효율적 자원제한 스케줄링 알고리즘

송호정, 정회균, *황인재, 송기용
충북대학교 컴퓨터공학과, *컴퓨터교육과

An Efficient Resource-constrained Scheduling Algorithm

Ho-Jeong Song, Hoi-Gun Jeong, *In-Jae Hwang, Gi-Yong Song
Dept. of Computer {Engineering, *Education}, Chungbuk National University
{hjsong, harnle}@archi.chungbuk.ac.kr, {*ihwang, gysong}@chungbuk.ac.kr

요 약

High-level synthesis(HLS)는 주어진 동작(behavior)과 면적(area), 성능(performance), 전력 소비량, 패키징, 테스팅등의 주어진 제한을 만족하게 구현된 구조적 디자인을 생성한다. 즉 high-level synthesis란 디지털 시스템의 알고리즘 단계 서술로부터 레지스터 전달구조의 구현에 이르는 과정을 의미한다. 이러한 high-level synthesis의 과정은 컴파일, 분할(partitioning), 스케줄링(scheduling)등의 단계를 거쳐 디지털 시스템을 설계할 수 있다.

본 논문에서는 high-level synthesis의 단계 중 스케줄링 과정에서 제한조건이 실리콘 면적으로 주어지는 경우에 최적의 functional unit의 수를 찾아내어 최소의 control step에 효과적으로 스케줄링 가능한 알고리즘을 제안하였다.

Abstract

High-level synthesis generates a structural design that implements the given behavior and satisfies design constraints for area, performance, power consumption, packaging, testing and other criteria. Thus, high-level synthesis generates that register-transfer(RT) level structure from algorithm level description. High-level synthesis consist of compiling, partitioning, scheduling.

In this paper, we proposed the efficient scheduling algorithm that find the number of the functional unit and scheduling into the minimum control step with silicon area resource constrained.

I. 서론

VLSI 기술은 단일 칩에 수백만개 이상의 트랜지스터를 집적하고 있는 가운데 computer-aided design(CAD) 연구는 설계 과정상 회로 시뮬레이션, 배치, floorplanning, 라우팅 등 low level에서의 여러 문제들을 성공적으로 해결해 오고 있다.

설계 복잡도가 증가하고 time-to-market이 짧아지면서 CAD 연구는 high level로 그 초점이 옮겨지고 있으므로, 게이트 또는 트랜지스터가 아닌 기억소자, 레지스터 ALU, 버스 등이 구성단위가 되는 high level은 다루어야 할 설계 객체(object)가 줄어드는 반면 설계 공간은 확대된다. 즉, 다양한 설계가 존재하게 되며 최적구현을 위한 설계 공간의 빠른 탐색이 필요하게 된다.

High-level synthesis는 디지털 시스템의 동작(behavior)을 레지스터 전달 단계 구현으로 바꾸어주는 변환으로, 동작은 입력과 출력 사이의 매핑(mapping)을 명시하며 레지스터 전달 단계 구조는 데이터패스(datapath)와 유한상태제어기(finite state controller)로 구성된다. 데이터패스는 통상 레지스터 컴포넌트 라이브러리로부터 선택된 소자들로 구성된다.

본 논문에서 제안된 알고리즘은 제한조건이 실리콘 면적으로 주어지는 경우에 최적의 functional unit의 수를 찾아내어, 최소의 control step에 스케줄 하는 알고리즘이다.

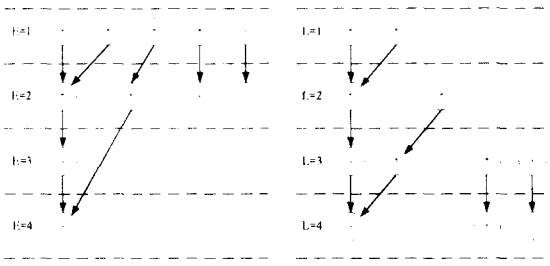
II. 스케줄링

동작은 구현된 회로에서 수행되는 내용을 나타내며 control/data flow 그래프(CDFG) 등의 중간 표현을 거쳐 구현된다. 스케줄링은 CDFG를 하나의 control step에서 수행되는 부그래프로 구분하는 기능이며 각각의 control step은 제어부 유한상태기계의 한 상태에 대응한다. 하나의 control step 내에서 할당된 연산수행을

위해 별도의 functional unit이 필요하며 여러 연산이 한 control step에 스케줄되면 그 만큼의 functional unit이 필요하게 되고, 각 control step에 스케줄된 연산을 적게하면 보다 많은 control step이 필요하게 된다.

