論文94-31A-6-18

가변 데이타 입력 간격을 지원하는 파이프라인 구조의 합성

(Synthesis of Pipeline Structures with Variable Data Initiation Intervals)

田 弘 信*, 黃 善 泳*

(Hong Shin Jun and Sun Young Hwang)

要 約

시스템 설계자는 상위수준 합성을 통하여 데이타패스를 얻을 수 있을 뿐만 아니라 설계 공간에서 면적과 속도의 trade-off 정보를 얻을 수 있다. 파이프라인 합성이 데이타 입력간격을 고정한 파이프라인 구조를 목표로 하드웨어를 생성한 기존의 방법에 비하여 본 논문에서는 보다 넓은 설계공간을 제공하는 가변 데이타 입력간격을 가지는 파이프라인 구조를 합성하는 방법을 제안한다.

파이프라인 구조의 상위수준 합성에서 중첩이 발생하지 않는 스테이지를 효과적으로 이용함으로써 하드웨어의 효과적인 사용이 가능하다. 본 논문에서는 가변 데이타 입력간격을 가지는 파이프라인 구조에서 스테이지의 중첩을 결정하였으며, 중첩이 발생하지 않는 스테이지의 정보를 효과적으로 사용하는 스케쥴링과 모듈할당 방법을 제안한다. 실험을 통하여 면적과 속도면에서 효과적인 설계를 생성함을 보인다.

Abstract

Through high level synthesis, designers can obtain the precious information on the area and speed trade-offs as well as synthesized datapaths from behavioral design descriptions. While previous researches were concentrated on the synthesis of pipelined datapaths with fixed DII (Data Initiation Interval) by inserting delay elements where needed, we propose a novel methodology of synthesizing pipeline structures with variable DIIs.

Determining the time-overlapping of pipeline stages with variable DIIs, the proposed algorithm performs scheduling and module allocation using the time-overlapping information. Experimental results show that significant improvement can be achieved both in speed and in area.

Ⅰ. 서론

상위수준 합성은 설계하고자 하는 하드웨어의 동작을 알고리듬 수준에서 기술하고 이로부터 레지스터 전송 수준의 설계를 자동적으로 생성하는 과정으로 스케쥴링과 모듈할당으로 구성된다. 스케쥴링은 수행하는 하드웨어의 면적, 지연시간, 파워 소모 등의 제약조건을 만족하는 범위 내에서 알고리듬 기술에서 사용된 연산을 최적으로 제어구간에 할당하는 과정으로 생성된 레지스터 전송 수준의 하드웨어의 동작속도와 면적 등에 큰 영향을 미친다. 모듈할당 과정에서는 하드웨어의 공유를 최대화하도록 설계기술상의

* 正會員, 西江大學校 電子工學科
(Dept. of Elec. Eng., Sogang Univ.)
接受日字 : 1993年 7月 22日

연산을 연산자에, 변수를 레지스터나 메모리에 할당하고 BUS나 MUX를 사용하여 연결구조를 생성한다.[5]

설계자동화에서 지원하는 수준이 높을수록 합성기가 탐색해야 할 공간이 증가하여, 상위수준 합성과정에서는 매우 넓은 범위의 설계공간의 탐색이 요구된다. 이러한 문제점을 해결하기 위해 대부분의 상위수준 합성기는 합성된 하드웨어의 응용범위를 구체화하여 지원하고 있다.[1] DSP (Digital Signal Processing) 등의 실시간 처리가 요구되는 응용분야에서 하드웨어의 속도향상은 항상 중요한 연구대상이다. 특히 큰 면적상의 overhead 없이 속도향상을 얻을 수 있는 파이프라인 구조의 하드웨어는 널리 사용되고 있으며, 이의 자동생성을 위한 연구도 활발하게 진행되어 왔다. Sehwa[6], PISYN[2], HAL[8] 시스템 등은 스케쥴링과 모듈할당 과정을 통하여 파이프라인 구조의 자동적인 합성을 이룬다. Sehwa와 PISYN은 각각 연산이 스케쥴될 수 있는 범위의 크기에 기반을 둔 urgency와 mobility를 우선 순위 함수로 반복적 구성 (iterative/constructive) 방식으로 스케쥴링을 수행하고, HAL 시스템은 force directed 스케쥴링 알고리듬으로 연산을 제어구간에 균등히 배분하여 하드웨어의 공유를 이룬다. 각각의 시스템들은 스테이지의 시간적인 중첩을 효과적으로 이용한 고유의 스케쥴링과 모듈할당 알고리듬을 가지고 있다.

기존의 파이프라인 구조의 합성은 고정된 데이타 입력간격(DII: Data Initiation Interval)을 가지는 경우에 국한되어 왔다. 가변 데이타 입력간격 파이프라인은 고정된 데이타 입력간격 파이프라인을 보다 일반화한 형태로 보다 넓은 설계공간을 갖지만, 이의 설계과정이 복잡하여 고정된 데이타 입력간격이 주로 사용되었다. 그러나, 설계과정을 자동화함으로써 가변 데이타 입력간격의 탐색이 가능하여 사용자는 면적과 속도면에서 보다 효과적인 구조를 얻을 수 있다.

본 논문에서는 가변 데이타 입력간격을 지원하는 파이프라인 구조의 합성방법을 제시한다. 2장에서 기존의 고정된 데이타 입력간격의 파이프라인 구조와 합성방식에 대해 설명하고, 3장에서 가변 데이타 입력간격의 파이프라인 구조와 이의 지원을 위한 스케쥴링과 모듈할당 그리고 콘트롤 합성 과정을 기술한다. 4장에서 실험결과를 보이고 5장에서 결론을 맺는다.

