論文98-35C-5-1

하드웨어-소프트웨어 통합설계에서의 새로운 분할 방법

(New Partitioning Techniques in Hardware-Software Codesign)

金 男 勳 *, 申 鉉 哲 **

(Nam-Hoon Kim and Hyun-Chul Shin)

요 약

본 논문에서는 하드웨어와 소프트웨어가 혼합된 시스템의 동작 기술을 하드웨어 부분과 소프트웨어 부분으로 분할하는 새로운 하드웨어-소프트웨어 분할 알고리즘을 기술하였다. 시스템의 성능·면적·지연시간, 그리고 하드웨어와 소프트웨어 간의 통신으로 인한 오버헤드 (overhead) 등을 고려하여, 여러 기능들을 하드웨어와 소프트웨어 부분으로 적절히 나눔으로써 설계사양을 만족시키고, 설계 비용이 최소화 되도록 하였다. 분할의 단위 (partitioning granularity)를 조정함으로써, 상위레벨에서 하위레벨까지의 분할이 모두 가능하게 했다. 또한 분할에서 소모전력과 메모리 비용을 고려하였으며, 하드웨어의 파이프라이닝 (pipelining) 등을 고려하여 우수한 분할 결과를 얻을 수 있도록 하였다. 또한, 실시간 처리되는 DSP용 구조에 맞는 데이터 처리속도 (throughput) 제한 조건하에서의 분할을 수행하였다. 이는 일반적인 혼합 시스템 설계는 물론, 저전력 시스템이나 DSP용 시스템에 응용될 수 있는 분할 알고리즘으로, 이는 다양한 설계 대안 및 구조를 평가하는데 유용하다.

Abstrac

In this paper, a new hardware-software partitioning algorithm is presented, in which the system behavioral description containing a mixture of hardware and software components is partitioned into the hardware part and the software part. In this research, new techniques to optimally partition a mixed system under certain specified constraints such as performance, area, and delay, have been developed. During the partitioning process, the overhead due to the communication between the hardware and software parts are considered. New features have been added to adjust the hierarchical level of partitioning. Power consumption, memory cost, and the effect of pipelining can also be considered during partitioning. Another new feature is the ability to partition a DSP system under throughput constraints. This feature is important for real time processing. The developed partitioning system can also be used to evaluate various design alternatives and architectures.

* 正會員, 三聖電子 半導體研究所

(CAE, Semiconductor R&D Center, Samsung Elect.)

** 正會員, 漢陽大學校 電子工學科

(Detp. of Electronics Eng., Hanyang. Univ.)

※ 본 연구는 한국과학재단의 지원을 받았음. (과제번호 94-0100-02-02-3)

接受日字:1998年2月9日, 수정완료일:1998年4月21日

Ⅰ. 서 론

하드웨어-소프트웨어 통합설계 (hardware-software codesign)에서의 분할 (partition)은 상위 단계의 시스템 동작기술을 하드웨어와 소프트웨어 부분으로 나누는 것이다. 시스템의 모든 기능을 소프트웨어로 구현할 경우 대량생산되는 고성능의 마이크로프로세서에서 프로그램을 수행할 수 있으므로 비용이 적게

들지만, 오퍼레이션들을 순차적으로 실행해야하기 때문에 성능이 떨어지게 된다. 반면에, 하드웨어로 모든 기능을 구현할 경우에는 병렬처리가 가능하여 수행시간을 빠르게 할 수 있어서 성능 요구조건을 만족시키기 쉬우나, 하나 혹은 그 이상의 ASIC 또는 FPGA 등이 필요하게 되어 비용이 많이 드는 단점이 있다. 따라서 통합설계를 이용하여 시스템을 하드웨어와 소프트웨어의 복합적인 요소들로 구현할 경우 하드웨어의 성능상의 장점과 소프트웨어의 비용면에서의 장점이 고려된, 적절한 성능과 비용 제약 조건을 만족하는 효율적인 시스템을 구현할 수 있다. 보통 하드웨어-소프트웨어 분할은 행위 단계 (behavioral level)에서 수행되기 때문에 논리설계 등의 다른 설계과정에 비하여, 최종적으로 구현한 시스템의 성능에 크게 영향을 미치게 된다. 그러나, 상위 단계에서 하위 단계 설계를 추정하여 최적화 되도록 분할하기가 어렵기 때문에 이에 대한 연구는 많이 이루어지지 않았고, 많은 통합설계 시스템들이 전문가에 의한 수동 분할에 의존하고 있다. 최근들어서 활발한 연구가 이루어지고 있으며, 일부 분야의 시스템에서는 자동화되어 사용되어지는 추세이다. 현재까지 발표된 하드웨어-소프트웨어 분할 방법들을 간단히 정리하면 다음과 같다.

