상위수준 합성에서의 클록 선택 방법

오주영*

A method for Clock Selection in High-Level Synthesis

Ju-Young Oh*

요 약

상위수준합성에서 클록 선택은 시스템의 성능과 설계의 질에 큰 영향을 미친다. 대부분의 시스템에서 클럭은 사전에 설계자에 의해 미리 명시되어야하지만, 최상의 클록은 상이한 스케줄의 결과를 평가한 후에 탐색이 가능하다. 본 연구에서는 체이닝이 가능한 연산 집합으로부터 클록을 선택하면서 동시에 스케줄링 하는 방법을 제안한다. 제안 스케줄링 알고리즘은 선택된 클록 주기에 기초하여 비트 단위 지연시간을 고려한 체이닝을 수행하며 리스트 스케줄링 방법으로 진행한다. 실험 결과는 제안 방법이 18%의 성능 개선이 있음을 보인다.

Key Words : Scheduling; Slack; Critical path; Clock Selection; Chaining.

ABSTRACT

Clock selection has a significant impact on the performance and quality of designs in high-level synthesis. Almost systems require that the clock length is required prior to scheduling, the best value of the clock can be found only after evaluating different schedules. In this study, we presents a scheduling method that works simultaneously with synthesis by selecting a clock from a chainable operation set. Our scheduling algorithm is based on list scheduling and executes chaining considering bit level delays based on selected clock period. Experimental results show that our method improves the performance by 18 percent.

Ⅰ. 서 론

요구시스템의 복잡도가 증가하고 시장성에 맞는 생산주기를 담보하기 위한 설계방법으로 상위수준 합성과 모듈 기반의 재사용 및 플랫폼을 수반한 기법 등의 다양한 연구가 진행 되고 있다. 특히, 행위 기술을 레지스터 전이수준의 구조기술로 번역해가는 상위수준 합성은 많은 연구가 진행되었다. 상위수준 합성 과정은 크게 스케줄링과 바인딩으로 이루어져 있으며 스케줄링은 주어진 제약조건 안에서 자료흐름도(DFG) 내의 연산자(OP)들의 시작 시간을 결정하는 과정으로 설계의 질을 결정하는데 가장 중요한 역할을 하며, 스케줄링 과정에서 멀티사이클링이나 체이닝 등을 함께 고려한다. 기존의 상위수준 합성에서 시간 모델은 최장 시간 자원의 지연시간으로 설정하였다. 즉, 두 연산의 지연시간의 합이 한 사이클 이내

* 교신저자 경인여자대학 정보미디어과 교수 (odid080@kic.ac.kr)
접수일자 : 2011년 4월 10일, 수정일자 : 2011년 5월 3일, 심사완료일자 : 2011년 5월 29일

인 경우만 체이닝이 가능했다. 그러나 체이닝 했을 때의 지연시간은 각 연산의 최장 시간의 합보다는 작은 값을 가지므로 하드웨어의 낭비가 발생한다. 이와 같은 낭비를 줄이기 위해 비트단위 체이닝[1], 비트단위 분할[2] 등의 기법이 제안되었다. 하지만 위 기법에서는 특정 자원에 대한 지연시간에 대해서만 고려하고, 클록의 주기에 대해서는 고려하지 않았다. 본 논문에서는 임계경로를 기반으로 체이닝이 가능한 연산집합으로부터 클록 주기를 선택하고 이를 기반으로 스케줄링 하는 방법을 제안하였다.

II. 재료 및 방법

1. 비트단위 지연시간 기반의 체이닝

그림 [1]은 0.18um 공정상 두 연산자의 체이닝 조합에 따른 실제 지연시간을 나타내며 체이닝의 각 조합은 단일 연산의 최장시간에 비해 작은 값을 가진다[3].

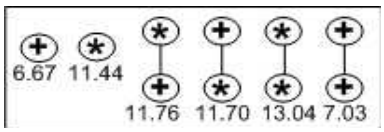


그림 1. 체이닝 연산의 지연시간
Fig. 1. Delay values of an adder, a multiplier, and several chained operations

그림 [2]는 비트단위 지연시간을 고려하여 자원제약 조건 없이 스케줄링 할 때 클록 주기를 0.18um 공정상의 6.67과 7.03으로 비교 수행한 결과로서 체이닝에 의해 제어단계를 줄임으로서 실행시간이 40.02에서 35.15로 향상됨을 알 수 있다.