Ⅱ. 파이프라인 구조와 합성방식

파이프라인은 반복적인 연산이 많고 연속적인 데이타를 고속으로 처리해야 하는 응용분야에 적합한 구조이다. 파이프라인 구조에서는 처리해야 할 일을 여러 단계로 분리시킴으로써 하나의 데이타가 입력되어 결과를 출력하기까지의 시간은 변화가 없지만 연속되는 데이타에 대해 분리된 단계를 병렬적으로 수행함으로써 performance 면에서 성능향상이 존재한다. DII는 고정된 데이타 입력간격을 가지는 파이프라인 구조에서 데이타의 입력이 발생하는 시간 간격을 클럭의 수로 표현한 값이다. 따라서, 파이프라인의 throughput은 DII와 클럭의 주기에 반비례 한다. 파이프라인 구조의 설계에 있어서 DII는 하드웨어의 면적과 속도의 tradeoff를 결정하는데 유용하게 사용된다. 일반적으로 파이프라인의 DII가 증가하면 속도가 감소하는 반면 파티션의 개수가 증가하여 공유 가능성이 커지고 면적면에서 효과적인 하드웨어를 얻을 수 있다.

그림 1은 5개의 스테이지를 가지고 데이타 입력간격이 2인 파이프라인 구조의 space-time diagram을 보인다. 그림에서 Ij는 task를 나타내고 Fi는 스테이지 si에서 수행하는 기능을 나타낸다. 시간 t = 0에서 task I1이 파이프라인에 입력되어 스테이지 s0과 s1에서 F0과 F1을 수행하고, 데이타 입력간격 만큼의 시간이 지난 시간 t = 2에 I1이 스테이지 s2과 s3에서 F2와 F3을 수행할 동안 새로운 task인 I2가 입력되어 스테이지 s0과 s1에서 F0과 F1를 수행한다.[4]

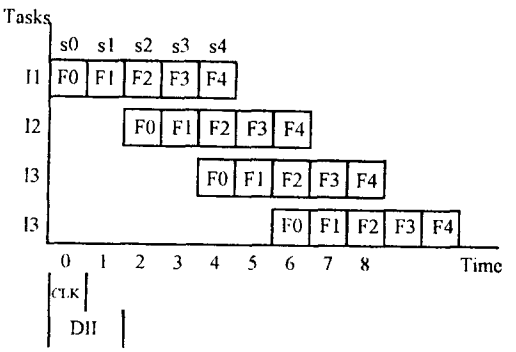


그림 1. DII가 2인 파이프라인의 space-time diagram

Fig. 1. Space-time diagram of pipeline with DII=2.

비파이프라인 구조의 합성의 경우 제어구간 사이에 시간적인 중첩이 없으므로 연산이 서로 다른 제어구간에 존재하기만 하면 하나의 연산자로 공유가 가능하다. 그러나 파이프라인 구조에서는 스테이지들이

DII의 간격으로 중첩되어 공유에 제약이 따른다. 그림 1의 경우 파이프라인이 정상상태에서 시간 4와 5의 상태를 반복하므로 F0, F2, F4와 F1, F3는 공유될 수 없다. 따라서, 그림 1에서 (F0, F1)과 (F2, F3)의 기능들을 각각 수행할 수 있는 하드웨어가 존재한다면 그림 2와 같이 하드웨어가 공유된 파이프라인 구조를 얻을 수 있다.

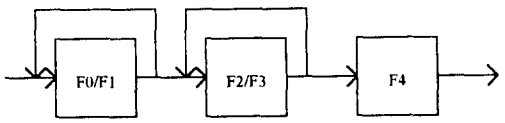


그림 2. 하드웨어가 공유된 파이프라인 구조
Fig. 2. Pipeline structure with hardware sharing.

파이프라인 파티션은 연속적으로 입력되는 데이타에 대해 동시에 수행되어 시간적인 중첩이 발생하는 스테이지의 집합으로 정의된다.(이하 파티션) 그림 1에서 집합 {s0, s2, s4}와 {s1, s3}이 두개의 파티션이다. 여기서, 파이프라인에는 DII개의 파이프라인 파티션이 존재한다는 것과 같은 파티션에 할당된 연산은 공유가 불가능하고 서로 다른 파티션에 존재하는 연산은 하나의 모듈로 공유가 가능하다는 것을 알 수 있다. 기존의 파이프라인 합성 시스템들은 이점을 효과적으로 이용하는 스케쥴링과 모듈할당 알고리듬을 가지고 있다.

Ⅲ. 가변 데이타 입력간격 파이프라인

1. 배경

그림 3은 파이프라인 구조에 따른 하드웨어 사용을 제시하고 있다. "+"와 "-"는 스테이지에서 수행되는 기능 F를 나타낸다. 그림 3의 (a)에 보인 파이프라인은 6단으로 구성되고 DII가 2로 고정된 경우로 2개의 파티션을 갖으나 공유가 이루어지지 못하여 덧셈기와 뺄셈기가 각각 3개씩 필요하다. 이때 효과적인 하드웨어의 공유를 위하여 지연 스테이지의 삽입 방식이 널리 사용된다.[7] 주어진 파이프라인 구조의 네 번째 스테이지에 1개의 지연 스테이지(D)를 삽입함으로써 그림 3의 (b)와 같이 7 단의 스테이지를 사용하여 덧셈기와 뺄셈기 각각 2개씩 사용하는 파이프라인 구조를 얻을 수 있다. 이 방법은 기존의 파이프라인 합성 시스템에서 스테이지의 수의 제약조건을 변화시킴으로써 얻을 수 있는 결과이다. 그림 3의 (c)의 파이프라인은 지연 스테이지의 삽입이 없이 데이타의 입력간격을 (1, 2)로 변화시킴으로써 얻은 결과로 덧셈기와 뺄셈기를 각각 2개씩 사용하여 면적을 줄이면서 평균적인 데이타 입력간격이 2에서 1.5로 줄어 25%의 성능향상도 얻을 수 있음을 알 수 있다.