Vulcan 시스템 [1] 은 HardwareC 언어로 기술되어 있는 시스템 사양으로부터 CDFG를 추출하고 greedy 알고리즘을 사용하여 성능에 별 영향을 미치지 않는 노드를 소프트웨어로 옮겨 실행하는 하드웨어 지향적 접근 방식이다. COSYMA 시스템 [2] 은 시뮬레이티드 어닐링 (simulated annealing) 방법을 이용하여 시스템 성능의 bottleneck이 되는 부분을 찾아내어 하드웨어로 구현하여 분할을 수행하여 시스템의 전체 수행시간을 최소화 시키는 소프트웨어 지향적 접근 방식이다. [3, 4] 에서는 디지탈 시스템의 시뮬레이션과 합성을 위한 통합 환경인 Ptolemy를 개발하였다. Ptolemy는 전체 시스템을 통일되고 일반적인 언어로 기술하는 대신 각 서브시스템에 적합한 표현 방식을 사용하고 이질적인 표현 방식들 간의 통일된 인터페이스를 제공하는 독특한 방법을 사용한다. [5] 에서는 메모리 최적화를 위한 하드웨어-소프트웨어 분할 알고리즘을 제안했다. 하드웨어-소프트웨어 분할로 인한 통신 메모리 크기 (communication memory size)에 미치는 영향을 연구하여, 성능 사양을 만족하는 범위 내에서 다중 포트 메모리 모듈 (multiport memory module)이 최적화 되도록 분할을 수행하였다. [6] 에서는 하위레벨 (fine-grain level)에서 분할할 때, 기본 블록이 많은 경우 생길 수 있는 수행시간 (run-time)의 증가를 줄이기 위해서 분할 단위를 조절함으로써, 상위레벨 (coarse-grain level)과 하위레벨에서의 분할의 장점을 이용한 알고리즘을 제안했다. [7] 에서는 주어진 시스템 레벨의 사양 (specification)에서 소프트웨어와 하드웨어 component를 선택하고, 선택된 component들을 데이터 처리속도 제한조건을 만족시키면서 최소의 하드웨어 비용이 들도록 분할하고, 파이프라인 (pipeline)하는 알고리즘을 제안하였다.

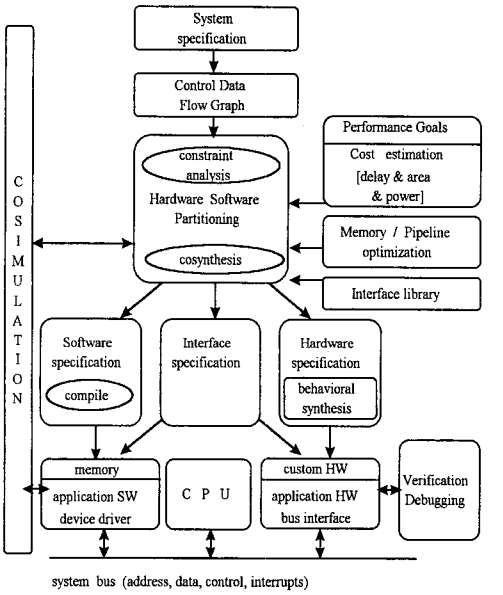


그림 1. 하드웨어-소프트웨어 통합설계의 전체 흐름도

Fig. 1. A typical overflow diagram of hardware-software codesign.

전체적인 하드웨어-소프트웨어 통합설계의 흐름은 그림 1과 같다. 시스템 사양은 C 나 C++, HardwareC, VHDL, Verilog, SpecChart 등의 기술언어로 기술될 수 있다. 시스템 사양은 그 시스템의 특성 및 사양이 적절히 기술된 Control Data Flow Graph (CDFG)나 Finite State Machine (FSM)으로 변환된 후, 각 오퍼레이션은 수행시간·면적·소모 전력·메모리 등의 목적함수에 따라 하드웨어 또는 소프트웨어로 분할되어, 하드웨어 부분은 ASIC이나 FPGA등으로 합성되며, 소프트웨어 부분은 주어진 마이크로프

로세서에서 실행된다. 통합-시뮬레이션 (co-simulation)은 혼합된 하드웨어-소프트웨어 시스템의 동작 모형을 이용하여 설계 과정의 여러 단계에서 각각의 기능들을 검증하는데 사용된다.

Ⅱ. 하드웨어-소프트웨어 분할 방법

분할에는 다양한 종류가 있다. 먼저, mincut을 목적으로 하는 분할이 있으며, 속도나 소모전력 등의 시스템의 성능을 최적화하기 위한 performance-driven 분할이 있다. 분할 과정에서, 시스템 속도의 경우에는 스케쥴과 같은 추정을 통해서 성능을 최적화할 수 있으며, 외부로 나가는 부하가 큰 신호들을 고려하여 분할하면 시스템의 소모 전력을 줄일 수도 있다. 하드웨어-소프트웨어 분할에서는 분할 후의 합성과정까지도 고려해야 한다. 하드웨어-소프트웨어 분할은 주어진 성능에 대한 제한 조건을 만족하는 범위 안에서 최소의 비용으로 구현할 수 있도록 디지탈 시스템을 하드웨어 부분과 소프트웨어 부분으로 나누는 것이다.

하드웨어-소프트웨어 분할에서 여러 가능한 해에 대한 전체 시스템의 성능과 비용을 정확하게 분석 (analysis)하고, 추정 (estimation)하는 것은 매우 중요하다. 그러나, 하드웨어-소프트웨어 분할이 아직 하드웨어가 설계되기 전에 수행되고, 소프트웨어 부분은 소프트웨어 컴파일러와 캐쉬, 메모리 구조 등에 따라 최종 합성 결과가 많이 달라지기 때문에 정확한 추정을 하기가 어렵다[9, 10]. 본 논문에서는 기본적으로 리스트 스케쥴링 알고리즘 (list scheduling algorithm)을 사용한다. 스케쥴링할 때 소프트웨어 오퍼레이션들은 순차적으로 실행되어야 하지만 데이터 의존 (data dependency)이 없는 소프트웨어 오퍼레이션과 하드웨어 오퍼레이션은 동시에 실행될 수 있도록 하였다. 하드웨어와 소프트웨어는 공통의 데이타 버스를 사용하므로 통신은 순차적으로 일어나게 된다. 데이터 플로우 그래프의 각 노드의 priority는 그 노드로부터 sink노드까지의 최장거리로 정의하였다.

