論文 97-22-3-5

파이프라인을 지원하는 ASIP 합성 시스템의 설계

正會員 현 민 호*, 이 석 근**, 박 창 욱***, 황 선 영*

Design of a Cosynthesis System for Pipelined Application-Specific Instruction Processors

Min-Ho Hyun*, Seok-Goun Lee**, Chang-Wook Park***, Sun-Young Hwang* *Regular Members*

요 약

본 논문은 주어진 제약 조건을 만족하는 범위 내에서 입력 응용 프로그램에 최적화된 파이프라인 형태의 ASIP 설계를 위한 하드웨어-소프트웨어 통합 합성 시스템의 설계와 구현에 대하여 다룬다. 제안된 시스템은 상위 수준 VHDL 설계 기술을 입력으로 하여 아키텍춰 기본틀에 의해 제한된 데이타패스 설계 공간 내에서 입력 제약 조건을 만족하며 응용 프로그램의 수행에 적합한 파이프라인 형태의 ASIP 구조을 합성하고 응용 프로그램의 분석 결과 구성된 중간코드을 이용하여 생성된 ASIP에서 실행시킬 수 있는 어셈블리 코드를 생성한다.

ABSTRACT

This paper presents the prototype design of a hardware/software cosynthesis system for pipelined application-specific instruction processors. Taking application programs in VHDL as inputs, the proposed system generates a pipelined instruction-set processor and the instruction sequences running on the generated machine. The design space of datapath and controller is defined by the architectural templates embedded in the system. Generating the intermediate code adequate for parallelism analysis and extraction, the system converts it into assembly codes. Experimental results show the effectiveness of the proposed system.

Ⅰ. 서 론

최근 내장형 시스템의 응용 범위가 광범위해지고 복잡한 제어를 요하는 시스템의 설계가 요구됨에 따라 설계자의 경험에 근거한 수작업으로는 시스템의 설계가 불가능하게 되었으며, 큰 규모의 복잡한 제어를 요하는 시스템의 설계를 위하여 내장형 시스템의 하드웨어 및 소프트웨어에 대한 자동 합성기가 필요하게 되었다[1].

*서강대학교 전자공학과
**삼성항공 정밀기기 연구소
***서두로직 종합기술연구소
論文番號:96035-0201
接受日字:1996年 2月 1日

하드웨어-소프트웨어 통합 설계의 대표적인 구현 방식은 범용 프로세서와 프로세서에서 수행될 소프트웨어를 사용하는 방법과, 입력 응용 프로그램에 대하여 범용 프로세서의 성능 향상을 위한 보조 ASIC을 추가하여 시스템을 구성하는 방법, 주어진 응용 프로그램의 전용 처리를 위한 전용 프로세서(ASIP: Application-Specific Instruction Processor)와 소프트웨어를 생성하는 방법, 그리고 입력 응용 프로그램의 처리를 위한 주문형 반도체(ASIC: Application-Specific Integrated Circuit)를 설계 제작하여 사용하는 방법이 있다. 범용 프로세서와 소프트웨어를 사용하는 방법은 프로그램만의 변경으로 시스템 수정이 가능하여 유연성(flexibility)이 뛰어나지만 타 구현 방식에 비하여 시스템 성능이 떨어진다. 범용 프로세서의 사용에 따른 시스템의 성능 저하는 보조 ASIC을 사용함으로써 보완할 수 있으나 이는 보조 ASIC 및 이들간의 인터페이스를 위한 코스트가 증가하는 단점이 있다. ASIC을 이용한 전체 시스템의 구현 방법은 가장 좋은 성능을 얻을 수 있으나 시스템의 유연성이 떨어지고 긴 개발 기간을 필요로 한다[2]. 전용 프로세서 설계 방식은 시스템의 성능을 크게 희생하지 않으면서 주어진 입력 응용 프로그램의 수행에 뛰어난 성능을 발휘하며 한 응용 분야 내에서 부분적인 기능 변경에 대하여 프로그램의 변경만으로 시스템 수정이 가능하여 유연성이 뛰어난 특성을 가지고 있다. 그러나, 전용 프로세서의 개발은 입력 응용 프로그램에 대한 하드웨어와 소프트웨어가 동시에 구현되어야 하므로, 다른 설계 방식보다 많은 개발 시간과 비용이 필요하며, 이를 줄이기 위한 자동 설계 시스템의 개발이 절실히 요구된다[3].