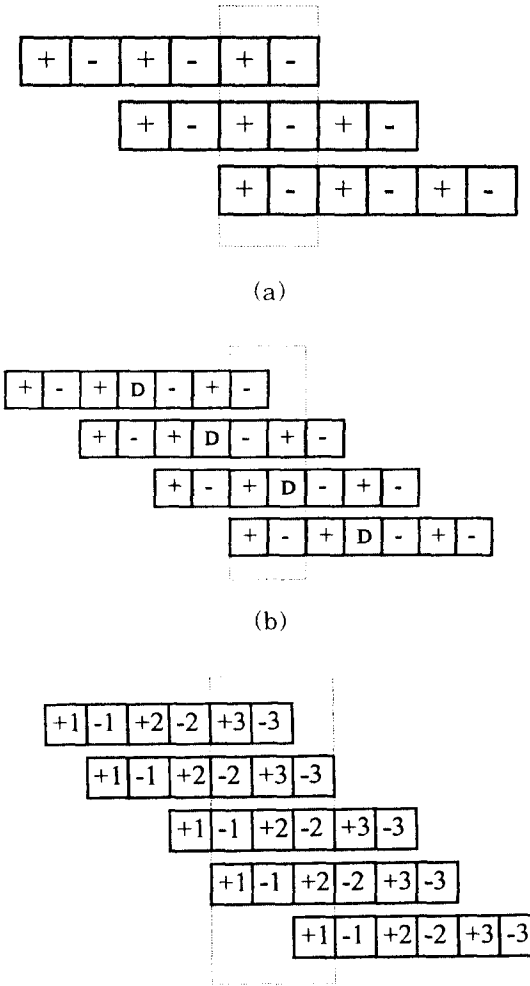


그림 3. 파이프라인 구조와 하드웨어 사용
(a) 고정된 데이타 입력간격 파이프라인 구조
(b) 지연 스테이지의 삽입 결과
(c) 가변 데이타 입력간격 결과

Fig. 3. Pipeline structures and hardware utilization.
(a) Pipeline structure with a fixed DII,
(b) After delay insertion,
(c) Pipeline structure with variable DIIs.

데이타 입력간격이 변하는 파이프라인 구조를 사용하여 보다 효과적인 파이프라인을 구성할 수 있음이 제안되었으나, 이의 자동설계를 지원하는 시스템은 발표되지 않고 있다.

가변 데이타 입력간격을 가지는 파이프라인 구조가 외부의 고정된 처리주기를 가지는 시스템과 연결될 경우에 버퍼를 필요로 한다. 버퍼의 크기는 가변 데이타 입력간격 파이프라인의 파티션의 수와 같다. 반면, 처리할 데이타가 메모리상에 존재하고 결과 또한 메모리에 저장된다면 메모리 억세스를 하나의 task로 생각하여 공유할 수 있다. 처리할 연산의 형태가 데이타 입력간격이 고정되어 효과적인 공유를 이루지 못하는 경우에, 일반화된 가변 데이타 입력 간격을 사용함으로써 FU를 효과적으로 사용하는 파이프라인 구조를 얻을 수 있다.

2. 파티션의 구성

고정된 데이타 입력간격의 파이프라인 구조의 합성에서 파티션은 하드웨어의 공유방법과 알고리듬의 설계에 있어서 매우 중요한 의미를 가진다. 가변 데이타 입력간격의 파이프라인에서 파티션을 표현하기 위하여 다음을 정의한다.

정의 1. 입력간격열 (Initiation interval Sequence, IS)

순차적으로 입력되는 데이타의 간격을 말하며 순환적으로 적용되는 입력간격렬 IS = $(I_0, I_1, \cdots, I_{L-1})$로 표현한다.

정의 2. 비순환 입력간격렬 (AIS: Acyclic IS)

주어진 입력간격렬 IS = $(I_0, I_1, \cdots, I_{L-1})$이 다음을 만족하는 정수 t가 존재하지 않으면 비순환 입력간격렬이다. 여기서, %는 *modulo* 연산을 의미한다.

$$(I_0, I_1, \cdots, I_{L-1}) = (I_{t\%L}, I_{(t+1)\%L}, \cdots, I_{(t+L-1)\%L}),$$
$$0 < t < L$$

예를 들어 입력간격렬 (1, 2, 1, 2)는 t가 2일 때 같은 간격렬을 생성하여 순환 입력간격렬이고, 비순환 입력간격렬 (1, 2)로 표현할 수 있다.

정의 3. 입력시간렬

입력간격렬 IS = $(I_0, I_1, \cdots, I_{L-1})$에 대하여, 입력시간렬 IT는 식 (1)과 같이 정의된다.

$$IT = (t_0, t_1, \ldots, t_L), t_0 = 0, t_i = \sum_{j=0}^{i-1} I_j (i \geq 1) \quad (1)$$

그림 4는 데이타 입력간격이 1과 2로 반복적으로 변하는 경우, 즉 IS = (1, 2), IT = (0, 1, 3)일 때 발생하는 데이타 입력을 보이고 있다. 첫 번째 행은 클럭의 수에 해당하는 시간을 나타내며, 두 번째 행은 해당 시간 동안 발생한 데이타 입력의 수를 나타내고 있다. 시간 k까지 발생한 데이타 입력의 시간을 입력 시간렬을 사용하여 보이고 있다. 수평 방향의 선분이 나타내는 시간 간격 동안 비순환 입력간격렬의 크기인 L번의 데이타 입력이 발생한다. 시간 k까지 포함된 선분의 수는 k/t_2 보다 크지 않은 최대의 정수인 $[k/t_2]$ 개로 $[k/t_2] * L$ 번의 데이타 입력이 존재한다. 시간 k에 걸쳐 있는 구간에서, 시간 k-2부터 시간 k까지 $k\%t_2$ 시간 간격 동안에 2번에 데이타 입력이 발생한다. 이 숫자는 $t_j \leq k\%t_2 < t_{j+1}$을 만족하는 j에 1을 더함으로써 얻을 수 있다. 그림에서 $t_1 \leq k\%t_2 < t2$을 만족하여 j가 1임을 알 수 있다. 이와 같은 계산을 일반화하면 정리 1을 얻는다.

