論文93-30A-6-7

디지탈 신호처리를 위한 파이프라인 데이타패스 합성 시스템의 설계

(Design of a Pipelined Datapath Synthesis System for Digital Signal Processing)

田 弘 信*, 黃 善 泳*

(Hong Shin Jun and Sun Young Hwang)

要 約

본 논문에서는 DSP 응용분야에 사용할 수 있는 파이프라인 데이타패스 합성 시스템의 설계에 관해 기술한다. 본 시스템은 행위단계의 SFG (Signal Flow Graph)를 스케메틱으로 입력 받아 스케쥴링과 모듈할당 과정을 거쳐 자동적으로 파이프라인 데이타패스를 합성한다. 효과적인 합성을 위하여 연산을 파티션에 균등히 배분하는 것을 목적함수로 반복적 구성 (iterative/constructive) 방식의 스케쥴링 알고리듬과 초기 할당으로부터 연결구조의 비용을 최소화하도록 할당을 개선시키는 반복적 개선 (iterative improvement) 방식의 모듈할당 알고리듬을 개발하였다. 실험에서 기존의 파이프라인 합성 시스템과 데이타패스의 면적을 비교하여 효율적인 합성이 이루어짐을 보인다.

Abstract

In this paper, we describe the design of a pipelined datapath synthesis system for DSP applications. Taking SFG (Signal Flow Graph) in schematic as inputs, the system generates pipelined datapaths automatically through scheduling and module allocation processes. For efficient hardware synthesis, scheduling and module allocation algorithms are proposed. The proposed scheduling algorithm is of iterative/constructive nature, where the measure of equi-distribution of operations to partitions is adopted as the objective function. Module allocation is performed to reduce the interconnection cost from the initial allocation. In the experiment, we compare the results with those of other systems and show the effectiveness of the proposed algorithms.

I. 서론

최근 몇 년 동안 단시간에 효율적인 VLSI 설계를 위한 자동화의 노력이 계속되고 있으며 응용 수준도 점차 높아지고 있다. 이미 논리합성의 수준에서의 설계 자동화는 실제 칩 설계에 적용되고 있으며, 그 윗 수준인 상위수준 합성이 필요성과 함께 최근에 큰 잇슈로 대두되고 있다.[5,10,15]

상위수준 합성은 설계하고자 하는 하드웨어의 동작을 알고리듬 수준에서 기술하고 이로부터 레지스터

*正會員, 西江大學校 電子工學科
(Dept. of EE., Sogang Univ.)
接受日字: 1992年 9月 18日

전송 수준의 설계를 자동적으로 생성하는 과정이며, 스케쥴링과 모듈할당으로 구성된다. 스케쥴링 과정에서는 알고리듬 기술에서 사용된 연산을 하드웨어의 면적, 지연시간, 전력소모 등의 제약조건을 만족하는 범위내에서 최적으로 제어구간에 할당한다. 모듈할당 과정에서 최적의 하드웨어 공유를 위하여 연산을 연산자에, 변수를 레지스터나 메모리에 할당하고 버스나 먹스를 사용하여 연결구조를 형성한다. [2,11]

설계 자동화에서 지원하는 수준이 높을수록 합성기가 탐색해야 할 공간이 증가하여 상위수준 합성기는 매우 넓은 범위의 설계 공간을 탐색해야 한다. 따라서, 알고리듬 설계가 어렵고 설계된 알고리듬이 다양하게 주어질 하드웨어 기술을 지원해야 한다는 부담 때문에 최적의 성능을 발휘할 수 없게 된다. 이러한 문제점의 해결을 위해 상위수준 합성기 설계에 있어서 합성기가 하드웨어의 동작속도나 알고리듬 특성에 무관하게 만능일 수는 없다는 것을 인정하고 지원하는 범위를 구체화하여 문제의 복잡도를 줄이고 제한된 범위내에서 효과적인 합성을 수행하는 아키텍춰 구동 합성방식을 취해야 한다. [1,3]

DSP (Digital Signal Processing)는 speech, audio, image processing, video, radar등 넓은 응용 범위를 가지고 급속한 성장을 하고 있으며, 선형 계산뿐만 아니라 로그나 비트 연산과 같은 비선형 계산을 포함하고 벡터나 행렬과 같은 다차원 신호를 처리하는 경우가 대부분이다. [3] DSP를 위한 상위수준 합성은 전기한 특징을 갖는 DSP 알고리듬 기술로부터 데이타패스를 합성하는 과정으로 DSP 알고리듬은 샘플 또는 프레임을 단위로 주기적으로 수행되므로 이의 합성과정에서 실시간 처리의 개념이 포함되어야 한다. 즉, 한 프레임에 대한 계산이 정해진 프레임 주기동안에 수행될 수 있는 하드웨어를 합성해야 한다. 이와같이 프레임 레이트를 만족하면서 칩의 면적을 최소화 하는 DSP 컴파일러로 Sehwa [12], PISYN [8], HAL [14], SPAID [6], CATHEDRAL [1] 등이 발표되어 있다.

