 계산되며, 각각의 메모리 모듈의 비용은 포트 수와 레지스터 수의 함수로 표현된다. 그 밖의 메모리 모듈과 기능 블럭간을 연결하는 MUX와 tri-state buffer등의 연결 회로가 있으나, 본 논문에서는 각각의 다중 포트 메모리 모듈의 포트 수와 레지스터 수만을 고려한다. 메모리 비용은 실제의 다중 포트 메모리 모듈의 physical layout의 크기를 고려하여 실험적으로 정하였다.[9]

전체 다중 포트 메모리 모듈들의 비용은 식 (4)와 같이 계산된다.

$$M\_cost = \sum_{i=1}^{K} (register_i)^{\eta} \cdot (1+\omega \cdot (port_i - 1)) \quad (4)$$

K : 메모리 모듈의 수

$port_i$: i 번째 다중 포트 메모리 모듈의 포트 수

$register_i$: i 번째 다중 포트 메모리 모듈의 레지스터 수

η는 0.68, ω는 0.48로 실험적으로 정하였다.

메모리의 비용을 하드웨어-소프트웨어 분할에서 고려하기 위해서 다음과 같이 전체 비용에 M_cost를 포함시켰다. 리소스 제약 조건하에서의 분할을 위한 비용 함수는 식 (5)와 같다.

$$cost = Texe + \gamma \cdot M\_cost \quad (5)$$

Ⅳ 실험 결과

하드웨어-소프트웨어 분할을 위한 벤취마크 예제는 발표된 것이 없다. 따라서 여러 논문들은 영상 처리나 디지탈 신호 처리 등에 사용되는 알고리듬 등을 예제로 사용하거나 high level synthesis 예제 등을 사용해 왔으며, 사용하는 하드웨어 라이브러리와 합성 툴들이 서로 다르므로 공통의 예제로 결과 비교를 하기는 어렵다.

본 논문에서는 high level synthesis 예제로 많이 쓰이는 elliptical wave filter와 JPEG과 MPEG같은 영상 압축의 기본이 되고 high level synthesis 예제로도 많이 쓰이는 FDCT(Forward Discrete Cosine Transform)알고리듬을 예제로 사용하였다. 예제에서 사용된 각 기능 블럭의 하드웨어 크기와 속도 등은 논문 [10] 과 [4] 에 제시된 값들을 사용하였다.(표1)

예제에 대하여 각각 리소스 제약조건 하에서의 분할과 수행 시간 제약조건 하에서의 분할을 수행하였다. 리소스 제약조건 하에서의 분할에서는 처음에 하드웨어에서 시작하였고 latency constrained 분할에서는 소프트웨어에서 시작하였다.

표 1. 각 기능 블럭에 대한 면적과 지연시간[10]
Table 1. Area and delay of functional units[10].

Functional unit	Bit width	Area(μm^2)	Delay(ns)
adder	16	40,000	15.0 (1 cstep)
subtracter	16	40,000	15.0 (1 cstep)
multiplier	16	58,000	24.4 (2 csteps)

표 2에서 모두 하드웨어로 구현했을 때(All HW sol.)의 전체 지연시간(total delay)은 [12] 에서 계산한 값과 같은 값을 얻었다. 모두 하드웨어로 구현했을 때보다 리소스는 50%정도만 사용하더라도 수행 속도는 약간만 느려짐을 알 수 있다.

표 2. 리소스 제약조건하에서의 분할 결과
Table 2. Resource constrained partitioning results (THRESHOLD = -10).

(a) Elliptical wave filter

	All SW sol.	All HW sol.	HW-SW sol
HW size (μm^2)	0	196000	98000
Total delay (csteps)	250	19	25
Speed Up over all SW	-	-	10.0
Area Reduction over all HW	-	-	0.5
Functional Units		+: 2, *: 2	+: 1, *: 1

(b) Forward DCT

	All SW sol.	All HW sol.	HW-SW sol
HW size (μm^2)	0	294000	98000
Total delay (csteps)	393	15	41
Speed Up over all SW	-	-	9.58
Area Reduction over all HW	-	-	0.36
Functional Units		+: 2, -: 2, *: 2	+: 1, *: 1

실험 결과 하드웨어와 소프트웨어 분할이 여러 가지 설계 대안을 효과적으로 평가(evaluation)할 수 있음이 확인되었다. 예를 들면 FDCT의 경우, 전 과정을 소프트웨어로 수행하면 393 csteps이 소요되고 모두 하드웨어로 수행하면 15 csteps이 소요되며, 분할에서의 trade-off를 이용하면, 59, 41, 27, 17 csteps 등의 다양한 성능 및 비용을 갖는 설계를 얻을 수 있었다. (표 2 (b), 그림 3 참조)