1. 하드웨어-소프트웨어 분할 단위(partitioning granularity)

하드웨어-소프트웨어 분할은 태스크 (task), 펑션 (function), 기본 블록 (basic block) 등의 상위 단계 (coarse-grain)에서 수행할 경우, 분할이 간단해지고 소프트웨어 코드가 분산되는 것을 막을 수 있다. 그러나, 하드웨어 부분에서 리소스 쉐어링 (resource sharing) 등을 고려하기 힘든 단점이 있다. 오퍼레이션 (operation, statement) 단위의 하위 단계 (fine-grain)에서 분할을 수행할 경우에는 리소스 쉐어링을 고려하기가 용이하여 하드웨어의 비용을 줄여 최적화에는 유리하지만, 소프트웨어 코드가 잘게 분할되어 분산되며, 면적, 통신 시간 (communication time), 그리고 소프트웨어 컴파일 효과 등을 추정하는데 어려움이 있다. 본 논문의 분할 알고리즘은 시스템 분할 단위를 변화시키며 분할을 수행하여 회로가 큰 경우에는 회로의 규모를 줄이면서 빠른 시간내에 분할을 수행하고, 회로가 작은 경우에는 오퍼레이션 단위까지의 세부적인 분할결과를 얻을 수 있다.

1) 하위 단계 분할 (fine-grain partition)

하위 단계에서의 분할에는 리소스 제약 조건하에서의 (resource constrained) 분할과 수행시간 제약 조건하에서의 (latency constrained) 분할 등이 있다.

하드웨어의 리소스가 제약 조건으로 주어진 경우에는 주어진 제약 조건하에서 전체 수행시간을 최소화하도록 분할한다. 각 기능 블록 (functional unit)의 하드웨어 및 소프트웨어 수행시간은 사용자가 입력으로 정해 주며, 전체 수행시간은 하드웨어-소프트웨어 리스트 스케쥴링을 통하여 추정하였다. 비용 함수는 다음과 같다.

$$cost = total\_delay \tag{1}$$

설계하고자 하는 시스템의 전체 수행시간이 제약 조건으로 주어진 경우에는 주어진 제약 조건하에서 하드웨어 면적을 최소화하도록 분할한다. 현재의 분할에 대해 스케쥴링을 한 뒤, 각 기능 블록의 개수에 그 면적을 곱하여 더한 값이 사용하는 하드웨어의 추정 면적이고, 비용 함수는 이 면적과 전체 수행시간 제약 조건 위반에 대한 함수로 정의한다. 이 때의 스케쥴링은 수행시간 조건하에서의 리스트 스케쥴링을 이용한다. 면적 계산을 수식으로 표현하면 식 (2)와 같다.

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

여기서 N_i는 type i인 functional unit의 개수,

A_i는 type i인 functional unit의 면적이다.

주어진 분할의 비용은 식 (3)과 같이 계산한다.

$$cost = \alpha * Violation + \beta * area \tag{3}$$

Violation = *0*, if *total_delay* < *time_constraint*

Violation = *total_delay* - *time_constraint*, otherwise

최종적인 분할은 주어진 수행시간 제약 조건을 만족해야 하므로, $\alpha >> \beta$ 인 값을 사용한다.

전체 분할 알고리즘을 알고리즘 1에 보였다. 분할 알고리즘은 data flow graph (DFG)를 입력으로 받으며, 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 이동하였을 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 리소스 제약 조건하에서의 분할의 경우 식 (1)과 같은 설계하고자 하는 시스템의 전체 수행시간이며, 수행시간 제약 조건하에서의 분할의 경우 식 (3)과 같은 전체 시스템의 면적과 전체 수행시간 제약 조건 위반에 대한 함수이다. 이것은 하드웨어-소프트웨어 스케줄링으로 계산하며, 분할을 수행할 때 하드웨어와 소프트웨어 노드들은 서로 데이터 의존도 (data dependency)가 없는 한 동시에 수행될 수 있다.

알고리즘 1 하드웨어-소프트웨어 분할 알고리즘

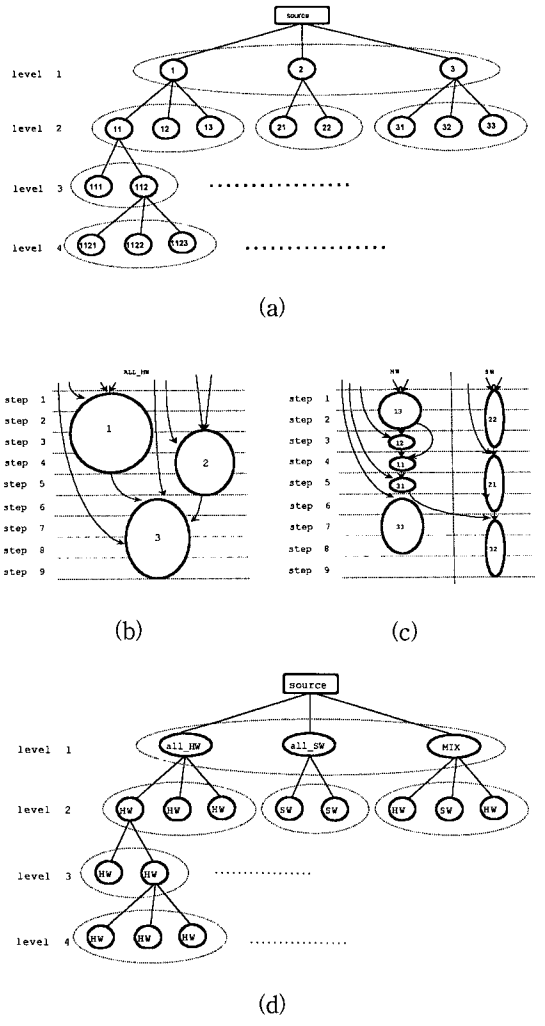
```
HW_SW_Partition(DFG)
/* DFG = G(V, E) : Data flow graph */
{
    from = Select_From_Group(); /* either HW or
    SW */
    min_cost = Calc_Cost(DFG); /* calculate current
    cost */
    cur_cost = ∞;
    while (min_cost < cur_cost) {
        Update_Gain(from); /* find gain of moving
        each 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;
}
```