정리 1. 비순환 입력간격렬 IS = $(I_0, I_1, \cdots, I_{L-1})$과 입력시간렬 IT = $(t_0, t_1, \cdots, t_L)$을 가지는 가변 데이타 입력간격 파이프라인 구조에서 시간 k까지 발생한 데이타 입력의 수 NI_k는 식 (2)와 같이 표현된다.

$$NI_k = L * \lfloor k / t_2 \rfloor + (j+1) \quad (2)$$

여기서 j는 $t_j \leq k \% t_L < t_{j+1}$를 만족하는 값이다.

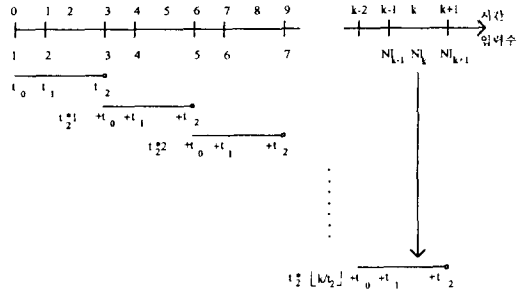


그림 4. 가변 데이타 입력간격에 따른 데이타 입력의 예

Fig. 4. An example of data initiation in pipeline with variable DIIs.

그림 5는 그림 4와 같은 데이타 입력간격을 가지고 스테이지의 수가 6인 파이프라인의 space-time diagram을 보이고 있다. 시간 k에서 스테이지 s1, s2, s4, s5가 활성화된다. 이들 중에서 최소 스테이지 번호 1은 시간 k에서 가장 최근에 발생한 데이타 입력시간과 시간 k의 차에 의해 결정됨을 알 수 있다. 그림 4에서 시간 $t_2 * [k/t_2] + t_1$에 데이타 입력이 발생했으므로 시간 k에 활성화되는 최소 스테이지 번호 $s_{min,k}$는 $k - (t_2 * [k/t_2] + t_1)$이 된다. 이것을 일반화하면 정리 2를 얻을 수 있다.

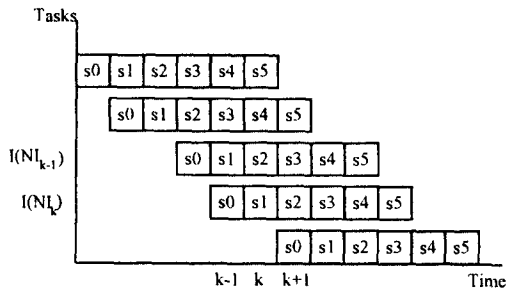


그림 5. 그림 4의 파이프라인에 대한 space-time diagram

Fig. 5. Space-time diagram for the pipeline in fig. 4.

정리 2. 비순환 입력간격렬 IS = (I_0, I_1, ⋯, I_{L-1})과 입력시간렬 IT = (t_0, t_1, ⋯, t_L)을 가지는 가변 데이타 입력간격 파이프라인 구조에서 시간 k에 수행되는 스테이지 중에서 최소 스테이지 번호 $s_{min,k}$는 시간 k를 기준으로 최근에 발생한 데이타 입력시간과 k의 차로서 결정되며 식 (3)과 같다.

$$s_{min,k} = k - \left(t_L * \lfloor k / t_L \rfloor + t_j\right) \tag{3}$$

여기서 j는 $t_j \le k\%t_L < t_{j+1}$를 만족하는 값이다.

시간 k에 수행되는 스테이지의 번호들은 정리 2의 최소 스테이지 번호에 입력간격렬을 역순으로 더함으로써 얻을 수 있다. 그림 5에서 최소 스테이지 번호 2에 (1, 2)를 반복적으로 누적하여 더한 3, 5, 6이 중첩되는 스테이지 번호와 같음을 알 수 있다.

정리 3. 비순환 입력간격렬 IS = (I_0, I_1, ⋯, I_{L-1})을 가지는 가변 데이타 입력간격 파이프라인 구조에서, 시간 k에 중첩되는 스테이지 번호는 최근에 발생한 데이타 입력을 수행하는 최소 스테이지 번호로부터 전에 발생한 데이타 입력의 시간차를 누적함으로써 계산할 수 있다. 시간 k에 중첩되는 스테이지의 집합 (파티션) P_k는 식 (4)와 같이 점화식으로 표현된다.

$$P_k = \{\text{stages whoes index is } s_n | s_n = s_{n-1} + I_{(j-n)\%L}, s_0 = s_{min,k}, 0 \le s_n \le N_{stage} - 1\} \tag{4}$$

여기서 j는 $t_j \le k\%t_L < t_{j+1}$를 만족하는 값이다.

정리 4. 파티션의 수

주어진 비순환 입력간격렬 IS = (I_0, I_1, ⋯, I_{L-1})에 대하여, $N_p = \Sigma I_i$ 개의 서로 다른 P_k (파티션) 가 존재한다.

증명〉 식 (1)의 입력시간렬의 정의에 의하여 N_p는 t_L과 같다. 입력시간 t_j의 j가 $k\%t_L$에 의해 결정되고 최소 스테이지 번호 $s_{min,k}$는 $s_{min,k+t_L}$과 같으므로 식 (5)와 같이 시간 $k+t_L$에서 중첩되는 스테이지의 집합 P_{k+t_L}는 P_k와 같다.

$$\begin{aligned} s_{min,k+t_L} &= \left(k + t_L\right) - \left(t_L * \lfloor \left(k + t_L\right) / t_L \rfloor + t_j\right) \\ &= k - \left(t_L * \lfloor k + t_L \rfloor + t_j\right) \\ &= s_{min,k} \end{aligned} \tag{5}$$

따라서, k를 $0 \le k < t_L$로 제한하면 식 (2)(3)(4)는 식 (6)(7)(8)로 간단히 표현될 수 있다.

$$NI_k = j + 1, \text{ s.t. } t_j \le k < t_{j+1} \tag{6}$$

$$s_{min,k} = k - t_j \tag{7}$$

$$P_k = \{\text{stages whose index is } s_n | s_n = s_{n-1} + I_{(j-n)\%L}, s_0 = s_{min,k}, 0 \le s_n \le N_{stage} - 1\} \tag{8}$$

k1, k2 ($k1 < k2$, $0 \le k1, k2 < t_L$)에 대하여, $t_j \le k1, k2 < t_{j+1}$을 만족하는 j가 존재하여 j1 = j2가 성립한다면 식 (7)에서 $s_{min,k1}$와 $s_{min,k2}$이 같을 수 없으므로 $P_{k1} \ne P_{k2}$이다. j1 ≠ j2일 경우도 비순환 입력간격렬의 정의 (정의 2)에서 식 (8)의 $I_{(j-n)\%L}$ 항이 같을 수 없으므로 $P_{k1} \ne P_{k2}$이다. 이상의 논리에 의하여 $N_p = t_L = \Sigma I_i$ 개의 서로 다른 파티션이 존재한다.