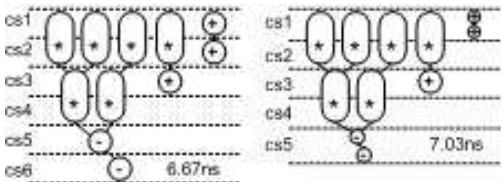
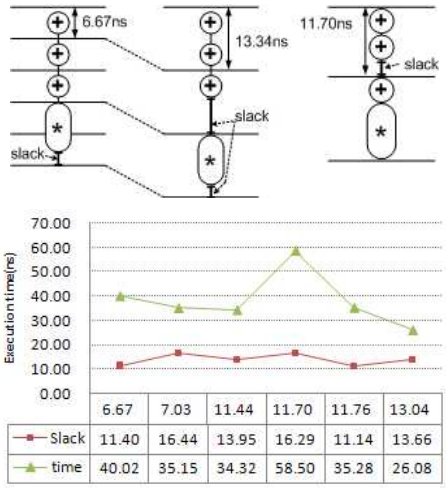


그림 2. 비트단위 지연시간을 고려한 스케줄링
Fig. 2. Scheduling considering bit level delays

2. 클록 선택

그림 [3]은 클록 주기 할당이 스케줄링에 부과하게 되는 잠재적 자원 낭비 요인인 slack과 0.18um 공정에 준해 체이닝을 고려한 스케줄링 결과 실행시간에 미치는 영향을 예시한다.



	6.67	7.03	11.44	11.70	11.76	13.04
Slack	11.40	16.44	13.95	16.29	11.14	13.66
time	40.02	35.15	34.32	58.50	35.28	26.08

그림 3. 스케줄상의 클록 주기의 영향
Fig. 3. Effect of clock period on scheduling

3. 클록 선택 방법

스케줄링은 자원제약 하에서 실행시간을 줄이는 목적함수로 설정하고 클록 선택을 위한 가정으로서 연산자 간의 체이닝은 0.18um 공정에 준해 진행하며 모듈 선택이 선행되어 할당된 자원의 지연시간이 라이브러리로부터 제공되는 것으로 하였다.

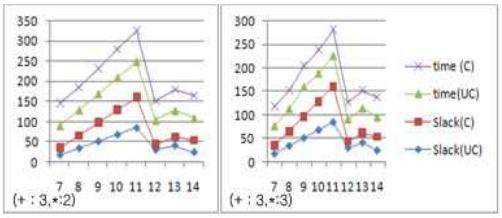


그림 4. 상이한 클록에 대한 자원제약 스케줄 결과
Fig. 4. RCS results for different clocks

제안 방법의 동기는 그림[4]의 결과에서 도출하였으며, 각기 상이한 클록 주기에 대한 스케줄 결과 slack과 스케줄 결과의 실행시간은 최소시간의 클록 주기로부터 11까지 비례 상승하며 체이닝이 가능한 연산 조합이 적용될 수 있는 12에서 현격하게 줄어들며 부과되는 자원 제약에 무관하게 동일한 결과를 보인다.

$$slack_i = clk \bullet \left\lceil \frac{delay_i}{clk} \right\rceil - delay_i \quad (1)$$

$slack_i$는 클록 주기 clk에서의 유휴시간이고, num_j는 임계경로상의 연산의 발생 수로 정의할 때, 클록 주기 clk 선택에 의해 유발되는 임계 경로상의 slack 총량은 수식(2)와 같다.

$$Slack_{tot}(clk) = \sum_j \{num_j \bullet slack_j(clk)\} \quad (2)$$

$$relTimeRed_{clk} = reduceTime_{clk} \bullet \frac{num_j^{clk}}{\sum_{i \in opSet} num_j^i} \quad (3)$$