Sehwa와 PISYN은 파이라인 데이타패스 합성 시스템으로 스케쥴링 과정에서 urgency나 mobility를 우선순위 함수로 사용하여 지엽적인 면만이 고려되어 연산자의 수 측면에서 비효율적인 결과를 얻을 수 있고, HAL 시스템은 광역적인 force directed 스케쥴링 알고리듬을 사용하여 연산을 제어구간에 균등히 분배하는 것을 목적으로 반복적 구성 방식의 스케쥴링을 수행하였으나 우선순위 함수인 force에 연산의 스케쥴링 후에 발생하는 분배 변화의 고려가 미흡하다. CATHEDRAL 시스템은 아키텍춰 구동 합성방식을 가진 전형적인 시스템으로, CATHEDRAL-I은 bit-serial 아키텍춰 [4,7]를 지원하고 있으며 CATHEDRAL-II는 bit parallel 구조를 지원하며 CATHEDRAL-III은 계층적인 제어를 가지고 불규칙적이고 recursive 형태의 고속 알고리듬 합성에 적합하고 CATHEDRAL-IV는 현재 개발중으로 규칙적인 어레이 구조를 타겟으로 하고 있다. 파이프라인 구조를 지원하는 CATHEDRAL-II는 ILP (Integer Linear Programming) [16] 기법을 사용하여 스케쥴링을 수행한다. ILP 기법은 최적의 해를 보장하기는 하지만 매우 긴 시간을 필요로 하여 큰 시스템을 설계하거나 상위수준 합성기가 시스템 설계자에게 아이디어를 제공한다는 측면에서 보면 문제가 있다. SPAID 시스템은 N개의 레지스터파일에 저장된 데이타가 각각에 연결된 N개의 버스를 통하여 연산자에 전달되는 구조를 합성한다. 합성된 데이타패스는 비교적 단순하고 규칙성이 있는 장점이 있으나 피연산자의 입력과 계산 결과의 저장을 위해 버스를 통해야 하므로 클럭 싸이클이 길어져 고속 처리에 적합하지 않은 문제점이 있다.

본 논문에서는 상위수준 합성기 개발에 있어서 타겟 아키텍춰를 고정된 DII (Data Initiation Interval) [13]를 갖는 파이프라인 데이타패스로 결정하고, 이의 효과적인 합성을 위하여 연산이 스테이지에 분배된 정도를 목적함수로 정의하고 목적함수의 미분계수를 우선순위 함수로 사용하여 점진적으로 연산을 스테이지에 균등히 배분되도록 하는 스케쥴링 알고리듬과 연결구조로 사용하는 먹스 입력의 수와 랫치의 수를 비용함수로 고려하여 타겟 아키텍춰를 효과적으로 지원하는 모듈할당 알고리듬을 제시한다. 2장에서 타겟 아키텍춰와 설계방식에 대해 설명하고, 3장에서 스케쥴링 알고리듬, 4장에서 모듈할당 알고리듬을 다루고 5장에서 실험결과를 통하여 알고리듬의 효율성을 보이고 6장에서 결론을 맺는다.

Ⅱ. 타겟 아키텍춰와 설계방식

1. 타겟 아키텍춰

DSP는 반복적인 연산이 많고 연속적인 입력 데이타를 처리해야 특성을 가지므로 파이프라인 구조가 적합하다. 본 연구에서 채택한 타겟 아키텍춰는 Sehwa와 PISYN이 채택하고 있는 고정된 DII를 가지는 파이프라인 데이타패스 아키텍춰이다.

그림 1은 5개의 스테이지를 가지고 DII가 2인 파이프라인 구조의 space-time diagram을 보이고 있다. 그림에서 I는 task를 나타내고 F_i는 스테이지 i

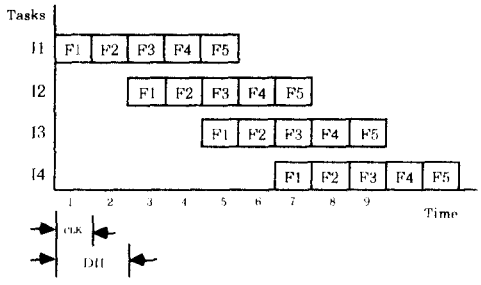


그림 1. DII가 2의 경우 space-time diagram [12]
Fig. 1. Space-time diagram for DII=2.

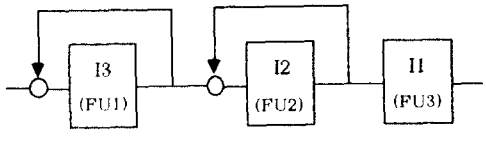


그림 2. 자원이 공유된 파이프라인 데이타패스 [12]
Fig. 2. Hardware sharing for pipelined datapath.

에서 수행하는 function을 나타낸다. 시간 1에서 task I1이 파이프라인에 입력되어 function F1, F2를 수행하고 시간 3에 I1이 F3, F4를 수행할 동안 새로운 task인 I2가 입력되어 F1, F2를 수행한다. 결과적으로 파이프라인이 완전히 동작할때는 시간 5와 6의 상태를 반복하게 된다. 그림에서 {F1, F3, F5}와 {F2,F4}는 각각 연속되어 입력되는 task에 대해 동시에 수행되는 작업을 나타내고 있으며, 이를 파티션 [8] 이라 한다. 파티션의 수는 DII와 같고 파티션 내부에서는 연산의 공유가 일어날 수 없으며 서로 다른 파티션에 존재하는 연산은 공유가 가능하여 그림 2와 같이 자원이 공유된 파이프라인 구조를 가질 수 있다.