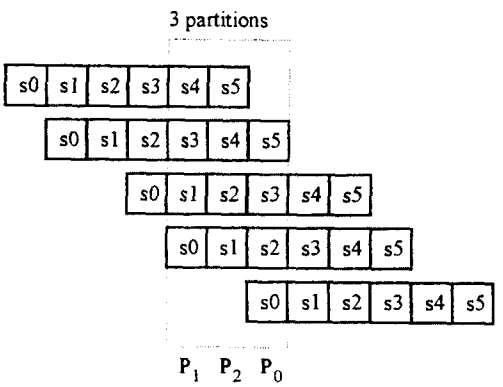


그림 6. 가변 데이타 입력간격 시의 파티션의 구성

Fig. 6. Pipeline partitions for the pipeline with variable DIIs.

그림 6은 데이타 입력간격렬 IS = (1, 2) 이고 스테이지의 수가 6인 파이프라인 구조의 스테이지 중첩

을 보이고 있다. 여기서 L은 2이고 입력시간렬 IT는 식 (1)에 의해 (0, 1, 3)이며 $t_L = 3$ 이므로 파티션의 수는 3이다. 각 파티션에서 최소 스테이지 번호는 식 (7)에 의하여 $s_{min,0} = 0 - 0 = 0$, $s_{min,1} = 1 - 1 = 0$, $s_{min,2} = 2 - 1 = 1$이다. 파티션 P_0에 속하는 스테이지 번호는 식 (8)에 의하여, $s_0 = s_{min,0} = 0$, $s_1 = s_0 + I_{(0-1)\%2} = 0 - I_1 = 2$, $s_2 = s_1 + I_{(0-2)\%2} = 2 + I_0 = 3$, $s_3 = s_2 + I_{(0-3)\%2} = 3 + I_1 = 5$이다. 따라서, 파티션 P_0 = { s0, s2, s3, s5 }가 된다. 같은 방법으로, 파티션 P_1와 P_2을 계산하면, P_1 = { s0, s1, s3, s4 } P_2 = { s1, s2, s4, s5 }가 되어 그림 6과 같음을 알 수 있다.

3. 가변 데이타 입력간격 파이프라인의 합성

가변 데이타 입력간격 파이프라인의 합성은 크게 스케쥴링, 모듈할당 그리고 콘트롤 합성으로 구성된다. 가변 데이타 입력간격 파이프라인을 지원하기 위한 스케쥴링은 주어진 제약조건을 만족하면서 연산을 스테이지에 최적으로 할당하며, 고정된 데이타 입력간격의 파이프라인 스케쥴링 알고리듬[9]에 III.2절에서 정의한 파티션을 고려하여 수행한다. 식 (9)에서 N_p는 파티션의 개수이고 p_{OP}와 P_{OP}는 각각 스테이지 i와 파티션 k에 OP 타입의 연산이 스케쥴될 수 있는 확률을 나타낸다. 이 식은 각 파티션에 속한 연산의 확률적인 분포를 나타내는 식으로 파티션 k의 스테이지를 나타내는 식 (8)의 P_k를 사용하여 변형한 것이다. 식 (10)의 H는 연산이 파티션에 균등히 배분된 정도를 0과 1사이의 값을 갖는 엔트로피 함수로 표현하여 1에 가까울수록 균등한 배분이 이루어짐을 의미한다. 식 (11)의 목적함수는 연산자의 타입에 따라 계산된 H를 연산의 면적과 연산의 개수의 함수인 w(OP)의 가중치를 주어 더한 것으로 모든 타입의 연산이 파티션에 균등히 분포된 척도를 나타낸다. 식 (12)는 우선순위 함수로 목적함수의 미분치를 직선으로 근사화 한 값이다.[9] 식에서 $ASAP_{opn}$과 $ALAP_{opn}$은 현재의 상태에서 연산 opn이 스케쥴될 수 있는 구간이다.

$$P_{OP}(k) = \sum_{\substack{for\ all\ i,\\ s.t.\ \ si \in Pk}} P_{OP}(i),\ \ k = 0, \ldots, N_p - 1 \qquad (9)$$

$$H(OP) = -\sum_{k=0}^{N_p-1} P_{op}(k)\ \log P_{op}(k) / \log N_p \qquad (10)$$

$$OF(S) = \sum H(OP) w(OP) \qquad (11)$$

$S_{opn,k}$: 노드 opn이 k 스테이지로 스케쥴링 된 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)

$$= (Mx - Mn) / \left(ALAP_{opn} - ASAP_{opn}\right) \qquad (12)$$

스테이지의 수와 데이타 입력간격렬을 입력으로 하드웨어의 면적을 최적화하는 스케쥴링 알고리듬을 그림 7에 보이고 있다. 스케쥴링 알고리듬은 식 (11)의 미분치를 식 (12)와 같이 근사화 한 값을 우선순위 함수로 갖는 반복적 구성 방식을 취하고 있다.

단계 1. 입력간격렬, 스테이지의 수, 클럭 주기를 받아 들인다.
단계 2. 식 (7)(8)을 이용하여 파티션을 결정한다.
단계 3. 초기상태로 각 노드 opn에 대하여 $ASAP_{opn}$과 $ALAP_{opn}$을 결정한다.
단계 4. 모든 노드가 스케쥴되었으면 종료.
단계 5. 스케쥴되지 않은 노드에 대하여 식 (5)의 우선 순위함수를 구한다.
단계 6. 최대의 우선순위함수를 가지는 노드를 스케쥴한다.
단계 7. 단계 4로 간다.