즉, 스케줄링의 목적은 주어진 제한조건 내에서 수행에 필요한 소요시간 또는 control step 수를 최소화하도록 각 연산을 time step에 할당하는 것이다.

1. 기본 알고리즘



<그림 1> ASAP 스케줄링, ALAP 스케줄링

동작은 조건문, 반복문을 포함하지 아니하고 각 연산이 하나의 control step 내에서 수행되며 각 연산이 하나의 functional unit에서 수행되는 가정하에 ASAP(As Soon As Possible), ALAP(As Late As Possible)의 간단한 스케줄링 알고리즘을 생각할 수 있으며 이들은 제한조건을 가지는 경우의 기본형으로 적용된다<그림 1>

2. 시간제한 스케줄링

고정된 control step 내에서 소요 functional unit를 최소화하는 스케줄링으로 입력데이터 샘플링 비율에 의해 현재 데이터에 대한 최대처리시간이 제한되는 디지털신호처리(DSP) 시스템과 같은 실시간 시스템 설계에 적용된다.

수학적 프로그래밍, 구조적 발견(constructive heuristic), 반복적 개량(iterative refinement) 방식이 있으며 각각의 예로 일부초기의 결정이 탐색 진행 중 재방문되는 역추적(backtracking)을 이용하는 branch-and bound 방식으로 최적 스케줄을 수행하는 integer linear programming(ILP), 동일형태연산의 전체가용 상태에 대한 균등분배를 통하여 소요 functional unit의 최소화를 꾀하는 force-directed 스케줄링, 그리고 반복 재스케줄링(iterative rescheduling) 방식이 있다.

3. 자원제한 스케줄링

자원제한 스케줄링은 실리콘 면적에 제한이 주어지는 경우에 적용되는 방식으로 제한조건은 functional unit의 수 또는 전체할당 실리콘 면적으로 표시된다. 스케줄링은 제한조건의 범위 내에서 데이터 의존도(dependency)가 만족되도록 하나하나의 연산을 점진적으로 스케줄 하게된다.

자원제한 조건만족 ASAP 스케줄링인 list-based 스케줄링과 ALAP와 ASAP를 적용한 단일 우선순위 list를 이용하는 static-list 스케줄링이 있다.

III. 제안된 Scheduling 알고리즘

High-level synthesis에서 스케줄링을 수행할 경우 자원제한 조건이 functional unit의 개수로 주어지는 경우에는 적당한 알고리즘을 적용하여 스케줄 할 수 있다. 하지만 제한 조건이 전체 실리콘 면적과 각 functional unit이 차지하는 면적의 크기만이 주어졌을 경우에는 해당 연산들의 functional unit의 수를 먼저 결정해야 하기 때문에 주어진 그래프를 스케줄 하는 문제는 결코 쉽지만은 않다. 실리콘 면적이 충분히 넓다면 그다지 문제가 되지 않겠지만, 일반적인 경우에서는 전체적인 비용에 영향을 주기 때문에 ASAP나 ALAP를 적용시키기 어렵거나, 최소의 비용을 가지도록 설계하는 것을 이상적으로 생각하기 때문에 각 functional unit의 수를 적절히 조절하여야 한다. 즉 각 control step에 필요한 functional unit을 구하려면 전체 control step을 알고 있어야 하고, 전체 control step를 구하려면 각 control step에서 사용되는 functional unit을 알아야 한다. 그렇기 때문에 이러한 스케줄링 문제는 매우 어렵다.

다음에서 제안하고자 하는 알고리즘은 functional unit의 수를 고정시키지 않고 변화시키면서 control step이 증가되지 않도록 반복적으로 스케줄링 하여 최소의 control step에 스케줄링 할 수 있는 최적의 해를 찾아내는 알고리즘이다.

1. 표기법

G : Data Flow Graph
A : 전체 실리콘 면적
t_k : 각 functional unit의 종류
a_{tk} : t_k가 차지하는 실리콘 면적
P^j_{tk} : 각 functional unit의 각 control step에 스케줄 될 확률
P_{tk} : 최대 P^j_{tk}
n_{tk} : 한 control step에 사용될 수 있는 각 functional

unit의 갯수
R : 각 functional unit이 할당된 후 남은 실리콘면적
C_{step} : 스케줄된 control step의 길이

2. 알고리즘

제안된 스케줄링 알고리즘은 <알고리즘 1, 2, 3>에서 보였다.