파이프라인 아키텍춰의 동작속도는 파이프라인 hazard가 발생하지 않는 이상적인 파이프라인의 경우 DII에 반비례하고 클럭 주파수에 비례한다. 일반적으로 파이프라인 구조의 합성에서 스테이지의 수와 DII를 조정함으로써 합성되는 하드웨어의 면적과 속도의 trade off를 취한다. [9-13] 연산 모듈의 공유를 위한 연결구조로 버스와 먹스를 사용할 수 있으나 버스를 사용할 경우 클럭 주기의 증가가 필연적이고 결과적으로 동작속도의 저하를 가져오게 되어, 타겟 아키텍춰의 연결구조로 버스 보다 지연시간이 적은 먹스를 사용하였다.

2. 설계방식

파이프라인 데이타패스 합성 시스템인 SHSS-DSP (Sogang High-level Synthesis System - DSP)의 구성도는 그림 3과 같다. SFGDL (Signal Flow Graph Description Language)나 스케매틱 에디터로 입력한 SFG를 합성하는 모든 과정을 주관할 SFG View가 받아들여 메뉴 구동 방식으로 파이프라인 데이타패스를 합성하고 합성된 데이타패스는 Datapath View로 출력된다.

각각의 모듈을 합성하는 과정은 스케쥴링과 모듈할당 그리고 제어합성으로 구성된다. 스케쥴링 과정에서는 클럭 주기, FU의 개수, 최대 스테이지 수, DII 등의 제약조건을 만족하도록 SFG의 연산을 특정 스테이지에 할당한다. 모듈할당 과정에서는 공유 가능한 연산을 하나의 FU로 할당한 후 먹스를 사용하여 연결구조를 결정하고 필요한 스테이지 랫치를 삽입한다. SFG View와 Datapath View는 IPC (Inter Process Communication mechanism)를 통하여 연결되고 SFG와 데이타패스를 연결한 정보의 검색이 가능하다. 예를 들면, Datapath View 상에 나타난 ALU가 수행하는 SFG 상의 연산노드를 알아 볼 수 있고 SFG 연산노드를 수행하는 ALU를 검색할 수 있다. SFG View는 SFG를 행위단계의 VHDL을 추출하고 Datapath View는 합성된 데이타패스를 구조단계의 VHDL을 추출하여 VHDL 시뮬레이터로 검증이 가능하게 한다.

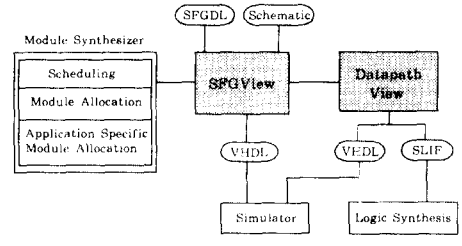


그림 3. SHSS-DSP의 구성도
Fig. 3. Block diagram of SHSS-DSP.

Ⅲ. 스케쥴링 알고리듬

스케쥴링은 SFG와 사용자가 부여한 클럭의 주기,

DII, 스테이지 수의 제약조건을 입력으로 연산을 파이프라인 스테이지에 할당하는 과정이다. 시간적으로 동시에 수행되는 스테이지를 표현하는 파티션 k (0≤ k≤DII-1)는 i modulo-DII = k를 만족하는 스테이지 i (1≤ i ≤max_stage)의 집합으로 정의된다. 서로 다른 파티션에서는 시간적인 중복이 없어서 하드웨어의 공유가 가능하므로 스케쥴링은 SFG의 연산을 DII개의 파티션으로 분할하는 의미를 가진다. 필요한 연산모듈의 수는 각 파티션에서 사용한 연산모듈 수의 최대값이 된다. 제안된 스케쥴링 알고리듬은 필요한 연산모듈의 수가 최소가 되도록 파티션에 할당된 연산의 수의 균형을 목적으로 반복적 구성 방식을 사용한다.

1. 목적함수의 정의

SFG에 ASAP와 ALAP 스케쥴링을 수행하면 연산 노드가 할당될 수 있는 스테이지의 범위를 알 수 있고, force directed 스케쥴링[14] 알고리듬에서 제안한 distribution graph (DG)를 이용하면 N_{op}번이 수행되는 op 타입의 연산이 스테이지 i (1≤i≤ max_stage)에 할당될 확률적인 분포는 식 (1)과 같다.

$$p_{op}(i) = DG_{op}(i) / N_{op},\ i = 1,\cdots,\max\ stage \qquad (1)$$

식 (1)의 스테이지에 할당될 확률을 이용하여 파티션 k (0≤k≤DII-1)에 할당될 확률적인 분포는 식 (2)와 같다. 식 (2)의 $P_{op}(k)$는 파티션 k에 할당될 op 타입의 연산의 확률적인 수를 의미하며, 연산 타입 op의 파티션간의 균등한 배분의 척도를 엔트로피를 이용하여 식 (3)과 같이 정의한다. 엔트로피 함수는 모든 확률값이 같을 경우에 최대값으로 1을 갖고 모든 확률이 한 경우에 집중된 경우에 최소값 0을 갖는다. 식(4)는 스케쥴링 과정에서 최대화해야 할 목적함수로 각 연산타입의 엔트로피에 가중치를 두어 더한 값이다. 가중치 w(op)는 op 타입의 연산의 수 N_{op}로 사용하였다.