그림 7. 스케쥴링 알고리듬
Fig. 7. Scheduling algorithm.

모듈할당은 연산을 연산자에 할당하고 MUX와 latch를 사용하여 연결구조를 결정하여 데이타패스를 생성한다. 연산자 할당의 목적은 FU를 최대한 공유하여 사용할 수 있게 하는 것이다. 그림 3의 (c)와 같이 스케쥴링 된 결과에서 덧셈기의 할당을 살펴보면, 각 파티션에 속한 덧셈은 (+1, +3), (+2, +3), (+1, +2) 이다. 덧셈 연산을 노드로 하고 서로 공유가 가능한 연산들을 에지로 연결한 공유 가능성 그래프는 그림 8 (a)와 같다. 공유 가능성 그래프는 모든 노드가 연결된 상태에서 충돌이 발생하는 에지를 제거하여 얻는다. 그림에서 연산들은 공유가 불가능하여 3개의 덧셈기가 필요함을 알 수 있다. 즉, 한 연산이 여러 개의 파티션에 존재하므로 연산을 특정 연산자에 고정시킬 경우에 충돌이 발생한다. 그림 3의 (c)에서 연산을 시간적으로 구분하여 아래 첨자를 붙여 가상연산을 만들면 각 파티션에 속한 덧셈은 $(+3_1, +1_4)$, $(+3_2, +2_3)$, $(+2_4, +1_5)$ 이다. 이 경우 공

유 가능성 그래프가 그림 8의 (b)에 나타나 있다. 그림에서 clique에 해당하는 ($+3_1$, $+2_3$, $+2_4$)와 ($+1_4$, $+1_5$, $+3_2$)는 각각 공유가 가능하여 덧셈기 2개를 사용하는 파이프라인 구조를 얻을 수 있다.

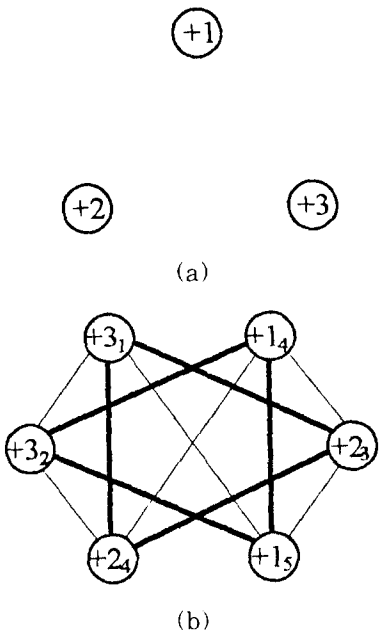


그림 8. 공유 가능성 그래프
(a) 연산이 연산자에 고정된 경우
(b) 가상연산을 사용한 경우

Fig. 8. Compatibility graphs.
(a) when an operation is executed on a fixed operator.
(b) when virtual operations areused.

가변 데이타 입력간격의 파이프라인이 정상적으로 동작하기 위해서는 적절한 콘트롤 신호가 필요하다. 파이프라인은 정상상태의 시간 k에서 수행해야 할 연산은 k % N_p 번째 파티션에 속하는 스테이지에 할당된 것들이다. 따라서, 콘트롤러는 파티션의 수인 N_p개의 상태를 반복하고, 각 상태에서 콘트롤러의 출력신호는 하드웨어 동작기술에 정의된 연산을 합성된 데이타패스로 수행하기 위해 피연산자를 연산자에 전달하고 연산 결과를 저장하는 load 신호, ALU 제어신호, MUX와 BUS 제어신호 등이다. 콘트롤 합성 과정에서는 파티션 별로 필요한 제어 신호를 모아 FSM (Finite State Machine)의 기술을 생성한다.

가변 데이타 입력간격을 지원하는 파이프라인의 합성에서는 가상연산의 사용이 불가피하여 동작기술 상의 한 연산이 시간에 따라 서로 다른 연산자에서 수행될 수 있으므로 부가적인 연결구조를 필요로 한다. 또한, 콘트롤러의 구현에 있어서도 상태의 수가 증가하고 부가적인 연결구조의 제어를 위해 출력신호의 수도 증가하여 overhead가 따른다. 본 연구에서는 가변 데이타 입력간격을 가지는 파이프라인 구조의 데이타패스를 MUX와 latch의 연결구조로 구성하고 FSM 합성을 통하여 랜덤로직으로 콘트롤러를 구현하여 전체적인 면적을 비교하였다.

Ⅳ. 실험 결과

본 단원에서는 가변 데이타 입력간격을 가지는 파이프라인 구조로 합성된 데이타패스와 콘트롤러의 간단한 예를 보이고, 기존의 상위수준 합성에서 벤치마크로 사용하는 회로에 대한 합성을 통하여 가변 데이타 입력간격 파이프라인의 효용성을 보인다.

그림 9는 그림 3의 (c)에 보인 파이프라인에 대해 합성된 데이타패스와 콘트롤을 보이고 있다. 그림 9의 (a)는 데이타패스를 보이고 있으며 그림 9의 (b)에서 TT format으로 기술된 콘트롤러를 보인다. 데이타패스는 4개의 ALU와 4개의 2:1 MUX로 구성

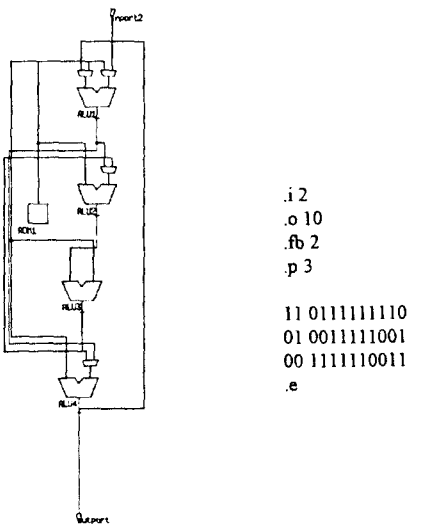


그림 9. 그림 3 (c)의 합성 결과 (a) 데이타패스와 (b) 콘트롤러 기술

Fig. 9. Synthesis results for the pipeline of Fig.3 (c), (a) Datapath and (b) Controller description.