DFG의 각 노드들 중에서 가장 큰 이득 값을 갖는 노드부터 이득이 threshold 값보다 크면 상대편 그룹으로 옮긴다. 이 때 threshold값은 사용자가 정하도록 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[8]. 이 과정을 반복하면서 가장 최소의 비용을 갖는 분할을 저장한다 (Save_Partition()). 한 쪽 그룹에서 다른 쪽 그룹으로의 이동이 모두 완료되었을 경우의 비용이 이동 전까지의 최소 비용보다 적으면 그룹을 바꾸어 (Change_From_Group()) 위의 과정을 반복하고 비용이 줄어들지 않았으면 분할을 완료하게 된다.

2) 상위 단계 분할 (coarse-grain partition)

상위 단계 분할은 회로의 크기가 큰 대규모 시스템에 쓰인다. 분할하고자 하는 CDFG의 노드수가 많아질수록 분할이 복잡해지고, 많은 연산을 필요로 하기 때문에, 분할의 소요 시간과 성능이 만족스럽지 않을 수 있다. 입력으로 주어지는 계층 단계 (level)에 따라서 모듈 (module) 단위의 계층적인 분할을 하면, 회로의 규모를 줄일 수 있어서 빠른 시간 내에 분할을 수행할 수 있고, 분할 결과를 개선할 수 있다.

분할은 계층구조의 원하는 단계에서 수행할 수 있다. 그림 2에서와 같이 level 1에서 분할하기를 원한다면, level 1 에서의 노드들의 하드웨어 또는 소프트웨어 유형을 결정하게 된다. 이때, level 1 에서의 각 노드의 크기는 상당히 크기 때문에 data dependency 등을 고려하게 되면 소프트웨어로 수행될 수 있는 노드가 없고, 모든 노드들은 하드웨어로 수행되게 된다. level 2에서 분할하기를 원한다면, level 2 노드들의 하드웨어 또는 소프트웨어 유형을 결정하게 된다. 이때 각 노드들의 하드웨어 또는 소프트웨어 수행시간은 그 노드가 라이브러리로 존재하면 그 노드의 수행시간과 면적은 쉽게 알 수 있다. 그러나 라이브러리에 존재하지 않는 계층구조를 갖는 모듈이라면, 각 모듈의 수행시간과 면적은 추정을 해야 한다.



(a)

(b) (c)

(d)

그림 2. 계층적 구조의 시스템의 하드웨어-소프트웨어 분할 과정
(a) 계층적 구조의 CDFG (b) level 1 에서의 분할 (c) level 2 에서의 분할 (d) level 2 에서의 분할 결과

Fig. 2. Hardware-software partitioning process in hierarchical system.
(a) A hierarchical CDFG (b) Partitioning at level 1 (c) Partitioning at level 2 (d) Partitioning result at level 2

추정은 각 노드의 하위 level (level 3)로 이동한 후, 각 노드 내에서 스케쥴링을 하여 각각 하드웨어와 소프트웨어의 수행시간을 추정한다. 이렇게 추정된 각 노드의 하드웨어 소프트웨어 수행시간에 따라 현 level (level 2)에서의 스케쥴링을 수행하여 전체 수행시간을 추정한다.

이 결과에 따라 분할 알고리즘을 수행하며, 이 때의 비용함수는 communication bit 수와 하드웨어 면적, 전체 수행시간이고, 가능한 한 통신을 적게 하면서 수행시간과 면적을 줄일 수 있다. 이 때 각 모듈의 하드웨어 또는 소프트웨어의 수행시간과 면적의 추정은 리스트 스케쥴을 사용하였으며, 전체 수행시간이 제약조건으로 주어진 수행시간을 만족하지 못한다면 각각의 functional unit을 하나 증가시켰을 때의 이득을 구하여, 이득이 가장 높은 functional unit 하나를 증가시켜 스케쥴하여, 하드웨어의 면적을 최적화시키면서 수행시간 제약 조건을 만족하게 했다. Functional unit의 이득 계산식은 다음의 식 (4)와 같고, K의 값은 실험적으로 2로 정하였다.

$$func\_gain = \frac{K * MIN\{red\_time, (total\_delay - time\_constraint)\}}{inc\_area} \quad (4)$$

red_time : functional unit을 증가시켰을 때, 줄어드는 수행시간

inc_area : functional unit을 증가시켰을 때, 늘어나는 하드웨어 면적

total_delay - time_constraint : 줄여야 할 수행시간

*MIN {a, b}*는 *a, b* 중 적은 것을 뜻하며, 줄여야 할 전체 수행시간보다 필요 이상으로 전체 수행시간이 줄어드는 것을 막으면서, 하드웨어 면적을 최적화하기 위한 항이다.