지금까지의 전용 프로세서 자동 설계 시스템에 대한 연구는 대부분 framework 구성을 위한 방향 제시 및 소프트웨어나 하드웨어 영역에서의 부분적인 결과들에 국한되었으나[4][5][6], 최근 소프트웨어 영역과 하드웨어 영역을 포함한 통합 설계 시스템의 구현에 관한 연구 결과가 보고되고 있다. Alomary 등에 의해 제안된 시스템은[7] 하드웨어 구현에 필요한 인스트럭션 셋의 선택 시 데이타패스의 면적과 전력 소모를 최소화 하는 셋을 선택하여, 선택된 인스트럭션 셋에 대응되는 하드웨어 블록을 데이타패스를 구성하고 있는 커널에 포함시킴으로써 최적화된 시스템을 구현하는 것을 목적으로 하고 있으며, 데이타패스 구현 시 파이프라인은 고려하지 않았다. Huang 등에 의해 제안된 시스템은[8] 입력 응용 프로그램을 분석하고 상호 의존성 관계에 있는 기본적인 연산들을 스케쥴링 함으로써 병렬 처리가 가능한 인스트럭션 셋을 구성하며, 시스템 내부에 포함하고 있는 다양한 종류의 파이프라인 형태에 따라 입력 응용 프로그램에 최적화된 하드웨어 구조를 구현하였으나, 다양한 인스트럭션 포맷을 갖게 되므로 복잡한 제어 신호와 제어 블록을 필요로 하여 하드웨어의 자동 생성이 어려워 지는 단점이 있다.

본 논문에서는 VHDL 형태의 입력 응용 프로그램을 분석하여 VHDL 입력 기술의 구현에 적합한 인스트럭션 셋을 선택한 후 선택된 인스트럭션의 수행에 적합한 파이프라인 형태를 구성하며 병렬 처리가 가능한 시스템을 합성하는 하드웨어-소프트웨어 통합 설계 시스템의 구현에 대하여 기술한다. Ⅱ 장에서는 하드웨어-소프트웨어 통합 설계 시스템의 전체적인 구성을 기술하였고, Ⅲ 장에서는 하드웨어-소프트웨어 통합 설계 시스템에서 인스트럭션 셋 선정을 위한 기본 연산자 매핑 과정에 대하여 기술하였으며, Ⅳ 장에서는 입력 응용 프로그램 분석 및 중간 코드 합성 과정을 기술하였다. Ⅴ 장에서는 하드웨어-소프트웨어 통합 설계 시스템에서 제공하는 하드웨어 프로토타입인 아키텍춰 기본틀의 종류와 형태를 제시하였고, 중간 코드를 분석하여 합성할 하드웨어의 스팩을 생성하는 과정과 합성될 하드웨어에서 실행될 어셈블리 코드를 생성하는 과정을 기술하였다. Ⅵ 장에서는 실험 결과를 통하여 제안된 하드웨어-소프트웨어 통합 설계 시스템에서 합성한 ASIP와 타 상용 프로세서 및 전용 ASIC과의 성능을 비교하였으며, Ⅶ 장에서는 결론 및 추후 과제를 제시하였다.

Ⅱ. 시스템 개관

그림 1에 제안된 하드웨어-소프트웨어 통합 설계 시스템의 구성도를 보였다. 중간 코드 생성기는 주어진 VHDL 입력 기술로부터 C/DFG(Control/Data Flow Graph)[9]를 생성하여 중간 코드를 합성하며, 중간 코드는 합성될 타겟 아키텍춰와 어셈블리 코드의 구성을 위한 정보를 포함한다. 아키텍춰 기본틀은 인스트

럭션 파이프라인의 형태와 stage 수에 따라 세분화되며, 매개 변수를 이용하여 데이타패스 기능 유닛의 선택 및 구조 조정이 가능한 유연한 형태로 제공한다. 중간 코드 분석기는 중간 코드 분석을 통하여 VHDL 입력 응용 프로그램의 수행에 적합한 인스트럭션 셋을 선택하고 아키텍처 기본틀의 매개 변수 값을 결정하여 선택된 인스트럭션 셋의 수행을 위한 ASIP 하드웨어 구조를 합성 할 수 있도록 한다. 타겟 아키텍춰 합성기에서는 결정된 아키텍춰 기본틀의 매개 변수 값을 이용하여 타겟 아키텍춰를 완성하고, 타겟 코드 생성기에서는 중간 코드를 생성된 타겟 아키텍춰에서 사용할 수 있는 어셈블리 코드로 변환하는 과정을 수행한다. 합성 시스템은 인스트럭션 셋의 생성과 타겟 아키텍춰 합성 시 사용자 인터페이스를 통한 설계자의 매개 변수 제어가 가능한 구조로 구성되어 있다.