되어 있으며, 각 ALU에서 수행하는 연산을 표 1과 보인다. 콘트롤러는 '00', '01' '11'로 할당된 3개의 상태를 가지며, 10개의 출력은 다음 상태를 위한 2비트, ALU의 enable신호 4비트 그리고 4개의 2:1 MUX에 대한 선택신호이다.

표 1. ALU가 수행하는 연산
Table 1. Operations executed in ALUs.

ALU	기 능	할당된 연산
ALU1	+	$+3_2$, $+1_4$, $+1_5$
ALU2	-	-1_3, -1_4, -2_3
ALU3	+	$+2_4$, $+2_3$, $+3_1$
ALU4	-	-3_2, -2_2, -3_1

표 2. 합성에 사용한 라이브러리
Table 2. Library modules used for synthesis.

모 듈	면적 (gate 수)	지연시간
+,-	292	64
*	3946	120
MUX	64	5
Latch	80	5

상위수준 합성의 벤치마크 회로에 대하여 가변 데이타 입력간격을 지원하여 합성한 결과를 기존의 방법과 비교하기 위하여 합성에 사용한 라이브러리 모듈의 면적과 지연시간을 표 2에 제시하였다. 표 3은 벤치마크 회로인 16-point FIR 필터를 스테이지의 수를 6으로 하고 클럭을 150nS로 하여 합성된 결과를 보인다. 왼쪽은 고정된 데이타 입력간격을 사용한 기존의 결과[9]이고, 중간에 가변 데이타 입력간격을 채택하여 합성된 결과를 보이며, 오른쪽에 속도와 연산자 면적의 변화를 %로 보이고 있다. 표 2에서 덧셈기와 곱셈기는 개수로 MUX와 latch 그리고 콘트롤러는 게이트 수로 나타낸 것이다. 데이타 입력간격이 5로 고정된 경우, 합성된 데이타패스는 4개의 덧셈기, 2개의 곱셈기, 23개의 2:1 MUX 그리고 46개의 latch로 구성되고 콘트롤러는 32개의 게이트를 가진다. 가변 데이타 입력간격 (4, 5)로 합성된 파이프라인은 같은 개수의 FU를 가지지만 전체 게이트 수는 연결구조의 overhead 때문에 4.6%가 증가하였으나 속도가 10%가 향상되었다.

표 3. 16 points FIR 필터의 합성 결과
Table 3. Synthesis results for 16 point FIR filter.

DII	+	*	MUX	Latch	Cont	Tot	AIS	+	*	MUX	Latch	Cont	Tot	AS(%)	AA(%)
5	4	2	23	46	32	14244	(4,5)	4	2	29	49	59	14859	10	4.6

표 4는 5차 엘립틱 필터를 클럭을 150nS로 하여 9개의 스테이지로 합성한 결과이다. 데이타 입력간격 열이 (4, 6)일 때 속도의 변화 없이 면적을 12.8% 줄일 수 있었다. 다른 열들의 결과는 적은 양의 하드웨어 overhead로 속도가 향상된 경우이다.

표 4. 5차 엘립틱 필터의 합성 결과
Table 4. Synthesis results for 5th order elliptic filter.

DII	+	*	MUX	Latch	Cont	Tot	AIS	+	*	MUX	Latch	Cont	Tot	AS(%)	AA(%)
5	8	4	29	52	25	24161	(4,6)	7	3	43	54	106	21060	0	-12.8
							(3,6)	6	8	42	57	60	25458	10	5.2
7	6	2	29	46	49	15229	(6,7)	6	2	36	48	99	15887	7.1	4.3
8	6	2	29	46	50	15230	(6,9)	5	2	35	48	67	13499	6.3	1.8

표 5는 AR 필터[3]의 합성 결과로 고정된 데이타 입력간격의 파이프라인 구조에서 얻을 수 없었던 새로운 설계점들이 합성되었다. IS가 각각 (1, 3)과 (3,5)인 경우에 속도의 감소 없이 14.4%와 22.7%의 면적이 절약되었다. 특히, IS가 (5, 6)인 경우 속도와 면적면에서 모두 향상된 결과를 보인다.

표 5. AR 필터의 합성 결과
Table 5. Synthesis results for AR filter.

DII	+	*	MUX	Latch	Cont	Tot	AIS	+	*	MUX	Latch	Cont	Tot.	AS(%)	AA(%)
2	12	10	8	10	5	44281	(1,3)	8	8	20	34	9	37911	0	-14.4
4	8	8	17	27	12	37164	(2,5)	6	8	30	29	43	37603	12.5	1.2
							(3,5)	4	6	30	24	41	28725	0	-22.7
6	4	4	28	2	25	21089	(5,6)	4	4	28	23	42	20625	8.3	-2.2

그림 10은 가변 데이타 입력간격 파이프라인에서 발생하는 면적의 overhead를 보이고 있다. 이 그림은 표 5의 첫 번 째와 두 번째 열의 결과에 해당한다. 그림 10의 (a)에서 가변 데이타 입력간격으로 합성한 경우 FU의 게이트 수가 42964에서 33904로 현저히 감소되었다. 반면, 콘트롤러와 연결구조에 사용된 게이트 수는 1317에서 4009로 증가하여 전체

면적에서 콘트롤러와 연결구조가 차지하는 면적이 3%에서 10.5%로 증가하였으나 그 양이 FU의 감소량에 못미쳐 결과적으로 14.4%의 면적 감소가 발생한다. 그림 10의 (b)는 IS가 (2, 5)로 평균 DII가 3.5로 12.5%의 속도 향상이 존재하는 경우이다. FU에 사용한 게이트 수는 33904에서 33320으로 감소된 반면 콘트롤러와 연결구조에 사용한 게이트 수가 3260에서 4283으로 증가하였다. 전체 게이트 수는 콘트롤러와 연결구조의 overhead 때문에 1.2% 증가하였으나 12.5% 의 속도향상을 고려하면 작은 값이다.