$$P_{op}(k) = \sum_{for\ all\ i,\ s.t.\ i\ \mathrm{modulo}-DII\ =\ k} P_{op}(i),\ k = 0,\cdots,DII-1 \qquad (2)$$

$$H(op) = -\sum_{k=0}^{DII-1} P_{op}(k)\ \log\ P_{op}(k) \qquad (3)$$

$$Objective\ Function = \sum H(op)\ w(op) \qquad (4)$$

2. 우선순위 함수

스케쥴링 알고리듬은 식 (4)의 목적함수를 최대화하도록 연산을 스테이지에 할당하는 과정이다. SFG 상의 연산이 스케쥴될 수 있는 범위를 가지는 것을 한 상태로 간주하면, 임의의 노드가 특정 스테이지에 스케쥴되는 것을 포함하여 노드의 범위가 변할 때 상태 변화가 일어난다. 각 노드는 범위가 ASAP에서 ALAP 사이의 한 스테이지로 변하는 상태 변화를 가질 수 있으며, 이때 발생하는 이득은 목적함수를 스테이지로 미분한 것에 비례한다. 제안된 알고리듬에서는 이 미분치를 우선순위 함수로 사용하여 반복적 구성 방식으로 스케쥴링을 수행한다. 우선순위로 사용한 미분치는 식 (5)와 같이 노드 opn이 스케쥴될 수 있는 구간에서 가지는 목적함수를 직선으로 모델링하여 근사화했을 때의 기울기로 사용하였다.

$S_{opn,k}$: 노드 opn이 k 스테이지로 스케쥴된 상태
$OF(S_{opn,k})$: $S_{opn,k}$ 상태의 목적함수
$Mx = MAX(OF(S_{opn,k}))$,
$Mn = MIN(OF(S_{opn,k}))$ for $ASAP_{opn} \le k \le ALAP_{opn}$
Priority Function(opn, k)
$= (Mx-Mn) / (ALAP_{opn} - ASAP_{opn})$ (5)

3. 스테이지 수 제약조건 하의 스케쥴링 알고리듬

스테이지 수 제약조건하의 스케쥴링 알고리듬을 개략적으로 보면 아래와 같다.

단계 1: 모든 노드의 우선순위함수를 계산한다. 스케쥴되지 않는 노드 opn에 대하여 $ASAP_{opn} \le k \le ALAP_{opn}$인 스테이지에서 식 (4)를 사용하여 목적함수를 계산하고, 식 (5)를 사용하여 우선순위 함수를 계산한다.

단계 2: 가장 큰 우선순위함수를 가지는 노드를 스케쥴한다.

단계 3: 스케쥴되지 않은 노드가 있으면 단계 1로 간다.

노드가 ASAP와 ALAP 스케쥴된 초기 상태로부터 모든 노드가 스케쥴될 때까지 각 연산의 우선순위 함수와 최대의 목적함수를 가지는 스테이지를 구하고 (단계 1), 최대의 우선순위 함수를 가지는 노드를 하나 찾아 그 스테이지에 스케쥴한다 (단계 2). 이와같은 과정을 모든 노드가 스케쥴될 때까지 반복한다.

4. 상호 배제의 지원

SFG에 조건분기가 존재하면 서로 다른 경로가 동시에 수행되는 경우는 존재하지 않고 조건에 따라 하나의 경로만 수행된다. 서로 다른 경로에 위치한 연산은 같은 스테이지에 할당되어도 상호 배제에 의해 공유가 가능하다.

상호 배제의 처리는 목적함수의 계산에 사용하는 DG를 계산할 때 아래 식 (6)과 같이 op 타입의 연산의 i 스테이지의 DG, $DG_{op}(i)$는 분기경로의 k의 $DG_{op,k}(i)$의 최대값을 사용한다.

$$DG_{op}(i) = \underset{k \in K}{MAX}(DG_{op,k}(i)) \tag{6}$$

식 (4)의 목적함수는 연산이 파티션에 균등히 배분된 척도를 나타내고 있다. 이 척도는 DG의 합이 일정할 때 연산자에 연산을 고르게 배분하므로써 사용하는 연산자의 수를 최소화할 수 있다는 데에 근간을 두고 있다. 그러나, 상호 배제를 고려할 경우 발생하는 이득은 DG의 합이 감소되어 나타난다. 이 점은 식 (4)의 가중치, w(op)를 다음 식 (7)과 같이 수정함으로써 고려하였다.

$$w(op) = 2 * N_{op} - \sum_{i=1}^{\max stage} DG_{op}(i) \tag{7}$$

식 (7)은 상호 배제가 존재하지 않는 경우 DG의 합과 N_{op}이 같으므로 식 (4)의 가중치와 같다. 상호 배제가 존재하면 N_{op}가 DG의 합보다 작고, 그 차가 상호 배제에 의한 이득이 되어 목적함수에 고려된다.