다음으로 수행 시간 제약조건 하에서의 분할을 FDCT예제에 대하여 실험한 결과를 설명한다. 표 3에서 latency constraints는 사용자가 조건으로 주는 값이고 HW-SW sol.은 분할 결과, 전체 시스템의 수행 시간이다. 분할은 모두 소프트웨어에서 시작하였다. 비용 함수의 식 (3)에서 α와 β값은 각각 1과 100으로 하였다.

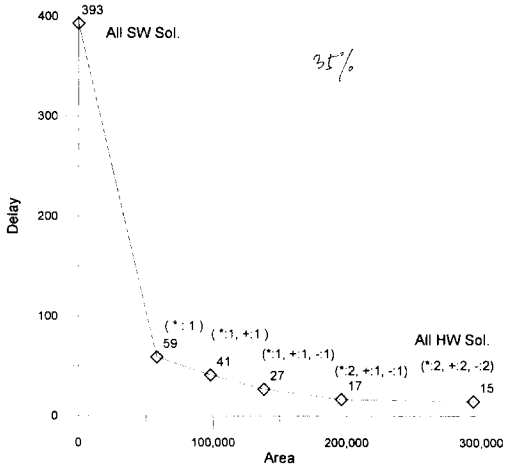


그림 3. 리소스 제약조건 변화에 따른 분할 결과. (FDCT)
Fig. 3. Resource constrained partitioning results of FDCT.

표 3. Forward DCT 예제
(수행시간 제약조건하에서의 분할)
Table 3. Forward DCT example.
(Latency constrained partitioning, THRESHOLD = -15000)

Latency constraints (csteps)	HW-SW sol. (csteps)	Functional Units
100	99	* : 1
50	49	+ : 1, * : 1
30	27	+ : 1, - : 1, * : 2

실험 결과, 본 논문의 분할 알고리듬은 latency constraint를 모두 만족시키며 분할을 수행했음을 알 수 있다. 또한 수행 시간이 빨라질 수록 사용하는 하드웨어도 많아짐을 알 수 있다.

다음은 메모리 비용을 고려한 분할 실험의 결과이다. 비용 함수의 식 (5)에서 γ값은 1로 하였다. 표 4는 elliptical wave filter를 덧셈기와 곱셈기가 각각 하나 그리고 포트가 3개 이하인 다중 포트 메모리 모듈들을 사용하도록 리소스 제약 조건하에서 분할하여, 메모리 비용을 고려하지 않은 경우와 메모리 비용을 고려한 분할을 하여, 수행 시간이 같은 경우를 비교하였다. 예를 들어 cstep = 32인 경우, 메모리를 고려하지 않은

분할은 레지스터 수가 4인 3 포트 모듈과 레지스터 수가 3인 2 포트 모듈이 필요한데 비하여, 분할에서 메모리 비용을 고려하면 레지스터 수가 3인 3 포트 모듈과 레지스터 수가 1인 1 포트 모듈만이 필요하게 되어 M_cost는 8.15에서 5.14로 크게 감소한다.

표 4. Elliptical wave filter 예제 (리소스 제약조건하에서의 분할, +: 1, * : 1, port ≤ 3)

Table 4. Elliptical wave filter (Resource constrained partitioning, +: 1, * : 1, port ≤ 3)

	control step	32	28	26	25
cost = Texe	memory module (#port:#reg.)	3:4 2:3	3:5 2:3	3:5 2:3	3:5 2:3
	M_cost	8.15	8.98	8.98	8.98
cost = Texe + M_cost	memory module (#port:#reg.)	3:3 1:1	3:3 2:2	3:3 3:3	3:4 2:2
	M_cost	5.14	6.51	8.27	7.4
	M_cost 개선 (%)	36.9	27.5	7.9	17.6

V. 결 론

하드웨어-소프트웨어 통합 설계에서 중요한 위치를 차지하는 하드웨어-소프트웨어 분할을 위한 알고리듬을 개발하였으며, 이를 C언어로 구현하여, 몇 가지 잘 알려진 설계에 대하여 실험하였다. 분할을 위한 하드웨어의 면적 및 수행 시간의 추정을 위해서는 리스트 스케듈링을 사용하였다. 보다 정확한 비용의 추정을 위하여 하드웨어 메모리의 비용도 포함하였다.