본 연구에서는 상위 level 에서 하위 level 까지 분할의 단위를 조정하며 분할을 수행할 수 있어서, 회로의 크기가 커서 필요이상으로 수행시간이 많이 들거나, 분할의 단위가 너무 커서 필요이상으로 하드웨어를 많이 사용하는 것을 막을 수 있다.

2. 여러 가지 성능을 고려한 분할

1980년대까지 시스템의 특성을 결정 짓는 가장 중요한 요소들 중의 하나는 속도 (speed)였다. 따라서, 구현되는 실리콘 면적을 제한 조건으로 속도를 최대화하기 위해 제안된 여러 기법들로 나타났다. 하지만, 최근 들어, 휴대폰, 노트북, 카세트 등 저전력 설계를 필요로 하는 휴대용 제품에 대한 관심이 많아지면서 휴대용 제품의 시장이 급격하게 성장하고, 또한 높은 클럭 주파수로 동작하는 시스템에 있어 고전력 소모로 인한 신뢰도 문제와 그에 따른 팩키징 (packaging) 비용의 증가가 발생함에 따라, 저전력을 고려한 설계

기술은 VLSI 설계의 모든 측면에서 점점 더 중요해지고 있으며, 소모 전력을 고려한 시스템 설계에 대한 많은 연구가 이루어지고 있다. 이러한 연구들은 회로와 논리 수준의 하위 수준에서의 설계에 대한 연구가 대부분이다. 그러나, 상위 수준에서 다양한 변환 기법들을 시스템 설계에 적용하여 저전력을 고려한다면, 적은 비용과 노력으로 하위 수준의 설계에 비해 훨씬 더 큰 전력 감소 효과를 얻을 수 있으며, 많은 다양한 기법을 적용할 수 있다.

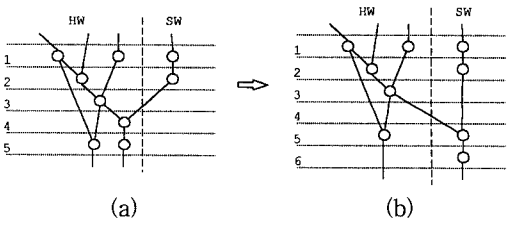
보통 하드웨어-소프트웨어 분할은 행위 단계 (behavioral level)에서 수행된다. 이때 행위 단계에서 각 functional unit과 시스템 버스에서 소모되는 전력을 분할의 비용함수로 고려하여 분할을 수행하게 되면, 저전력 시스템을 구현하는데 알맞은 분할 결과를 얻을 수 있다. 식 (5)의 비용함수에서 첫 번째 항은 각 functional unit이 한 번 수행될 때마다 소모되는 전력을 뜻하며, 두 번째 항은 하드웨어와 소프트웨어 사이에 통신이 일어날 때 시스템 버스에서 소모되는 전력이다.

$$Total\_Power = \sum_{all\_step}\sum_{i} N_i * P_i + \omega * num\_of\_comm\_bit \quad (5)$$

N_i : type i 인 functional unit의 개수
P_i : type i 인 functional unit의 소모 전력
$num\_of\_comm\_bit$: communication bit 수
ω : communication weight

소프트웨어로 수행되는 오퍼레이션들이 많아질수록 일반적으로 소모 전력이 줄어든다. 그림 3에서는 전체 수행시간과 소모 전력과의 trade-off 관계를 보여준다. 노드 한 개의 소모 전력을 1이라하고, communication weight ω를 3이라 했을 때, 소모 전력을 고려하지 않은 분할은 그림 3의 (a)처럼 노드의 소모 전력 7과 시스템 버스에서 소모되는 전력 3을 합한 10이지만, 소모 전력을 고려하면 그림 3의 (b)와 같이 노드의 소모 전력 5와 버스에서 소모되는 전력 3을 합한 8이다. 그러므로, 총 비용함수 계산에 소모 전력 함수 (Total_Power)를 고려한 분할 결과는 같은 하드웨어 면적에 대해 전체 수행시간은 약간 늘어날 수 있지만, 소모 전력면에서는 유리함을 알 수가 있다.

하드웨어에서 메모리가 차지하는 면적이 상당히 큰 경우가 많으므로 [5], 메모리 비용을 고려하면, 좀 더 정확한 하드웨어 비용을 추정할 수 있다.



(a) (b)

	(a)	(b)
Total_Power	10	8
Total_Delay	5	6
HW_area	2	2

그림 3. 소모 전력을 고려한 분할
(a) 소모 전력을 고려하기 전의 분할 (b) 소모 전력을 고려한 분할
Fig. 3. Partition results with/without consideration of power consumption.
(a) Without considering power consumption
(b) With considering power consumption

메모리의 비용은 다중 포트 메모리 (multi-port memory) 모듈들의 비용의 합으로 계산되며, 각각의 메모리 모듈의 비용은 포트 수와 레지스터 수의 함수로 표현된다. 전체 다중 포트 메모리 모듈들의 비용은 식 (6)과 같이 계산되며, 실험적으로 η는 0.68, ω는 0.48로 정하였다.

$$M\_cost = \sum_{i=1}^{K} (register_i)^{\eta} \cdot (1 + \omega \cdot (port_i - 1)) \quad (6)$$

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

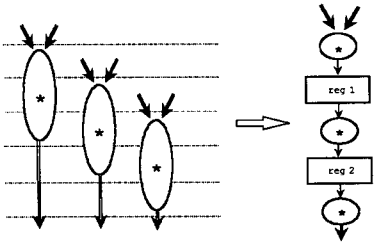


그림 4. 리소스 파이프라인
Fig. 4. Resource Pipeline.