Ⅳ. 모듈할당 알고리듬

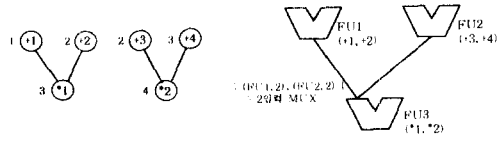
1. 비용함수의 정의

모듈할당 과정에서는 스케줄링의 결과를 입력으로 연산자, 랫치, 연결구조를 형성한다. 파이프라인 데이타패스의 합성에 있어서 스케줄링이 완료되면 필요한 op 타입의 연산자의 수는 $[N_{op}/DII]$ 로 결정된다.[12,13] 또한, 공유 가능한 연산을 공유하여 연산이 연산자에 할당되면 먹스 입력의 수, 랫치의 수는 결정되어 연산자 할당 이후의 먹스와 랫치의 할당은 기계적인 일이다. 따라서, 연산자 할당의 과정에서 먹스와 랫치의 수의 최소화를 고려해야 한다. 본 시스템에서는 식 (8)의 비용함수를 사용하였다. α 와 β 는 각각 먹스 입력과 랫치에 대한 가중치로 라이브러리에 따라 상대적인 면적을 사용한다.

비용함수 = (α * 먹스 입력의 수)
+ (β * 랫치의 수) (8)

연산자에서 스테이지 별로 수행해야 할 공유된 연산들이 필요로 하는 피연산자를 입력으로 받아들이기 위하여 먹스를 사용한다. 연산자 FU1에서 수행할 연산 op1의 피연산자가 op2의 결과로 생성된다면, op2가 수행되는 연산자 FU2로부터 연산자 FU1로 연결이 필요하고 그 사이에는 $stage_{op2}$-$stage_{op1}$ 개의 랫치가 필요하다. 연산 FU1에서 수행해야 할 모든 연산에 대하여 필요한 연결의 (출발점, 랫치의 수)를 원소로 집합을 구성하면 원소의 개수가 필요한 먹스 입력의 수가 된다. FU의 한 피연산자에 대해 피연산자의 출발점이 같고 사이에 필요한 랫치의 수가 같으면 별도의 먹스 입력이 필요없이 연결구조의 공유가 가능하다는 것이다.

그림 4에서 (a)는 DII가 3으로 스케줄된 SFG를 보이고 있고 (b)는 FU1에 (+1,+2), FU2에 (+3,+4) 그리고 FU3에 (*1,*2)가 할당되었을 때, FU3의 왼쪽 피연산자의 입력에 필요한 먹스의 수를 보이고 있다. FU2에 할당된 연산 *1,*2가 피연산자로 연산 +1, +2의 결과가 필요하고 필요한 랫치의 수는 각각 2개이다. 출발점과 랫치의 수를 원소로 집합을 만들면, { (FU1,2), (FU2,2) }로 2개의 입력을 가지는 먹스가 필요하다.

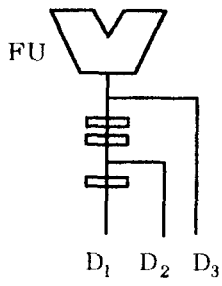


(a) (b)

그림 4. 먹스 입력의 계산 예
(a) 스케줄링된 SFG
(b) 먹스의 사용

Fig. 4. Example of calculation of MUX inputs.
(a) Scheduled SFG .
(b) Number of necessary mux inputs.

랫치의 수는 그림 5와 같은 공유를 고려하여 결정된다. 임의의 연산자 FU에 대하여 출력이 전달되야 하는 모든 목적지 D_j에 사이에 필요한 랫치의 수를 N_{FU,D_j}라 할때, 연산자 FU의 출력을 올바로 다른 모듈에 전달하기 위해 필요한 최소의 랫치 수는 N_{FU,D_j}의 최대값과 같다. FU 결과의 목적지 D_1, D_2, D_3에 필요한 랫치의 수를 각각 3, 2, 0라 하면 그림 5와 같은 구조로 랫치의 공유가 가능하다.



FU -> D_1 : 3 개
FU -> D_2 : 2 개
FU -> D_3 : 0 개

그림 5. 랫치의 공유
Fig. 5. Sharing of latches.

2. 모듈할당 알고리듬

모듈 할당 알고리듬은 반복적 개선 방식을 취하고 있다. 스케쥴링의 결과로 결정된 수만큼의 연산자를 사용하여 임의로 초기 할당을 수행하고 같은 종류의 연산모듈에 대하여 각각에 속한 연산 중에서 다른 연산모듈로 이동이 가능한 연산과 서로 교환이 가능한 연산쌍을 찾는다. 이들 상태 변화 후보중에서 상태 변화가 일어났을때 면적을 예측하여 면적의 감소량이 가장 큰 상태 변화를 수행하여 할당을 개선시킨다. 이 과정을 면적이 감소하지 않을 때까지 반복한다. 알고리듬을 개략적으로 기술하면 아래와 같다.

단계 1 : 초기 할당.
단계 2 : 상태 변화 후보들을 찾는다.
단계 3 : 최소의 비용을 갖는 상태변화와 이득을 계산한다.
단계 4 : 이득이 0 보다 작으면, 상태 변화를 수행하고, 단계 2로 간다.

하나의 상태 변화에서 위치가 변하는 연산의 수를 상태 변화 레벨이라 표현하자. N개의 노드를 가지는 SFG에서 최적의 결과를 생성하려면 N 레벨의 상태 변화 후보를 동시에 고려하여 목적함수의 변화를 조사해야 한다. 그러나, 상태 변화 레벨이 증가함에 따라 상태 변화 후보의 수가 NP로 증가하여 적절한 상태 변화 레벨의 선정이 필요하다. 실험한 결과 상태 변화 레벨을 2로 한 것이 평균적으로 30개의 상태 변화 후보로 10번 이내의 상태 변화를 거쳐 수행 시간과 면적의 감소 측면에서 효율적이었다.