설계 사양에 따라 하드웨어의 면적 또는 수행 시간의 제약 조건하에서 비용을 최소화하도록 하였으며, 음의 threshold 값을 사용하여 hill climbing을 허용하여 전역적 최적에 가까운 분할을 얻을 수 있도록 하였다. 실험 결과는 본 분할 기법이 여러 설계 대안을 효율적으로 평가·선택하는데 매우 유용함을 보여 주었다.

참 고 문 헌

[1] I. Ahmad and C. Y. Roger Chen, "Post-process for Data Path Synthesis using Multiport Memories", Proc. ICCAD, pp. 276-279, 1991.

[2] M. Chiodo, P. Giusto, Attila Jurecska, H. C. Hsieh, A. S. Vincentelli, L. Lavagno, "Hardware-Software Codesign of Embedded Systems," IEEE Micro, pp. 26-36, Aug, 1994.

[3] G. De Micheli, "Computer-Aided Hardware-Software Codesign," IEEE Micro, pp. 10-16, Aug. 1994.

[4] R. Ernst, J. Henkel and T. Benner, "Hardware-Software Cosynthesis for Microcontrollers", IEEE Design & Test, pp. 64-75, December 1993.

[5] D. Gajski, A. Wu, N. Dutt, and S. Lin, High-level Synthesis: Introduction to chip and system design, Kluwer Academic Publishers, 1993.

[6] R. Gupta and G. De Micheli. "System-level Synthesis Using-Reprogramable Components." In Proceedings of EDAC, pp.2-7, Brussels, Belgium, March 1992.

[7] J. Henkel, Th. Benner, R. Ernst, "Hardware generation and partitioning effects in the COSYMA system," International Workshop on Hardware-Software Codesign, pp. 29-40, October, 1993.

[8] T. Kim and C. L. Liu, "Utilization of Multiport Memories in Data Path Synthesis", Proc. 30th DAC, pp. 298-302, 1993.

[9] K. Nii et al., "A Novel Memory Cell for Multiport RAM on 0.5μm CMOS Sea-of-Gates", *IEEE J. of Solid-State Circuits*, pp. 316-320, Vol. 30, No. 3, March, 1995.

[10] S. Y. Ohm, F. J. Kurdahi, and N. Dutt. "Comprehensive Lower Bound Estimation from Behavioral Descriptions." Proc. of ICCAD, pp. 182-187, 1994.

[11] K. A. Olukotun, R. Helaihel, J. Levitt, R. Ramierz. "A Software-Hardware Cosynthesis Approach to Digital System Simulation." IEEE Micro, pp. 48-58, August 1994.

[12] P. G. Paulin, "Global Scheduling and

Allocation Algorithm in the HAL system," High-Level VLSI Systems, edited by R. Camposano, W. Wolf, Kluwer Academic Publishers, pp. 255-281, 1991.

[13] H. Shin and C. Kim, "A Simple Yet Effective Technique for Partitioning", IEEE Trans. VLSI systems, Vol. 1, No. 3, pp. 380-386, September 1993.

[14] D. Thomas, J. Adams, H. Schmit, "A Model and Methodology for Hardware-Software Codesign.", IEEE Design & Test, pp. 6-15, September 1993.

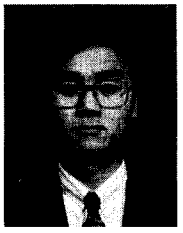
[15] F. Vahid, J. Gong and D. D. Gajski, " A Binary-Constraint Search Algorithm for Minimizing Hardware during Hardware/ Software Partitioning," Proc. of EDAC, pp. 214-219, 1994.

[16] W. Ye, R. Ernst, Th.. Benner, J. Henkel, "Fast Timing Analysis for Hardware-Software Co-Synthesis," Proc. of ICCD, pp. 452-457, 1993.

저 자 소 개

尹 坰 老(正會員)

1969년 9월 5일生. 1995년 8월 한양대학교 대학원 전자공학과 석사 졸업. 현재 삼성전자(주) 반도체마이크로본부 마이크로사업부 DSP팀 주임연구원



朴 東 河(準會員)

1970년 11월 5일生. 1994년 2월 한양대학교 전자공학과 졸업. 1992년 3월 ~ 현재 한양대학교 대학원 전자공학과 석사과정 재학 중. 주관심분야는 VLSI 설계 및 CAD 등임.

申 鉉 哲(正會員) 第 32券 A編 第 4號 參照

현재 한양대학교 전자공학과 부교수