또한 파이프라인 되지 않은 리소스들을 묶어서 파이프라인된 리소스로 교체할 수가 있다. 서로 데이터 의

존도가 없는 곱셈기 두 개 이상이 그림 4와 같은 구조로 되어 있거나, 이러한 구조로 구현이 가능하다면, 이 구조를 파이프라인 된 리소스 하나로 바꿀 수가 있어, 하드웨어 면적을 줄일 수 있다.

소모 전력과 메모리의 비용을 하드웨어-소프트웨어 분할에서 고려하기 위해서 다음과 같이 식(1)과 식(3)의 전체 비용에 소모 전력의 비용 식(5)와 메모리의 비용 식(6)을 포함시켰다. Total_Power는 전체 소모 전력이며, Total_Memory_cost는 전체 메모리 비용이다.

resource constrained partition : ($\alpha > \beta$)

$$cost = \alpha * Total\_delay + \beta * Total\_Power + Total\_Memory\_cost \quad (7)$$

latency constrained partition : ($\alpha >> \beta > \gamma$)

$$cost = \alpha * Violation + \beta * area + \gamma * Total\_Power + Total\_Memory\_cost \quad (8)$$

Violation = 0, if total delay < time_constraint

Violation = time_constraint - Total_delay, otherwise

Ⅲ. DSP용 구조를 위한 분할

최근 전자 시스템의 상당 부분이 영상이나 음성 등 실시간 신호를 처리하기 위한 DSP 용 구조를 가지고 있다. 이러한 DSP 용 구조에서는 일정한 속도(throughput)로 데이터가 연속적으로 입력으로 들어가고, 또 출력으로 나오기 때문에 입력 데이터 하나가 들어가서 출력으로 나온 후, 다음 입력이 들어가야 하는 순차적인 (sequential) CDFG와는 다르게 기술된다. 즉, 그림 5에서와 같이 동시에 동작하는 각 stage 단위로 파이프라인 (pipeline)되어 있는 CDFG로 기술된다. 각각의 stage들은 순차적으로 동작하며, 각 stage에서의 수행결과는 버퍼에 저장되어, 다음 stage의 입력으로 사용된다.

각각의 stage에는 데이터 처리속도가 제한조건으로 주어진다. 예를 들어 그림 5에서 stage 1에는 8 클럭(clock)마다 데이터가 하나씩 입력으로 들어가며, 네 개의 데이터가 출력으로 나온다. 따라서 데이터 처리속도는 0.125가 된다. stage 2에서는 stage 1에서 출력된 데이터를 입력으로 받아서 8클럭 후에 두 개의 데이터를 출력한다. stage 3에서는 stage 2의 결과를 받아서 한 개의 데이터를 출력한다. stage 2와 3의 데이터 처리속도는 각각 0.5와 0.25이다. 이와 같이 일반적인 구조에서는 전체 수행시간 (latency)이 제한조건이었던 반면에, DSP용 구조의 경우 데이터 처리속도가 제한 조건이 되는 것이 큰 차이점이다.

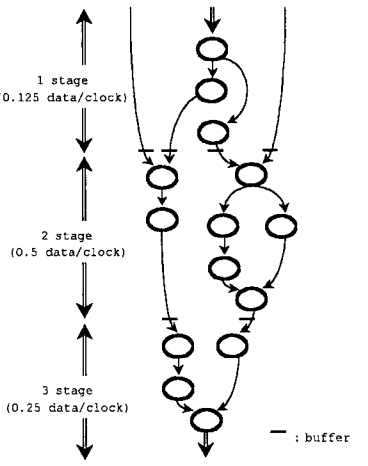
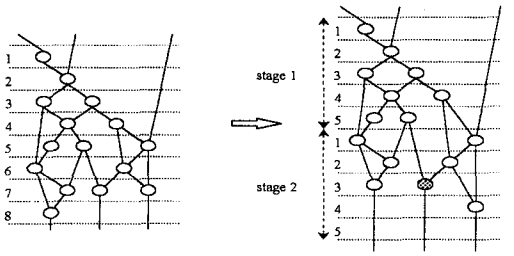


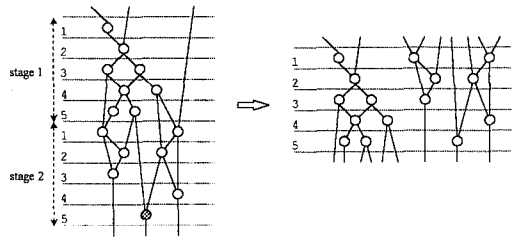
그림 5. DSP용 구조를 가진 CDFG
Fig. 5. CDFG with DSP structure.

데이터 처리속도 제한 조건하에서의 분할 알고리즘을 알고리즘 2에 기술하였다. 먼저 각 stage에 대해서 데이터 처리속도 제한 조건을 만족시키는지를 As Soon As Possible (ASAP) 스케쥴을 통해 알아본다. 데이터 처리속도 제한을 만족시킨다면, 식 (8)의 비용함수로 하드웨어의 면적을 최적화 시킬 수 있는 수행시간 제약 조건하에서의 리스트 스케쥴링을 수행한다.