연산자 FU1에 할당된 연산 opn 이 연산자 FU2로 이동하기 위해서는 FU2가 opn을 수행할 수 있어야 하고, FU2가 작업을 수행하는 모든 스테이지에 대해 파티션의 중복이 없어야 한다. 그러나, DII개의 파티션에 균등히 분할된 스케쥴의 결과로는 이와같은 경우가 존재하지 않는다. 따라서, 레벨 2의 상태 변화에 포함되는 연산의 상호 교환을 함께 고려한다. 연산자 FU1에 할당된 연산 op1이 연산자 FU2의 연산 op2와 교환이 가능한 경우는 FU2로 op1이 FU1으로 op2가 수행이 가능하고 op1과 op2를 FU1과 FU2에서 제거했을때 op1이 FU2로 이동 가능하고 op2가 FU1으로 이동 가능한 경우이다.

예를 들어 그림 4의 경우 FU1의 연산 +2와 FU2의 연산 +3은 같은 파티션에 속한 연산이므로 교환이 가능하다. FU2의 왼쪽 피연산자에 필요한 먹스 입력의 수를 계산하면 출발점과 랫치의 수를 원소로 하는 집합이 { (FU1,2) }이므로 먹스가 필요 없게 되어 하나의 먹스 입력을 절약할 수 있다. 이러한 변화를 이득이 발생하지 않을 때까지 반복적으로 수행하여 연산자 할당의 결과를 개선시킨다. 연산자 할당이 완료된 후 먹스를 할당하고 랫치를 사용하여 데이타패스를 구성한다.

Ⅴ. 실험결과

SHSS-DSP는 워크스테이션 SUN IPC에서 UNIX 환경에서 구현하였다. 구현된 시스템의 성능을 평가하기 위해 MCNC 벤치마크 회로인 16 point FIR 필터와 5차 엘립틱 웨이브 필터에 대하여, 그리고 FDCT 모듈의 설계에 대하여 실험한 결과를 보인다.

1. 16 point FIR 필터의 설계

16 point FIR 필터의 SFG는 15개의 덧셈과 8개의 곱셈으로 구성되어 있으며 상위 수준 합성의 벤치마크 중의 하나이다. 표 1은 PISYN과 SHSS-DSP로 FIR 필터를 스테이지의 수를 6으로 고정시키고 DII를 1 부터 6까지 변화시키며 합성한 결과를 보이고 있다.

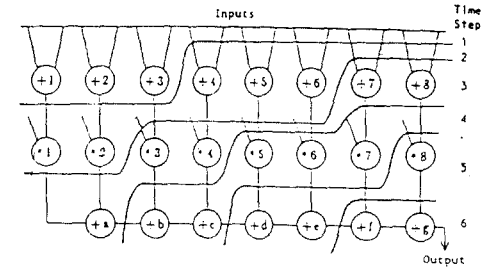
표 1. FIR 필터 합성결과 (# stages = 6)
Table 1. Synthesis results of the FIR filter. (# stages = 6)

(PISYN [8] /SHSS-DSP)

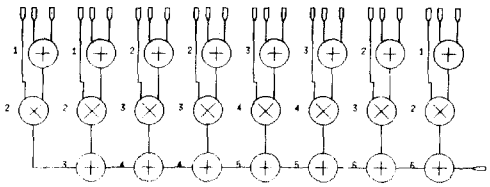
DII / Resource	1	2	3	4	5	6
# x's	8/8	4/4	3/3	2/2	2/2	2/2
# +'s	15/15	8/8	5/5	4/4	4/4	3/3
# MUX inputs	0/0	40/34	42/34	41/33	39/35	37/35
# Registers	57/50	42/46	43/39	50/49	46/46	43/44
CPU time (Sec)	9/0.70	138/0.90	144/0.75	142/0.83	143/0.70	147/0.86

CPU time은 PISYN은 XEROX에서 수행한 것이고 SHSS-DSP는 SUN IPC에서 수행한 것이다. SHSS-DSP는 모든 경우에 대해서 1초 이내에

PISYN과 같은 수의 덧셈기와 곱셈기를 사용하는 스케줄을 생성하였고 먹스 입력의 수와 랫치의 수를 비교하면 연결구조 면에서 전반적으로 7.3 % 효율적인 결과를 보이고 있다.



(a)



(b)

그림 6. FIR 필터의 스케줄링 결과
(# of stage= 6, DII=3)
(a) Sehwa와 PISYN의 스케줄링 [8,12]
(b) SHSS-DSP의 스케줄링

Fig. 6. Scheduling of FIR filter
(# of stages=6, DII=3).
(a) by Sehwa and PISYN,
(b) by SHSS-DSP.

그림 6은 FIR 필터에 대하여 최대 스테이지 수를 6, DII를 3으로 하여 Sehwa, PISYN, SHSS-DSP가 스케줄링한 결과를 보이고 있다. 세 시스템 모두 덧셈기 5개 곱셈기 3개를 사용하는 최적의 스케줄을 생성하였다.