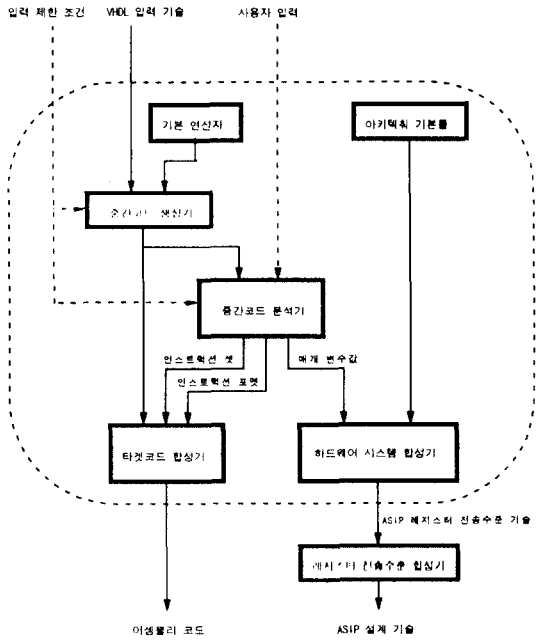


그림 1. 시스템 구성도
Fig. 1 Design flow of the proposed system

Ⅲ. 기본 연산자 매핑

제공되는 아키텍춰 기본틀에서 매개 변수의 조정에 따라 수행 가능한 모든 연산들에 대한 연산자를 기본 연산자라 정의하고, VHDL 입력 응용 프로그램으로부터 생성된 중간 코드의 연산 이용 빈도 분석 결과에 따라 VHDL 프로그램의 각 연산을 기본 연산자로 매핑하여 합성될 ASIP의 인스트럭션 셋을 구성한다. VHDL 기술에서 지원되는 모든 연산들은 각각 기본 연산자 혹은 기본 연산자 조합과 일대일 매핑이 가능하여야 하며, 이용 빈도가 높은 연산일수록 기본 연산자 매핑에 우선 순위를 갖고 이용 빈도가 낮은 연산들은 가능한 한 기존의 매핑된 기본 연산자들의 조합을 통해 구현하도록 하여 ASIP의 인스트럭션 셋을 최적화한다. 그림 2에 기본 연산자 매핑 알고리듬을 보였다.

```
OP : a set of operations in application program;
B_OP : a set of all basic operators supported in the architecture template;
B_OP_SEL : ∅; /* a set of basic operators mapped */
Analyze the # of execution for each operation in OP;
Sort OP in descending order of the # of execution;
For each operation OP in OP {
  Map OP to the combination of one or more basic operators in B_SEL_OP;
  if mapping fails {
    Map OP to the combination of one or more basic operators in B_OP;
    Insert the mapped basic operator(s) into B_OP_SEL;
```

그림 2. 기본 연산자 매핑 알고리듬
Fig. 2 Mapping algorithm for basic operators

기본 연산자 셋은 RISC-II[10], MIPS[11], Intel 80 계열[12], 모토로라 MC68020[13]의 인스트럭션 셋과 상위 수준 VHDL 설계 기술 벤치마크를 분석하여 결정하였으며, 기본 연산자는 기능에 따라 산술 및 논리 연산자, 메모리 연산자, 분기, 기타 연산자 등으로 나눌 수 있다. 표 1의 (a)는 산술 및 논리 연산자, (b)는 메모리 연산자, (c)는 분기, (d)는 move의 종류와 형식을 보인다.

Ⅳ. 중간 코드 합성

상위 수준 VHDL 설계 기술 형태로 주어진 입력 응용 프로그램은 최적화된 설계 해를 찾아내기 위한 분석 과정과 타겟 아키텍춰에서 수행될 어셈블리 코드 생성을 위하여 중간 코드로 변환된다. 중간 코드

표 1. 기본 연산자의 종류와 형식
(a) 산술 및 논리 연산자의 종류와 형식
(b) 메모리 연산자의 종류와 형식
(c) 분기의 종류와 형식 (d) Move의 종류와 형식

Table 1. Format of basic operators
(a) Arithmetic/logic operator
(b) Memory management operator
(c) Branch operator (d) Move operator

연산명	형식	수행 내용
add	add Rt, Rs1, Rs2	Rt <= Rs1 add Rs2
sub	sub Rt, Rs1, Rs2	Rt <= Rs1 sub Rs2
mul	mul Rt, Rs1, Rs2	Rt <= Rs1 mul Rs2
and	and Rt, Rs1, Rs2	Rt <= Rs1 and Rs2
inv	inv Rt, Rs	Rt <= complement value of Rs
or	or Rt, Rs1, Rs2	Rt <= Rs1 or Rs2
xor	xor Rt, Rs1, Rs2	Rt <= Rs1 xor Rs2
sl	sl Rt, Rs, Const,	Rt <= Rs Const, bit shift left
sr	sr Rt, Rs, Const,	Rt <= Rs Const, bit shift right

(a)

연산명	형식	수행 내용
store	store Rta,Rs	mem[Rta] <= Rs
	store $Addr, Rta	mem[$Addr] <= Rs
	store Rta,disp,Rs	mem[Rta+disp] <= Rs
load	load Rt,Rsa	Rt <= mem[Rsa]
	load Rt, $Addr	Rt <= mem[$Addr]
	load Rt,Rsa,disp	Rt <= mem[Rsa+disp]

(b)

연산 종류	연산명	형식	분기 조건
Conditional Branch	je	je Rs, Rt, Immediate[address]	Rs = Rt
	jei	jei Rs, Immediate, Immediate[address]	Rs = Immediate
	jne	jne Rs, Rt, Immediate[addrss]	Rs /= Rt
	jnei	jnei Rs, Immediate, Immediate[address]	Rs /= Immediate
	jg	jg Rs, Rt, Immediate[address]	Rs > Rt
	jgi	jgi Rs, Rt, Immediate[address]	Rs > Immediate
	jge	jge Rs, Rt, Immediate[address]	Rs >= Rt
	jgei	jgei Rs, Immediate, Immediate[address]	Rs >= Immediate
	jl	jl Rs, Rt, Immedate[address]	Rs < Rt
	jli	jli Rs, Immediate, Immediate[address]	Rs < Immediate
	jle	jle Rs, Rt, Immediate[address]	Rs <= Rt
	jlei	jlei Rs, Immediate, Immediate[address]	Rs <= Immediate
Unconditional Branch	jmp	jmp Immediate	