알고리즘 1. Calculation n_{tk}

Call ASAP (G)
Call ALAP (G)
for each o_i *do*
  for each j, $E_i \le j \le L_i$ *do*
  $P^j(\text{type of } o_i) = P^j(\text{type of } o_i) + \frac{1}{L_i - E_i + 1}$
  endfor
endfor
for each t_k *do*
  $p_{t_k} = \max_j P^j_{t_k}$
endfor
for each t_k *do*
  $q_{t_k} = \frac{P_{t_k} a_{t_k}}{\sum_{t_k} P_{t_k} a_{t_k}} \cdot A$
  $N_{t_k} = \left\lfloor \frac{q_{t_k}}{a_{t_k}} \right\rfloor$
endfor
$R = A - \sum_{t_k} N_{t_k} a_{t_k}$

알고리즘 2. Iterative refinement Scheduling

Prev_S_length = ∞
repeat
  INSERT_READY_OPS(V,PList$_{t1}$*,PList*$_{t2}$*,...,PList*$_{tm}$*)*
  C_{step} = 0
  while ((Plist$_{t1}$*=0) or...or (Plist*$_{tm}$*=0)) do*
    $C_{step} = C_{step} + 1$
    for k = 1 to m do
      for funit =1 to N_{tk} *do*
        if Plist$_{tk}$*=0 then*
          SCHEDULE_OP($S_{current}$*,FIRST(PList*$_{tk}$*),*C_{step}*)*
          PList$_{tk}$*=DELETE(PList*$_{tk}$*, FIRST(PList*$_{tk}$*))*
        endif
      endfor
      $R\_OP_{tk} = R\_OP_{tk} + 1$
    endfor
    INSERT READY_OPS(V,PList$_{t1}$*,PList*$_{t2}$*,...,PList*$_{tm}$*)*
  endwhile
  $R\_OP_{t_{min}} = Min_{t_k}(R\_OP_{t_k})$
  $R\_OP_{t_{max}} = Max_{t_k}(R\_OP_{t_k})$
  REALLOCATE(R, t_{min}, t_{max}*)*
  $S_{prev} = S_{current}$
until ($C_{step} \ge$ *Prev_S_length)*
$S = S_{prev}$

알고리즘 3. ReAllocate Function

REALLOCATE(R, t_{min}, t_{max}*)*
  if $R \ge a_{tmax}$ *then*
    $R = R - a_{tmax}$
  else
    $temp = a_{tmax} - R$
    $R = 0$
    repeat
      $temp = temp - a_{tmax}$
      $N_{tmin} = N_{tmin} - 1$
    until (temp ≤ 0*)*
  $N_{tmax} = N_{tmax} + 1$
return

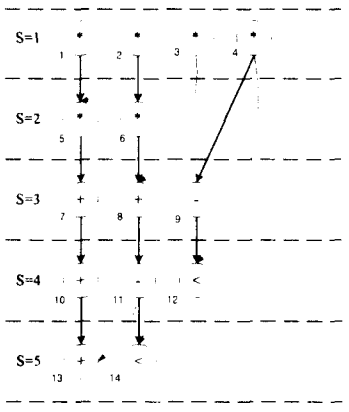


그림 2. 예제 Data Flow Graph

우선 <그림 2>의 예제 그래프를 ASAP와 ALAP 스케줄링 알고리즘을 사용하여 E_i값과 L_i값을 구한 후, 각 연산자들의 이동성(mobility)를 그린다<그림 3(a)>. 다음에 식(1),(2),(3)을 적용하여 각 functional unit의 최대 확률(Probability)을 구한다. <그림 3(b)>

$$P^j(\text{type of } o_i) = \frac{1}{L_i - E_i + 1} \quad (1)$$

$$P^j_{tk} = \sum P^j(\text{type of } o_i) \quad (2)$$

$$P_{tk} = \max P^j_{tk} \quad (3)$$

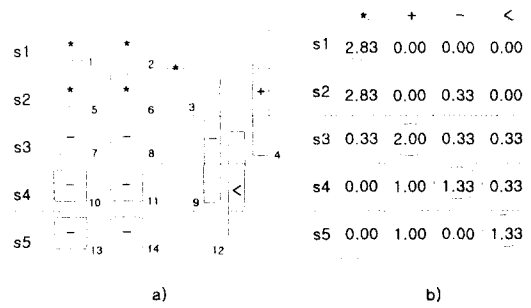


그림 3. 예제에 ASAP, ALAP를 적용한 후
(a) 연산자들에 control step에 스케줄될 확률
(b) 각 control step에서 각 연산자들의 최대 확률