그림 6의 (b)에서 add1와 mult1의 할당된 스테이지의 모양이 add3와 mult3와 같음을 알 수 있다. 이 경우 add1와 add3이 하나의 덧셈기로 할당되고 mult1과 mult3이 하나의 곱셈기로 할당하면 필요한 랫치의 수가 같으므로 먹스 입력의 수와 랫치의 수를 절약할 수 있다. FU의 개수면에서는 SHSS-DSP와 PISYN, Sehwa가 같은 결과를 가지지만 연결구조를 고려하면 SHSS-DSP의 스케줄링 결과가 효과적이다.

2. 5차 엘립틱 웨이브 필터의 설계

그림 7은 5차 엘립틱 웨이브 필터의 SFG를 보이고 있다. 엘립틱 웨이브 필터는 8개의 곱셈과 26개의 덧셈으로 구성되고 상위수준 합성의 벤치마크중의 하나이다. [15]

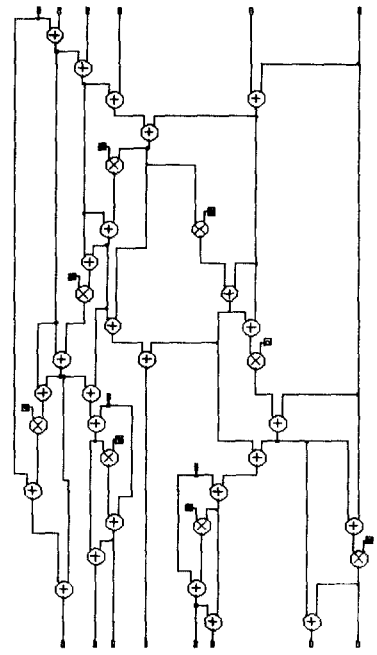


그림 7. 5차 엘립틱 웨이브 필터의 SFG

Fig. 7. SFG of the 5th order elliptic wave filter.

표 2와 표 3은 그림 7의 SFG를 PISYN과 SHSS-DSP가 합성한 결과를 보이고 있다. 표 2는 9개의 파이프라인 스테이지를 사용하여 DII를 1에서 9까지 변화시키면서 합성한 것이고, 표 3은 10개의 파이프라인 스테이지를 사용하여 DII를 2에서 10까지 변화시키면서 합성한 결과이다. 9개의 스테이지를 사용한 경우는 모든 DII에서 같은 연산자를 사용하는 결과를 생성했지만 10개의 스테이지를 사용한 결과에서 DII가 3인 경우 곱셈기와 덧셈기를 1개씩 절약하였고 DII가 8, 10인 경우에 덧셈기를 하나씩 절약한 스케줄링 결과를 생성하였다. 연결구조면에서도 15% 줄어듬을 알 수 있다.

3. FDCT (Fast Discrete Cosine Transform) 모듈의 설계 [10]

표 2. 5차 엘립틱 필터의 합성결과 (# stages = 9)
Table 2. Synthesis results of the 5th elliptic wave filter. (# stages=9)

(PISYN [8] /SHSS-DSP)

DII / Resource	1	2	3	4	5	6	7	8	9
# x's	8/8	4/4	4/4	3/3	4/4	2/2	2/2	2/2	2/2
# +'s	26/26	13/13	9/9	7/7	8/8	5/5	6/6	6/6	4/4
# MUX inputs	0/0	64/44	63/54	62/49	60/53	57/52	61/43	57/45	65/47
# Registers	132/77	69/64	60/58	53/48	54/52	49/48	48/46	47/46	46/45
CPU time (Sec)	26/ 1.81	238/ 3.43	262/ 2.90	265/ 3.00	267/ 1.23	273/ 1.44	271/ 1.67	272/ 1.21	277/ 1.06

표 3. 5차 엘립틱 필터의 합성결과 (# stages = 10)
Table 3. Synthesis results of the 5th elliptic wave filter. (# stages=10)

(PISYN [8] /SHSS-DSP)

DII / Resource	2	3	4	5	6	7	8	9	10
# x's	4/4	4/3	2/2	3/3	2/2	2/2	2/2	2/2	2/2
# +'s	13/13	10/9	7/7	6/6	5/5	5/5	6/5	4/4	4/3
# MUX inputs	66/54	63/60	66/53	58/50	62/53	55/44	57/46	50/44	52/43
# Registers	72/68	68/56	59/59	60/54	55/52	54/47	52/48	51/46	51/46
CPU time (Sec)	382/ 4.03	262/ 3.34	377/ 4.00	348/ 6.43	366/ 3.92	351/ 5.27	342/ 4.79	339/ 4.69	345/ 5.49

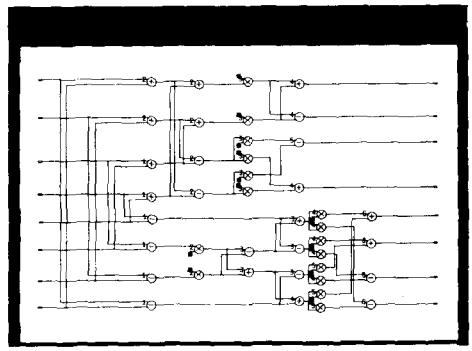


그림 8. FDCT 커널의 SFG [10]
Fig. 8. SFG of FDCT kernel.