(c)

연산명	형식	수행 내용
move	move Rt, Rs	Rt <= Rs
movei	movei Rt, disp	Rt <= disp
movepi	movepi Rt, $Addr	Input port로 부터 데이타 입력
movepo	movepo $Addr, Rs	Output port로 데이타 출력

(d)

는 수행되어야 할 기본 연산자를 표시하기 위한 연산자 필드와 오퍼랜드 필드, 병렬 처리를 위하여 처리 단계를 표시하는 필드 등으로 구성되며, 기본 연산자를 이용하여 구성된 중간 코드는 합성될 ASIP의 인스트럭션으로 일대일 변환이 가능하다.

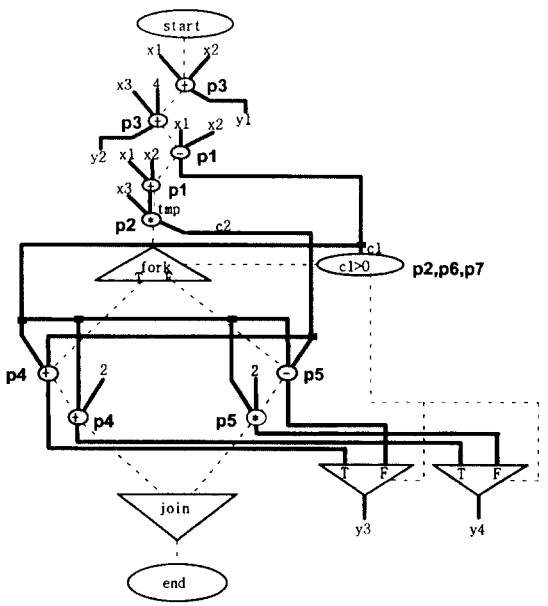
```
entity exp is
  port( x1, x2, x3 : in integer;
        y1, y2, y3, y4: out integer);
end exp;

architecture exp of exp is
begin
  y1 <= x1+x2;
  y2 <= x3+4;
  process
  variable c1, c2:integer;
  begin
    c1 := x1-x2;
    c2 := (x1+x2)*x3;
    if c1>0 then
      y3 <= c1+c2;
      y4 <= c1+2;
    else
      y3 <= c1-c2;
      y4 <= c2*2;
    end if;
  end process;
end exp;
```

(a)

```
          p1 add tmp x1 x2
          p1 sub c1 x1 x2
          p2 mul c2 tmp x3
          p2 jg c1 0 L_true
          p6 jmp L_false
L_true:   p4 add y3 c1 c2
          p4 add y4 c1 2
          p7 jmp L_exit
L_false:  p5 sub y3 c1 c2
          p5 mul y4 c1 2
L_exit:   p3 add y1 x1 x2
          p3 add y2 x1 4
```

(c)



(b)

그림 3. VHDL 입력 응용 프로그램의 중간 코드 변환 예
(a) VHDL 입력 응용 프로그램 (b) C/DFG
(c) 중간 코드

Fig. 3 Transformation of application input into intermediate code
(a) Application input in VHDL (b) C/DFG
(c) Intermediate code

그림 3은 VHDL 입력 기술의 C/DFG 변환 및 기본 연산자 매핑 과정의 수행 후 FDS 스케줄링 알고리듬[14]을 이용하여 생성된 중간 코드의 예이다. 그림 3 (a)의 VHDL 입력 응용 프로그램에 대한 C/DFG를 그림 3의 (b)에서 보였고, 기본 연산자 매핑 과정을 수행한 후 이를 FDS 스케줄링 알고리듬을 이용하여 중간 코드로 변환한 결과를 그림 3의 (c)에 나타냈다. 그림 3의 (b)에서 점선은 콘트롤 흐름을 나타내고 실선은 데이타 흐름을 나타내며, 제어 구간의 수는 7로 주어졌다. 중간 코드의 한 연산은 기본 연산자 매핑 과정을 거친 C/DFG의 모든 노드와 일대 일로 대응되며, 한 제어 구간 내에 존재하는 연산 노드들은 중간 코드로 변환 시 병렬 처리 연산 필드에 같은 레이블을 갖는다.

V. 중간 코드 분석 및 시스템 합성

5.1 아키텍춰 기본틀

아키텍춰 기본틀은 매개 변수화된 유연한 구조의 레지스터 전송 수준 VHDL 기술로 이루어져 있으며, 중간 코드 생성 후 중간 코드 분석기에서 제공되는 매개 변수값에 따라 기본 연산자 셋에서 정의된 연산자 및 파이프라인의 형태를 선택적으로 지원하는 아키텍춰를 제공한다. 제공되는 아키텍춰 기본틀은 인스트럭션 메모리와 데이타 메모리를 분리하는 하바드 아키텍춰[15]를 지원하며, 산술 및 논리 연산에서 메모리 변수를 사용할 수 있도록 전용 하드웨어를 사용한다[3].