이 때, 그 stage가 데이터 처리속도 제한을 만족시킬 수 없다면, 그림 6과 같이 stage를 확장하여 스케쥴을 하게 된다. 예를 들어 5 클럭 (clock) 마다 새로운 데이터가 들어 온다고 하면 ASAP 스케쥴을 통해 5 클럭 내에 구현을 할 수 있는 지의 여부를 알 수가 있다. 만약 5 클럭 내에 구현이 불가능하다면, 하나로 구성된 순차적인 CDFG를 n 개의 stage로 확장을 해서 데이터 처리속도를 맞춰 줄 수가 있다. 먼저 데이터 처리속도 제약 조건을 5n 클럭으로 늘려서 스케쥴(extended_list_schedule)을 한 뒤, 각 stage의 node들을 이동시켜가며, 하드웨어 쉐어링 (sharing)을 증가시켜 하드웨어의 개수를 줄인다 (optimized_schedule). 또한, 이 때 하드웨어-소프트웨어 분할은 stage단위의 상위 단계 (coarse-grain) 분할을 수행한다.



(a) ASAP schedule (b) Extended_list_schedule



(c) Optimized_schedule (d) Devided stages

그림 6. 데이터 처리속도 제약 조건하에서의 스케쥴
Fig. 6. Scheduling under data-rate constraints.

알고리즘 2 데이터 처리속도 제한 조건하에서의 분할 알고리즘

```
throughput_constrained_Partition (DFG)
/* throughput constraint(dr_time) */
{
    input_data();
    for (each_stage) {
    final_time = ASAP_schedule(); /* As Soon As
    Possible Scheduling */
        if (final_time <= dr_time)
            latency_constraint_list_schedule(dr_time);
        else {
            extended_latency_constraint_list_schedule
            (dr_time);
                    /* latency constraint를 (n*
                    dr_time)으로 확장하여 스케쥴
                    */
            optimized_schedule();
                /* node movement to obtain
                optimized result */
        }
    }
    HW_SW_partition(DFG);
}
```

Ⅳ. 실험 결과

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

본 연구에서는 high level synthesis 예제로 쓰이는 differential equation과 elliptical wave filter 그리고 JPEG과 MPEG같은 영상 압축의 기본이 되고 high level synthesis 예제로도 많이 쓰이는 FDCT 알고리즘과 Digital VCR 설계의 일부분을 예제로 사용하였다. 예제에서 사용된 각 function unit의 하드웨어 크기와 속도 등은 논문 [2, 11] 에 나와 있는 라이브러리 값들을 참조하였다.

실험 예제에 대하여 각각 리소스 제약 조건하에서의 분할과 수행시간 제약 조건하에서의 분할을 수행하였다. 리소스 제약 조건하에서의 분할에서는 알고리즘 1의 Select_From_Group() 에서 하드웨어 또는 소프트웨어로 주어지는 분할의 시작 type을 하드웨어에서 시작하였고 (from = HW), 수행시간 제약 조건하에서의 분할에서는 소프트웨어에서 (from = SW) 시작하였다.

표 1. 각 기능 블럭에 대한 면적과 지연시간[2, 11]
Table 1. Area and delay of each functional unit.

Functional unit	Bit Width	Area (μm^2)	Delay (ns)	# of control steps (= clock 수)
adder	16	40,000	15.0 (1 cstep)	1
subtracter	16	40,000	15.0 (1 cstep)	1
multiplier	16	58,000	24.4 (2 cstep)	28

표 2는 식 (8)의 비용함수를 사용하여 전체 수행시간 제약 조건하에서의 분할을 수행하는 경우 계층 level에 따른 HW-SW partition 결과이며, communication weight ω의 변화에 따른 변화를 보여준다. ω의 값이 클수록 communication overhead가 적어지며, 대단위의 상위레벨 (level 1)에서의 분할일 경우, 분할이 간단해지고, 소단위의 하위레벨 (level 4)로 갈수록 하드웨어 쉐어링을 고려하기가 쉬워 하드웨어의 비용을 줄일 수 있지만, 분할이 복잡해짐을 알 수 있다. 실험적으로 α, β, γ는 각각 100, 3, 2 로 정하였다.

표 2. Coarse-grain partition (Elliptical wave filter)

Table 2. Coarse-grain partition. (Elliptical wave filter)

		level 1	level 2	level 3	level 4
ω = 5	functional units	+:3, *:2 <:1, +:1	+:2, *:2 =:1	+:2, *:2 =:1	+:2, *:2 =:1
	communication bits	0 (all HW)	6	6	6
ω = 3	functional units	+:3, *:2 <:1, +:1	+:2, *:2 =:1	+:2, *:2	+:2, *:2
	communication bits	0 (all HW)	6	12	12

(+ : adder, * : multiplier, < : comparater, = : load)

표 3. 데이터 처리속도 제약 조건하에서의 분할 결과

Table 3. Partitioning results under data-rate constraints.

throughput(n)	stage 1		stage 2	cost
15	SW		<:1, =:1, &:1 ~:1	88.5
10	SW		<:1, =:2, &:1 ~:1	90.5
8	+:1, -:1, <:1 =:2, &:1, ~:1		<:2, =:4, &:1 ~:1	249.5
6	+:2, -:1, <:1 =:2, &:1, ~:1		<:2, =:4, &:2 ~:1	257.5
5	divide stage		<:3, =:4, &:2 ~:1	251.0
	stage 1'	stage 1''		
	+:1, -:1, <:1 =:2, &:1, ~:1	SW		
4	divide stage		divide stage	285.5
	+:2, -:2, <:2 =:3, &:1, ~:1		<:3, =:6, &:2 ~:1	

(- : subtracter, & : logic_and, ~ : logic_not, ^ : logic_xor)