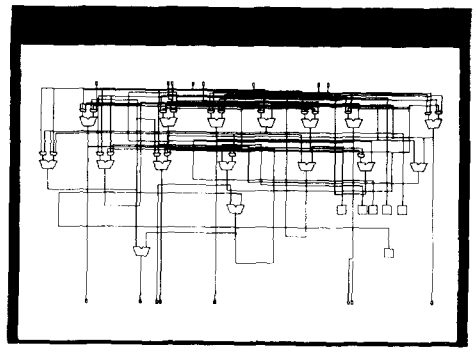


그림 9. FDCT 커널의 합성된 데이타패스
Fig. 9. Synthesized datapath of FDCT kernel.

그림 8은 FDCT 커널의 SFG를 보이고 있다. 합성에 사용한 라이브러리는 덧셈기 20nS, 뺄셈기 20nS, 곱셈기 80nS의 지연시간을 가지며, 클럭 싸이클을 50nS, 스테이지 수를 6 그리고 DII가 3인 제약조건으로 합성한 결과를 그림 9에 보이고 있다. 스케쥴링에 6.71초와 모듈할당에 3.42초로 합성에 10.13초가 소요되었으며, 5개의 덧셈기와 뺄셈기와 6개의 곱셈기 그리고 6개의 ROM이 사용하는 결과를 얻어 참고문헌 [10] 의 결과와 동일하다. 연결구조에 사용한 먹스 입력의 수는 53개이며 랫치의 수는 30개를 사용하였다.

Ⅵ. 결론

본 논문에서는 SFG를 입력으로 파이프라인 데이타패스를 자동적으로 합성하는 SHSS-DSP에 대하여 설명하였다. 파이프라인 데이타패스의 효과적인 합성을 위하여 연산을 DII개의 파티션에 균등히 배분하는 것을 목적으로 반복적 구성 방식의 스케쥴링 알고리듬을 제안하였으며, 초기 할당으로부터 먹스 입력의 수와 랫치의 수를 줄여 나가는 반복적 개선 방식의 모듈할당 알고리듬을 제안하였다. 실험에서 기존의 파이프라인 데이타패스 합성기와 비교하여 본 시스템이 연산자와 연결구조 면에서 효율적임을 보였다.

현재 스케쥴링 과정에서 모듈할당시에 연결구조를 고려하는 방안에 대해 연구중이고, 실제 DSP에서 많이 사용되는 multi-port 메모리, 레지스터 파일과 RAM 모듈등을 지원하도록 시스템의 응용범위를 확장할 계획이다.

參 考 文 獻

[1] J. Allen, F. Catthoor, "Architecture driven synthesis technique for VLSI implementation of DSP algorithms," *IEEE Proceedings*, vol. 78, no. 2, pp. 319-335, Feb. 1990.

[2] R. Camposano, W. Wolf, *High-level VLSI Synthesis*, Kluwer Academic Publishers, 1991.

[3] F. Catthoor, H. J. De Man, "Application specific architectural methodologies for high throughput digital signal and image processing," *IEEE Trans. ASSP.*, vol. 38, no. 2, pp. 339-349, Feb. 1990.

[4] P. Deneyer, D. Renshaw, *VLSI Signal Processing : A Bit Serial Approach*, Addison Wesley Pub: Reading, Mass., 1985.

[5] D. D. Gajski, *Silicon Compilation*, Addison Wesley Pub. : Reading, Mass., 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] R. I. Hartley, J. R. Jasca, "Behavioral to structural translation in a bit-serial silicon compiler," *IEEE Trans. CAD*, vol. 7, no. 8, pp. 877-886, Aug. 1988.

[8] K. Hwang, A. E. Casavant, "Scheduling and hardware sharing in pipelined data paths," in *Proc. ICCAD*, pp. 24-27, nov. 1989.

[9] P. M. Kogge, *The Architecture of Pipelined Computers*, McGraw-Hill, 1982.

[10] D. J. Mallon, P. B. Denyer, "A new approach to pipeline optimisation," in *Proc. EDAC*, pp. 83-88, March 1990.

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

[12] 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.

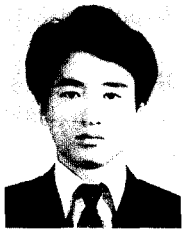
[13] N. Park, *Synthesis of High-Speed Digital Systems*, PhD thesis, University of Southern California, Oct. 1985.

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

[15] G. Saucier, P. M. McLellan, *Logic and Architectural Synthesis for Silicon Compilers*, Elsevier Science Publishers : north-Holland : Amsterdam, 1989.

[16] C. T. Hsung, J. H. Lee, "A formal approach to the scheduling problem in high level synthesis," *IEEE Trans. CAD*, vol. 10, no. 4, April 1991.

著 者 紹 介



田 弘 信(正會員)

1967年 8月 22日生. 1989年 2月 서강대학교 전자공학과 졸업. 1991年 2月 서강대학교 전자공학과 석사과정 졸업. 1991年 3月 부터 서강대학교 전자공학과 박사과정 재학중. 주관심분야는 High-level Synthesis, DSP Architecture, VLSI 설계등임

黃 善 泳(正會員) 第 30卷 A編 第 2號 參照

현재 서강대학교 전자공학과 교수.