아키텍춰 기본틀은 매개 변수에 따라 그림 4에 보인 4가지 형태의 파이프라인을 제공할 수 있으며, 아키텍춰 기본틀에서 제공하는 정형화된 데이타패스 상에서 면적은 파이프라인의 단 수와 비례하므로 제공하는 4가지 형식 중 클럭 주기에 대한 제약 조건을 만족하는 범위 내에서 최소의 면적을 갖는 형식을 선택한다. 파이프라인 형식의 선택은 그림 5에서 제시한 방법으로 이루어진다. 주어진 입력 응용 프로그램의 수행을 위한 ASIP을 5단 파이프라인 형식으로 구현했을 때 각각의 파이프라인 단계에 대한 지연 시간 t1 (IF-stage delay), t2 (ID & OF-stage delay), t3 (EX-stage delay), t4(MEM-stage delay), t5(WB-stage delay)를 구하고, 그림 4의 (a), (b)에서 사용되는 IF & ID &

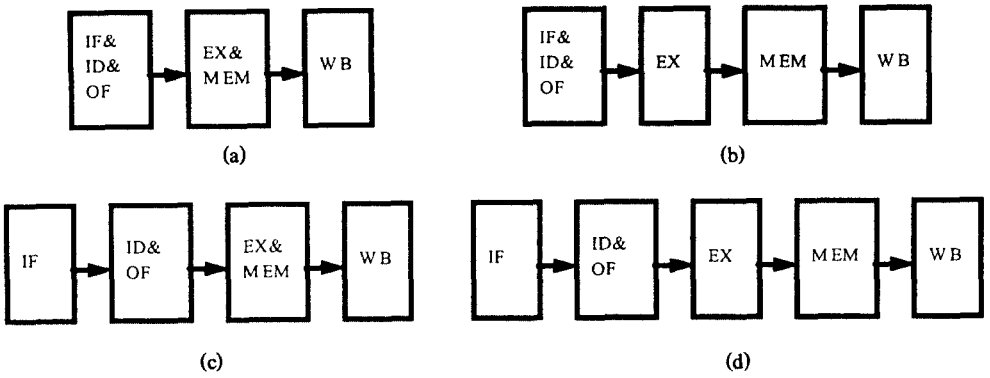


그림 4. 아키텍춰 기본틀의 파이프라인 형태.
(a)3단 파이프라인(형식1) (b)4단 파이프라인(형식2)
(c)4단 파이프라인(형식3) (d)5단 파이프라인(형식4)

Fig. 4 Architecture templates for pipelined datapath.
(a)3-stage pipeline(Type1) (b)4-stage pipeline(Type2)
(c)4-stage pipeline(Type3) (d)5-stage pipeline(Type4)