표 3은 데이터 처리속도에 따른 분할 결과이다. 분할된 stage에 입력 데이터가 들어가는 속도가 제약조건인 경우의 분할 결과를 보여준다. 데이터 처리속도 (n clock)가 느릴수록 하드웨어 공유가 많이 됨을 알 수가 있으며, 그 stage가 데이터 처리속도 제한을 만족시킬 수 없다면 (n = 5,4 인 경우), stage를 확장하여 (divide stage) 스케줄을 하게 된다. 사용된 예제는 digital VCR 설계의 일부분인 24/25 modulation의 부분회로이며, 데이터 처리속도를 15에서 4까지 변화시켰을 때의 결과이다. 데이터 처리속도가 충분히 느리면, 모든 블록이 소프트웨어로 수행될 수 있다. 또한, 데이터 처리속도가 빠르면 빠를수록 하드웨어가 더 많이 필요한 것을 알 수 있으며, 지나치게 빨라 진다면, 시스템이 구현 불가능한 상태까지도 갈 수 있다. 여기서 α, β, γ, ω는 실험적으로 각각 100, 3, 2, 3 으로 정하였다.

V. 결 론

본 연구에서는 하드웨어와 소프트웨어가 혼합된 시스템의 동작수준의 기술을 하드웨어 부분과 소프트웨어 부분으로 분할하는 새로운 분할 방법을 개발하였다. 본 논문의 알고리즘은 주어진 시스템의 제약 조건하에서 하드웨어-소프트웨어 구성 부분간의 trade-off를 고려하여 소프트웨어의 구현에 비하여 수행시간을 빠르게 하고, 하드웨어 구현에 비하여 비용을 감소시킬 수 있는 분할을 수행할 수 있다.

대단위의 상위 레벨에서부터 소단위의 하위 레벨까지 분할이 가능하여, 여러 설계에서 우수한 분할 결과를 얻을 수 있었다. 또한, 기능 블록과 시스템 버스등에 소모되는 전력을 고려함으로써, 저전력 소모를 위한 최적화가 가능하도록 하였다. 또한 리소스 파이프라인과 메모리의 비용 등을 고려할 수 있으며, 파이프라인 되지 않은 리소스들을 묶어서 파이프라인 된 리소스 하나로 바꿀 수가 있어, 하드웨어 면적을 줄일 수 있다. 그리고, 성능 사양을 만족하는 범위 내에서 다중 포트 메모리 모듈이 최적화 되도록 분할을 수행하였다. 데이터 처리속도 제한 조건하에서의 분할을 위하여 영상이나 음성 등 실시간 처리되는 DSP용 구조를 위한 분할 알고리즘을 제안하였다.

실험 결과는 본 논문의 분할 알고리즘이 추정 (estimation) 및 평가 (evaluation)에 효과적임을 보여준다.

참 고 문 헌

[1] R. Gupta and G. De Micheli, "System-level Synthesis Using Reprogrammable Component," Proc. of EDAC, pp. 2-7, March 1992.

[2] R. Ernst, J. Henkel, and T. Benner, "Hardware-Software Cosynthesis for Microcontrollers," IEEE Design and Test of Computers, vol. 10, pp. 64-75, Dec. 1993.

[3] J. Buck, S. Ha, E. Lee, and D. Messerschmitt, "Ptolemy: a framework

for simulating and prototyping heterogeneous systems," International Journal of Computer Simulation, vol. 4, pp. 155-182, April 1994.

[4] A. Kalavade and E. A. Lee, "A hardware-software codesign methodology for DSP applications," IEEE Design and Test of Computers, vol. 10, no. 3, pp. 16-28, Sept. 1993.

[5] D. Park and H. Shin, "Partitioning for Minimal Memory in Hardware-Software Codesign," Proc. of ISCAS, pp. 647-650, May 1996.

[6] J. Henkel and R. Ernst, "A Hardware /Software Partitioner using a dynamically determined Granularity," Proc. of DAC, pp. 691-696, 1997.

[7] S. Bakshi and D. D. Gajski, "Hardware/Software Partitioning and Pipelining," Proc. of DAC, pp. 713-716, 1997.

[8] H. Shin and C. Kim, "A Simple Yet Effective Technique for Partitioning," IEEE Trans. on VLSI Systems, vol. 1, no. 3, pp. 380-386, Sept. 1993.

[9] J. Henkel, T. Benner, and R. Ernst, "Hardware Generation and Partitioning Effects in the COSYMA System," IEEE Int'l Workshop on Hardware/Software Codesign, pp. 29-40, 1993.

[10] W. Ye, R. Ernst, T. Benner, and J. Henkel, "Fast Timing Analysis for Hardware-Software Co-synthesis," Proc. of ICCAD, pp. 452-457, 1993.

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

저 자 소 개

金 男 勳(正會員)

1996년 2월 한양대학교 전자공학과 졸업. 1998년 2월 한양대학교 대학원 전자공학과 공학석사. 1998년 1월 ~ 현재 삼성전자 반도체 연구소 CAE 팀 연구원. 주관심 분야는 VLSI CAD, 디지탈 시스템 설계, hardware estimation, high level synthesis 등임

申 鉉 哲(正會員)

1978년 2월 서울대학교 전자공학과 졸업. 1980년 2월 한국과학기술원 전기 및 전자공학과 공학석사. 1987년 미국 U.C Berkeley 전기 및 컴퓨터공학과 공학박사. 1980년 ~ 1983년 금오공과대학 전자공학과 전임강사 조교수. 1983년 ~ 1987년 Fulbright scholarship. 1987년 ~ 1989년 미국 AT&T Bell Laboratories, Murray Hill 연구원. 1989년 9월 ~ 현재 한양대학교 전자공학과 부교수. 관심 분야는 집적회로 설계자동화, 디지탈 신호처리 시스템 설계 등임