수식(3)은 클록 주기로 clk를 선택할 때, 임계경로상의 상대적 발생 빈도수가 반영된 일반 연산 수행대비 체이닝에 의해 감소되는 실행 시간분이다.

$$Weight_{clk} = \alpha \cdot \frac{1}{Slack_{tot}(clk)} \times \beta \cdot relTimeRed_{clk} \quad (4)$$

클록 주기는 slack 총량이 작고, 자원제약이 만족되는 경우 발생 빈도수가 커서 잠재적인 스케줄 길이를 줄일 수 있게 하고, 비체인 일반 연산에 비해 체이닝 시에 감소되는 시간을 발생 빈도수를 고려해서 큰 값이 선택될 수 있도록 해야 하므로 수식(4)에 의해 도출되는 값을 최대로 하는 clk를 클록 주기로 선택한다.

4. 스케줄링 알고리즘

비트단위 지연시간에 의한 체이닝을 고려한 스케줄링 알고리즘은 리스트 스케줄링을 기반[3]으로 하였으며 그림[5]와 같다. 자원 제약조건에 대해 입력 DFG의 임계경로를 기준으로 클록 주기를 선택하고, 연산들의 우선순위를 결정한다. 생성된 readyList에 삽입된 연산의 조합으로 체이닝할 경우의 지연시간이 클록 주기 내에 있을 때 체이닝 가능 연산자로 연결되며 우선순위가 높은 연산자 순으로 스케줄하고 DFG내의 모든 연산자가 스케줄 될 때 까지 반복된다.

```
step1: Select the clock period
step2: Calculate priorities for all nodes
step3: Build the readyList
step4: While (scheduling is not completed)
step5:    for (each node n in Ready_list)
             Find chainable OP with n
          end for
step6: While(under RC || chainable OP)
          Schedule the highest priority OP
       end while
step7: If(multi-cycle is allowed)
          for(scheduled node in the readyList)
             if(exists available resource)
                Execute multicycle chaining
             end if
          end for
       end if
step8: Increase control step
step9: Update the readyList
step10: end while
```

그림 5. 스케줄링 알고리즘
Fig. 5. Scheduling algorithm

III. 결과 및 고찰

스케줄링은 이차미분방정식을 입력 DFG로 하고 자원제약 하에서 실행시간을 최소화하는 목적

함수에 따라 진행된다.

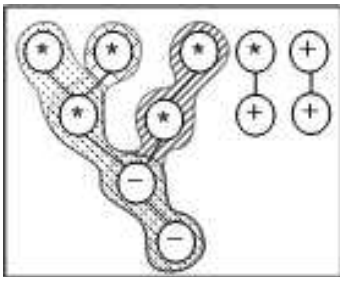


그림 6. 입력그래프와 인계경로
Fig. 6. Input DFG and critical path

입력 DFG로부터 축출되는 임계경로에서 0.18um 공정상 체이닝이 가능한 각 연산자의 조합에 대한 빈도수와 자원제약 및 잠재적인 slack 총량에 기준한 컬럭 주기 clk를 결정한 후, 결정된 주기에 대해 제어단계 내에서 체이닝이 가능한 후보군을 선별하여 우선순위에 따라 스케줄링을 수행해 나간다. clk 적용에 따른 slack 총량은 임계경로 상에 분포하는 연산 유형별로 계산되며 자원제약에 상관없이 DFG의 입력 상황에 따라 결정되며 표 1과 같다.

표 1. 각 클록에 대한 slack 총량
Table 1. Total slack for each different clock

clk	+	++	*	+*	*+	**	$Slack_{tot}(clk)$
num_j	2	1	5	0	2	3	
6.67	0	6.31	1.9	1.64	1.58	0.3	19.87
7.03	0.36	0	2.62	2.36	2.3	1.03	21.51
11.44	4.77	4.41	0	11.18	11.12	9.84	65.71
11.7	5.03	4.67	0.26	0	11.64	10.36	70.39
11.76	5.09	4.73	0.32	0.06	0	10.48	47.95
13.04	6.37	6.01	1.6	1.34	1.28	0	29.31

연산의 발생 빈도와 자원제약을 고려한 값에 slack의 총량을 역으로 반영한 계산 결과는 표 2와 같고 자원제약이 없을 경우 13.04를, 자원제약 +:3, *:2에서는 11.44를 각각 clk로 선택하게 되며 그림[7-8]과 같이 선택된 clk에 대한 리스트 스케줄 결과에서 실행시간이 최적이 됨을 확인하였다.