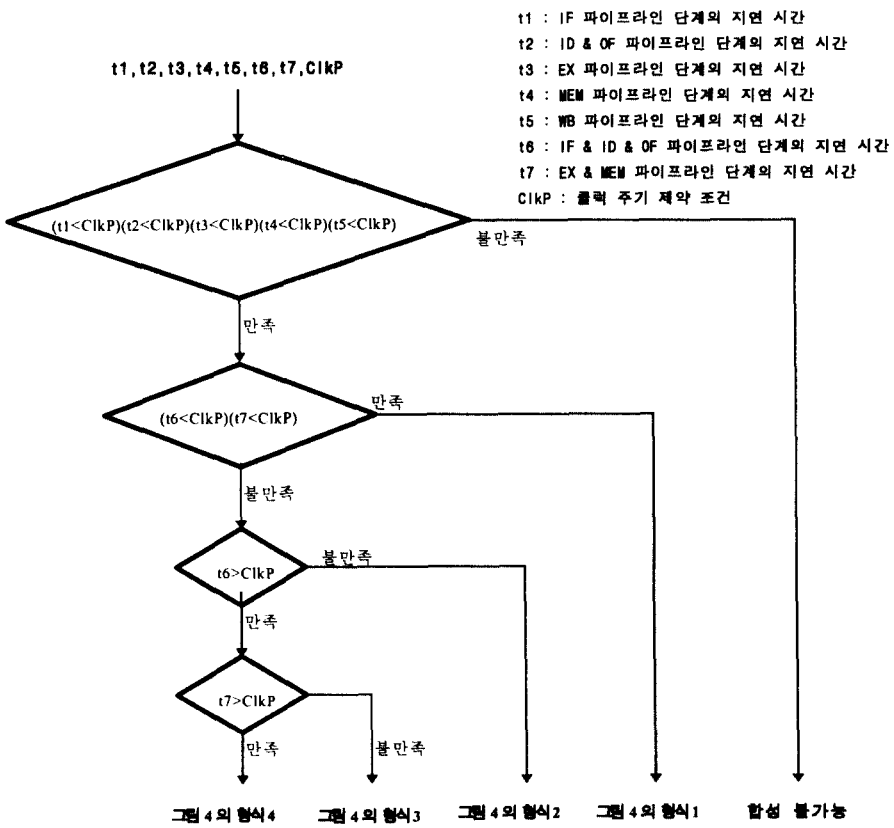


그림 5. 파이프라인 형태 결정 방법
Fig. 5 Method of pipeline style decision

OF 파이프라인 단계의 지연 시간 t6, 그림 4의 (a), (c)에서 사용되는 EX & MEM 파이프라인 단계의 지연 시간 t7을 구한다. 클럭 주기에 대한 제약 조건을 ClkP로 정의할때 아키텍춰 기본틀에서 제공하는 파이프라인 형식으로 클럭 주기에 대한 제약 조건을 만족하는 설계해를 구하기 위하여는 t1, t2, t3, t4, t5가 모두 ClkP 보다 적어야 한다. 이 조건을 만족했을 때 t6, t7이 모두 ClkP보다 적은 경우에는 그림 4의 파이프라인 형식1이 선택되고, t6, t7이 모두 ClkP보다 큰 경우에는 그림 4의 파이프라인 형식4가 선택된다. ClkP가 t6보다 적고 t7보다 큰 경우에는 그림 4의 파이프라인 형식2가 선택되고, ClkP가 t6보다 크고 t7보다 적은 경우에는 그림 4의 파이프라인 형식3이 선택된다.

시스템의 throughput은 각 파이프라인 수행 단계 중 가장 큰 지연 시간을 갖는 단계에 의하여 결정되므로, 파이프라인 형태의 선택은 최대 지연 시간을 갖는 파이프라인 단계에 의해 결정된다. 중간 코드에 사용된 연산 중 큰 지연 시간을 요하는 곱셈기 연산이 사용되거나 메모리 참조 연산이 많은 경우, t7이 큰 값이 된다. 사용되는 연산의 종류가 많고 레지스터의 수가 많은 경우 인스트럭션 디코딩과 오퍼런드 fetch에 큰 지연 시간을 필요로 하므로 t6이 큰 값이 된다.

5.2 중간 코드 분석기

중간 코드의 분석은 입력 제약 조건을 만족하는 타겟 아키텍춰의 구현을 위하여 연산기의 종류 및 갯수, bitwidth, 버스와 MUX의 갯수 등 아키텍춰 기본틀을 구성하는 속성 요소들의 매개 변수값을 결정하는 과정으로, 중간 코드 상의 기본 연산과 병렬 처리 필드를 분석하여 각 기본 연산들을 실제 연산기에 할당하고 이의 구현을 위하여 아키텍춰 기본틀의 속성 요소들에 대한 값을 결정한다. 타겟 아키텍춰에 대한 비용은 연산기 할당 후 결정된 연산기의 종류별 갯수 및 버스, MUX, 레지스터의 갯수 등에 따라 예상한다.

입력 제약 조건으로는 입력 응용 프로그램 수행에 필요한 총 제어 구간 수과 합성될 ASIP의 면적, 클럭 주기가 주어진다. 비용 예상 시 생성된 면적, 클럭 주기에 대한 예상값이 제약 조건을 만족하지 않을 경우에는 제약 조건을 만족하는 해를 찾을 수 없고 만족하는 경우에는 결정된 매개 변수 값으로 ASIP의 하드웨어 블록과 입력에 대한 어셈블리 코드를 생성한다. 그림 6은 중간 코드 분석 과정을 보인다.

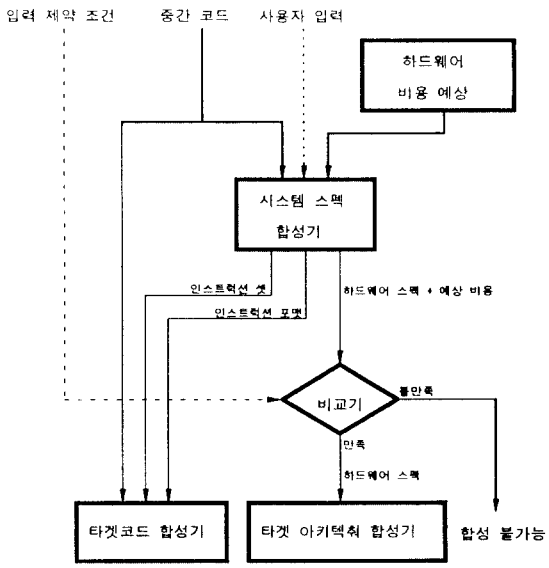


그림 6. 중간 코드 분석기
Fig. 6 Intermediate code analyzer

타겟 아키텍춰에서 사용할 인스트럭션 포맷은 opcode와 operand의 크기가 정해진 후 결정할 수 있다. 인스트럭션 포맷의 opcode의 크기는 중간 코드에서 사용한 연산들을 모두 표현할 수 있는 bitwidth로 결정하고, operand의 크기는 데이타 메모리와 레지스터 파일의 어드레스 공간을 표현할 수 있는 bitwidth와 중간 코드에서 사용한 상수값의 bitwidth를 고려하여 결정한다.