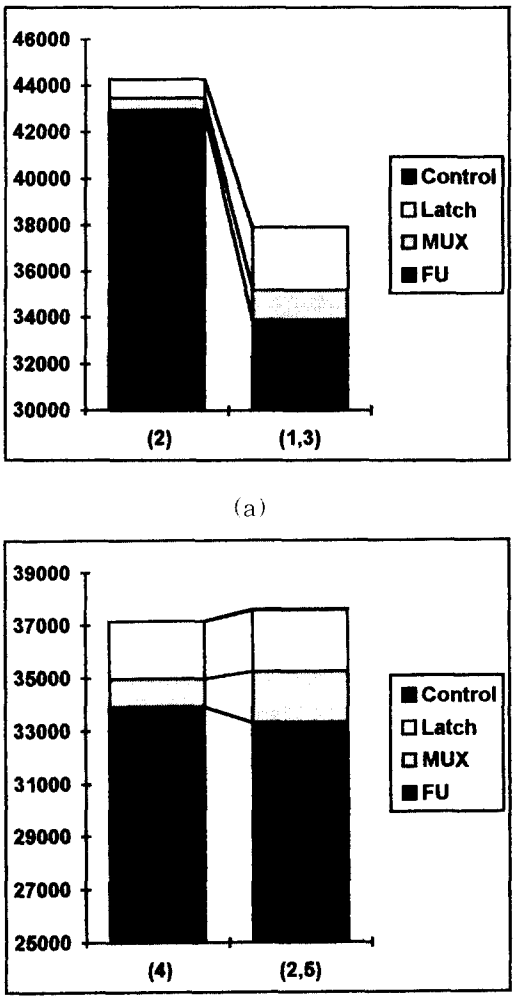


그림 10. AR 필터에 대한 게이트 수 비교
(a) 면적면에서 효율적인 설계
(b) 속도면에서 효율적인 설계

Fig. 10. Comparison of gate counts for AR filter.
(a) Area efficient design.
(b) Higher throughput design.

V. 결론

본 논문에서는 기존의 고정된 데이타 입력간격의 파이프라인 합성보다 넓은 설계공간을 사용자에게 제공할 수 있는 가변 데이타 입력간격을 가지는 파이프라인 구조를 자동적으로 합성하는 방법을 제시하였다. 가변 데이타 입력에 의한 스테이지의 시간적인 중첩을 결정하여 파티션을 정의함으로써 효율적인 하드웨어 모듈의 공유를 가능하게 하였고, 결정된 파티션에 연산을 균등히 배분하는 스케쥴링 알고리듬과 가상연산을 사용한 모듈할당 알고리듬을 구현하여 데이타패스를 합성하였으며 FSM 합성과 논리합성 과정을 통하여 콘트롤러를 구현하였다 파이프라인의 데이타 입력간격을 변화시킬 경우 고정된 경우보다 연산의 공유 형태가 불규칙하여 연결구조와 콘트롤러에 overhead가 따른다. 실험에서 연결구조와 콘트롤러를 포함한 면적과 속도를 비교함으로써, 기존의 파이프라인 구조 합성 시스템에서 얻을 수 없는 면적과 속도면에서 효율적인 설계점에 도달할 수 있음을 보였다. 추후 과제로 입력된 하드웨어 기술의 특성에 따라 효과적인 입력간격렬을 결정하기 위한 설계의 예측 방법에 관한 연구가 필요하다.

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.

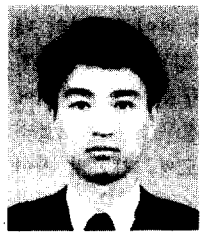
[2] K. Hwang, A. E. Casavant, "Scheduling and hardware sharing in pipelined data paths," in *Proc. ICCAD*, pp. 24-27, Nov. 1989.

[3] R. Jain, A. C. Parker, "Predicting area-time tradeoffs for pipelined design," in *Proc. 24th DAC, pp. 35-40*, June 1987.

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

[5] M. C. McFarland, A. C. Parker, "The high level synthesis of digital systems," *IEEE Proceedings*, Vol. 78, No. 2, pp. 301-318, Feb. 1990.
[6] 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.
[7] N. Park, *Synthesis of High-Speed Digital Systems*, PhD thesis, University of Southern California, Oct. 1985.
[8] P. Paulin, "Force directed scheduling for the behavioral synthesis of ASIC's," *IEEE Trans. CAD*, Vol. 8, No.6, pp. 661-679, June 1989.
[9] 전 홍신, 황 선영, "디지탈 신호처리를 위한 파이프라인 데이타패스 합성 시스템의 설계", 대한 전자공학회 논문지, 30-A권 6호, 1993년 6 월.

著 者 紹 介



田 弘 信(正會員)

1967年 8月 22日生. 1989年 2月 서강대학교 전자공학과 졸업(학사). 1991年 2月 서강대학교 대학원 전자공학과 석사. 1991年 3月 ~ 현재 동대학원 박사과정 재학중. 주관심 분야는 상위수준 합성, DSP 아키텍춰 등임.

黃 善 泳(正會員)

1976年 2月 서울대학교 전자공학과 졸업. 1978年 2月 한국 과학원 전기 및 전자 공학과 공학 석사 취득. 1986年 10月 미국 Stanford 대학 공학 박사 취득. 1976年 ~ 1981年 삼성 반도체 주식회사 연구원. 1986年 ~ 1989年 Stanford 대학 Center for Integrated Systems 연구소 연구원, Fairchild Semiconductor Palo Alto Research Center 기술자문. 1989年 3月 ~ 현재 서강대학교 전자공학과 부교수. 주관심 분야는 CAD 시스템, Computer Architecture 및 Systems Design, VLSI 설계 등임.