각 functional unit의 최대 확률을 구한 후, 식 (4), (5)에 의하여 각 functional unit의 사용 개수 n_{tk}를 구한다.

$$q_{t_k} = \frac{P_{t_k} a_{t_k}}{\sum_{t_k} P_{t_k} a_{t_k}} \cdot A \qquad (4)$$

$$N_{t_k} = \left\lfloor \frac{q_{t_k}}{a_{t_k}} \right\rfloor \qquad (5)$$

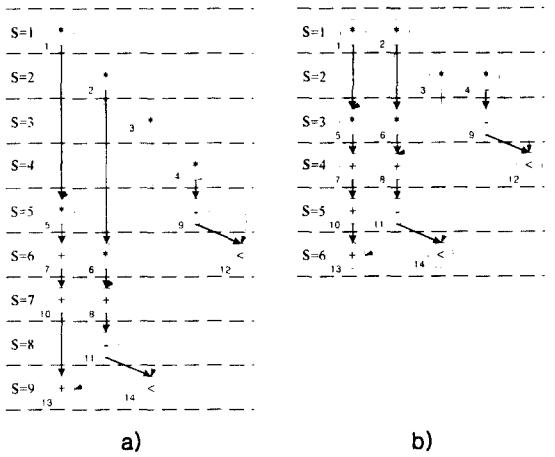


그림 4. 제안된 알고리즘의 수행 결과

여기서 a_*=2, a_+=1, a_-=1, $a_<$=1, A=8으로 가정한다면 N_*=1, N_+=2, N_-=1, $N_<$=1를 구할 수 있다(만일 N_{tk}<1 이면 N_{tk}=1). 이렇게 각 N_{tk}를 구한 다음 각 functional unit의 개수에 맞게 iterative refinement 스케줄링 알고리즘을 수행하여 스케줄링 한다. 이렇게 스케줄링 된 결과는 <그림 4(a)>와 같이 9개의 control step에 스케줄 된다. 여기서 REALLOCATION 함수를 사용하여 functional unit의 개수를 N_*=2, N_+=2, N_-=1, $N_<$=1로 수정하고 다시 스케줄링 한다. 이 과정을 control step이 증가하지 않을 동안 반복을 한다<그림 4(b)>.

Ⅳ. 결론

High-level synthesis에서 스케줄링을 수행할 경우 자원제한 조건이 전체 실리콘 면적과 각 functional unit이 차지하는 면적의 크기만이 주어졌을 경우에는 해당 연산들의 functional unit의 수를 먼저 결정해야 하기 때문에 주어진 문제를 스케줄 하는 문제는 결코 쉽지만은 않다. 본 논문에서 제안된 "효율적 자원제한 알고리즘"은 주어진 실리콘 면적에서 각 functional unit의 개수를 적절히 선택하여 주어진 스케줄링 문제에 대한 최적의 결과를 얻을 수 있다.

참고문헌

[1] Daniel D.Gajski, Nikil D.Dutt, *High-Level Synthesis Introduction to Chip and System Design*, Kluwer Academic Publishers

[2] J.Lee, Y.Hsu, and Y.Lin, "A New Integer Linear Programming Formulation for the Scheduling Problem in Data-Path Synthesis", *Proceedings of the International Conference on Computer-Aided Design*, pp.20-23, 1989.

[3] P.G.Paulin and J.P.Knight, "Force-Directed Scheduling for the Behavioral Synthesis of ASIC's", *IEEE Transactions on Computer-Aided Design of Integrated Circuites and Systems*, vol.8, no.6, pp.661-679, June 1989.

[4] I-C.Park and C-M.Kyung, "Fast and Near Optimal Scheduling in Automatic Data Path Synthesis", *Proceedings of the 28th Design Automation Conference*, pp.680-685, 1991.