5.3 타겟 아키텍춰 합성기

타겟 아키텍춰 합성기는 중간 코드 분석 결과 결정된 매개 변수의 값을 이용하여 각 하드웨어 블록과 타겟 아키텍춰를 합성하는 과정을 수행한다. 하드웨어 블록은 인스트럭션 메모리, 데이타 메모리, ALU, 전용 연산기, 제어 블록 등으로 구성되며, 하드웨어 블록의 합성은 입출력 변수의 크기와 내부 기능이 매개 변수로 정의된 각 블록의 레지스터 전송 수준 설계 기술을 가지고 수행하며, 제공되는 아키텍춰 기본

틀의 각 속성 요소들을 결정하여 합성된 ASIP 타겟 아키텍춰를 레지스터 전송 수준 VHDL 기술로 출력한다.

5.4 타겟 코드 생성기

타겟 코드 생성기는 응용 프로그램으로부터 생성된 중간 코드에 대하여 결정된 레지스터의 갯수에 따라 레지스터를 할당하고 메모리 참조 연산을 삽입한 후 코드 최적화 과정을 통하여 ASIP에서 수행될 어셈블리 코드로 변환하는 과정을 수행한다. 코드 최적화 과정은 데이타 의존성의 유무에 따라 각각 다른 방법으로 처리된다. 데이타 의존성이 있는 연산들의 경우는 연산들을 통합하여 새로운 연산을 생성하는 방법으로 코드의 크기를 줄이고[3], 데이타 의존성이 없는 연산들의 경우는 연산들의 병렬 처리를 통하여 코드의 실행 시간을 줄인다[16]. 데이타 의존성이 있는 연산들의 통합에 의한 코드 최적화는 이에 따른 하드웨어의 증가는 적으나, 인스트럭션의 종류가 증가하므로 인스트럭션 포맷이 복잡해져 제어 블록의 자동 생성이 어려워지게 되므로 인스트럭션 포맷의 consistency와 orthogonality를 유지할 수 있는 범위 내에서 수행된다.

Ⅵ. 실험 결과

본 논문에서 구현된 시스템에 대한 실험은 8-point FFT(Fast Fourier Transform)과 EWF(Elliptical Wave Filter)에 대하여 수행하였다. 표 2는 본 합성 시스템으로 합성한 ASIP과 MIPS[17], SPARC[18]와의 성능 비교 결과를 나타낸 것으로, 합성 결과는 Synopsys Design Analyzer v3. 3을 사용하였고, 타겟 라이브러리는 LSI 10K를 사용하였다. 합성된 ASIP는 입력 설계 기술에 대한 수행 시간이 FFT의 경우 MIPS와 SPARC에 대하여 각각 17.5%, 16.2% 줄어들고, EWF의 경우 각각 53.0%, 55.6% 줄어드는 것을 표 2에서 확인할 수 있다.

Ⅶ. 결론 및 추후 과제

본 논문에서는 상위 수준 VHDL 설계 기술로 표현된 입력 응용 프로그램을 분석하여 입력 제약 조건을 만족하는 ASIP를 자동 설계하는 하드웨어 소프트웨어 통합 설계 시스템의 구현에 대하여 기술하였다. ASIP 타겟 아키텍춰의 합성은 아키텍춰 기본틀로 제한된 설계 공간상에서 제약 조건을 만족하는 설계해를 찾는 과정으로 수행되고, 타겟 아키텍춰에서 수행될 어셈블리 코드는 VHDL 입력 기술을 변환하여 생성하였다. 아키텍춰 기본틀에서 제공하는 설계 공간은 시스템의 병렬 처리를 위한 버스, MUX 및 전용 연산기의 종류와 갯수, ALU의 기능, 생성되는 데이타패스의 파이프라인 형태 등으로 주어진다.

추후 과제로는 곱셈기 등 지연 시간이 큰 연산기들에 대하여 파이프라인화 된 연산기의 사용이 고려되어야 하며, 아키텍춰 기본틀이 제공하는 설계 공간의 확장을 위하여 많은 DSP 응용 프로그램에 사용되는 MAC 연산이나, FIFO 구조의 입출력 버퍼 등 다양한 하드웨어 블록을 지원하여야 한다.

표 2. 생성된 ASIP와 MIPS, SPARC의 성능 비교
Table 2. Performance compare result of synthesized ASIP and MIPS/SPARC

Benchmark		ASIP	MIPS(MIPS-X)	SPARC(microSPARC)
FFT	어셈블리 코드 크기	298	502	617
	Clock 주기(ns)	57.31	50	40
	수행 시간(ns)	20,689	25,100	24,680
	파이프라인 형태	그림 4의 형식2		
EWF	어셈블리 코드 크기	117	112	148
	Clock 주기(ns)	19.20	50	40
	수행 시간(ns)	2,630	5,600	5,920
	파이프라인 형태	그림 4의 형식4		