표 2. 자원제약에 대한 클록 선택
Table 2. Clock selection for resource constraint

clk	num_j	$Slack_{tot}(clk)$	without RC		RC(+:3, *:2)	
			$relTime\ \mathrm{Re}\, d_{clk}$	$Slack_{tot}(clk)$	$relTime\ \mathrm{Re}\, d_{clk}$	$Slack_{tot}(clk)$
6.67	2	19.870	0.965	23.272	0.322	7.757
7.03	1	21.510	0.983	25.323	0.114	2.934
11.44	5	65.710	3.297	167.684	1.197	83.723
11.7	0	70.390	0.01	0.772	0.003	0.259
11.76	2	47.950	2.94	153.465	0.495	25.854
13.04	3	29.310	7.842	263.151	1.965	65.949

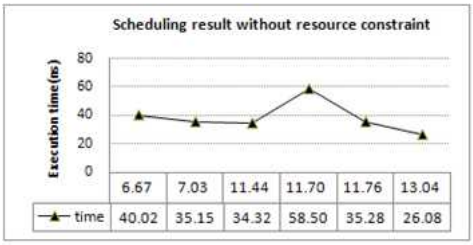


그림 7. 자원제약 없는 스케줄링 결과
Fig. 7. Scheduling result without resource constraint

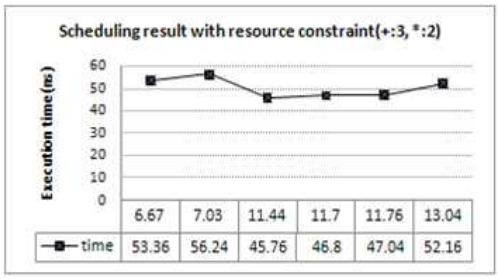


그림 8. 자원제약하의 스케줄링 결과(+:3,*:2)
Fig. 8. Scheduling result with resource constraint(+:3,*:2)

Ⅳ. 결 론

본 논문에서는 상위수준합성의 스케줄링 결과에 큰 영향을 주는 클록 선택 방법을 제안하였다. 클럭은 체이닝에 의해 줄일 수 있는 잠재적인 스케줄 시간을 조합이 가능한 연산 유형의 발생 빈도와 자원제약 및 slack의 총량을 반영하여 선택하였으며 선택된 클럭에 따른 스케줄 결과는 18%의 효율을 보였다. 향후 작업으로서 다양한 DSP 예제에 대한 실험과 복합적인 목적시스템에 부합되는 클록 주기 선택에 대한 연구가 진행되어야 한다.

참 고 문 헌

[1] S. Park and K. Choi, “Performance-driven high-level synthesis with bit-level chaining and clock selection,” IEEE Trans. CAD, Vol. 2, no. 2, pp. 199 - 212, Feb. 2001.

[2] M. Molina, R. Ruiz-Sautua, J. Mendias, and R. Hermida, “Bitwise scheduling to balance the computational cost of behavioral specifications,” IEEE Trans. CAD, Vol. 25, no. 1, pp. 31-46, Jan. 2006.

[3] Jiwoong Kim, Hyunchul Shin, "Scheduling considering bit level delays," SoC Design Conference, Vol. 1 pp. 330-333, Nov. 2008.

[4] J. Cong, Z. Zhang, “An Efficient and Versatile Scheduling Algorithm Based On SDC Formulation,” In Proc DAC, pp. 433 -438, 2006.

저자약력

오 주 영(Ju-Young Oh) **종신회원**

1998년 2월 : 홍익대학교 전자계산과(이학 석사)
2004년 8월 : 홍익대학교 전자계산과(이학 박사)
2002년 3월~현재 경인여대 정보미디어학부 교수

<관심분야> 설계자동화, 통합설계, 암호화/보안