참 고 문 헌

1. W. Wolf, "Hardware-Software Co-Design of Embedded Systems," Proceedings of the IEEE, Vol. 82, No. 7, pp. 967-969, July 1994.
2. C. Liem, T. May, and P. Paulin, "Register Assignment through Resource Classification for ASIP Microcode Generation," in Proc. Int'l Conf. on CAD, pp. 397-398, Nov. 1994.
3. I. Huang, A. Despain, "Generating Instruction Sets and Microarchitectures from Application," in Proc. Int'l Conf. on CAD, pp. 391-392, Nov. 1994.
4. J. Sun, R. Brodersen, "Design of System-Level Modules," in Proc. Int'l Conf. on CAD, pp. 478-481, Nov. 1992.
5. G. Menez, R. Ortega, "A Partitioning Algorithm for System-Level Synthesis," in Proc. Int'l Conf. on CAD, pp. 482-487, Nov. 1992.
6. M. Corazao, M. Khalaf, "Instruction Set Mapping for Performance Optimization," in Proc. Int'l Conf. on CAD, pp. 518-521, Nov. 1993.
7. A. Alomary, T. Nakata, "An ASIP Instruction Set Optimization Algorithm with functional Module Sharing Constraint," in Proc. Int'l Conf. on CAD, pp. 526-532, Nov. 1993.
8. I. Huang, A. Despain, "Hardware/Software Resolution of Pipeline Synthesis of Instruction Set Processors," in Proc. Int'l Conf. on CAD, pp. 391-396, Nov. 1994.
9. D. Ku, G. De Micheli, "High Level Synthesis of ASICs under Timing and Synchronization Constraints," Kluwer Academic Pub., pp. 62-68, 1992.
10. M. Katevenis, "Reduced Instruction Set Computer Architectures for VLSI," MIT Press, pp. 43-52, 1985.
11. G. Kane, "MIPS R2000 RISC Architecture," Prentice Hall Inc.: Englewood cliffs, NJ, 1987.
12. B. Brey, "The Advanced Intel Microprocessors," Macmillan Pub., 1993.
13. A. van de Goor, "Computer Architecture and Design," Addison Wesley Pub.: Reading, MA, 1989.
14. P. Paulin, J. Knight, "Force-Directed Scheduling for the Behavioral Synthesis of ASIC's," IEEE Trans. on CAD/ICAS, Vol. CAD-8, No. 6, pp. 661-679, July 1989.
15. B. Wilkinson, "Computer Architecture," Prentice Hall Inc.: Englewood cliffs, NJ, 1991.
16. K. Hwang, "Advanced Computer Architecture," McGraw-Hill, 1993.
17. P. Chow, "MIPS-X Instruction Set and Programmer's Manual," Technical Report CSL-86-289, Computer Systems Lab., Stanford University, May 1986.
18. B. Case, "SPARC Hits Low End with TI's microSPARC," Microprocessor Report, Vol. 6, No. 14, Oct. 28, pp. 11-14, 1992.

현 민 호(Min-Ho Hyun) 정회원

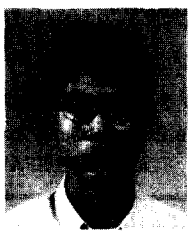
1968년 11월 14일생

1991년 2월 : 서강대학교 전자공학과 졸업(공학사)

1993년 2월 : 서강대학교 대학원 전자공학과 석사과정 졸업(공학석사)

1993년 3월~현재 : 서강대학교 대학원 전자공학과 박사과정 재학중

※주관심분야 : 시스템 합성, VLSI 설계, multimedia 신호처리 등



이 석 근(Seok-Goun Lee)정회원

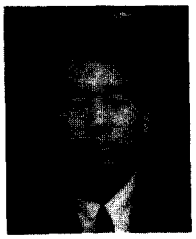
1970년 3월 19일생

1994년 2월 : 서강대학교 전자공학과 졸업(공학사)

1996년 2월 : 서강대학교 대학원 전자공학과 석사과정 졸업(공학석사)

1996년 3월~현재 : 삼성항공 정밀기기 연구소 주임 연구원

※주관심분야 : VLSI CAD, 신호처리 등



박 창 욱(Chang-Wook Park) 정회원

1970년 11월 7일생
1993년 2월 : 서강대학교 전자공학과 졸업(공학사)
1995년 2월 : 서강대학교 대학원 전자공학과 석사과정 졸업(공학석사)
1997년 2월 : 서강대학교 대학원 전자공학과 박사과정 수료
1996년 3월~현재 : 서두로직 종합기술 연구소 연구원
※주관심분야 : 컴퓨터 아키텍처, 시스템 합성 등

황 선 영(Sun-Young Hwang) 정회원

1976년 2월 : 서울대학교 전자공학과 졸업
1978년 2월 : 한국과학원 전기 및 전자공학과 공학석사 취득
1986년 10월 : 미국 Stanford 대학 공학박사 학위 취득
1976년~1981년 : 삼성 반도체 주식회사 연구원
1986년~1989년 : Stanford 대학 Center for Integrated Systems 연구소 책임연구원, Fairchild Semiconductor Palo Alto Research Center 기술자문.
1989년~1992년 : 삼성전자(주) 반도체 기술자문
1989년 3월~현재 : 서강대학교 전자공학과 부교수
※주관심분야 : CAD 시스템, Computer Architecture 및 Systems Design, VLSI 설계 등임.