논문 2006-43CI-4-3

광프로세서를 위한 효율적인 제어회로 설계 및 검증

(A Design and Verification of an Efficient Control Unit for Optical Processor)

이 원 주*

(Won Joo Lee)

요 약

본 논문은 $LiNbO_3$ 광스위칭 소자를 이용한 광컴퓨터 시스템인 SPOC(Stored Program Optical Computer)의 제어 동작의 문제점을 개선한 회로를 설계하고 검증한다. SPOC의 메모리는 DLM(Delay Line Memory) 구조이고, 오퍼런드가 필요 없는 명령어도 메모리 접근 과정을 수행하기 때문에 메모리 접근에 많은 시간이 소요되는 문제점이 있다. 또한 원하는 연산만을 선택하여 수행할 수 없기 때문에 산술논리장치에서 불필요한 연산이 많이 수행된다.

따라서 본 논문에서는 오퍼런드를 찾기 전에 미리 명령어를 해독함으로써 오퍼런드가 필요 없는 명령어의 메모리 접근을 제거하도록 회로를 개선한다. 또한 산술논리장치내의 모든 연산회로에 오퍼런드를 보내지 않고 특정 연산회로에만 오퍼런드를 보냄으로써 불필요한 연산을 줄인다. 그리고 DIR(Dual Instruction Register) 구조를 제시하여 전체 프로그램의 실행시간을 최소화 한다.

Abstract

This paper presents design and verification of a circuit that improves the control-operation problems of Stored Program Optical Computer (SPOC), which is an optical computer using $LiNbO_3$ optical switching element. Since the memory of SPOC takes the Delay Line Memory (DLM) architecture and instructions that are needless of operands should go though memory access stages, SPOC memory have problems; it takes immoderate access time and unnecessary operations are executed in Arithmetic Logical Unit (ALU) because desired operations can't be selectively executed.

In this paper, improvement on circuit has been achieved by removing the memory access of instructions that are needless of operands by decoding instructions before locating operand. Unnecessary operations have been reduced by sending operands to some specific operational units, not to all the operational units in ALU. We show that total execution time of a program is minimized by using the Dual Instruction Register(DIR) architecture.

Keywords : $LiNbO_3$ (Lithium Niobate), SPOC(Stored Program Optical Computer), DLM (Delay Line Memory), DIR (Dual Instruction Register)

I. 서 론

고성능 컴퓨터 개발은 반도체 제조 기술에 의존한 고성능 프로세서 개발에 집중되어 왔다. 이러한 노력은 반도체 칩의 집적도가 향상되고 프로세서를 구성하는 소자의 수가 증가하면서 발생하는 전자적 간섭 현상과 높은 열, 과다한 소비전력 등의 문제점 때문에 점차 한계점에 이르고 있다[1]. 이러한 한계점을 벗어나기 위한 하나의 방법으로 광소자를 사용한 광컴퓨터 개발이다.

광컴퓨터에는 $LiNbO_3$ (lithium niobate), GaAs, S-SEED 등의 광소자들이 사용된다[2]. 4-비트 카운터[3], SLM (spatial light modulator)을 사용한 비트 슬라이스 전가산기[4]와 $LiNbO_3$ 광스위칭 소자를 사용하여 초보적인 비트단위 직렬 (bit-serial) 방식으로 동작하는 SPOC(stored program optical computer) 광컴퓨터가 개발되었다[5-9]. 또한 광소자와 전자회로를 접목하여 이진논리 벡터/행렬 승산이 가능한 프로세서로 DOCII

* 정회원, 두원공과대학 인터넷프로그래밍과
(Department of Internet Programming, Doowon Technical College)

※ 본 연구는 2005년 두원공과대학 학술연구과제지원 사업의 지원으로 수행되었습니다.
접수일자: 2006년1월2일, 수정완료일: 2006년6월27일

(second-generation digital opticalelec_tronic computer)를 개발하였다[10]. 그리고 DOCII를 확장하여 단위 소비 전력이 낮고, 성능이 향상된 HPOC (high performance optoelectronic computing)모듈을 개발하였다[11, 12]. 또한 $LiNbO_3$ 광스위칭 소자를 사용하여 메모리, 멀티플렉서, 승산기 등의 필수적인 논리연산회로를 설계하고, 이 회로들을 이용하여 간단한 구조의 ALU를 설계하였다[13].

본 논문에서는 광컴퓨터인 SPOC 제어회로의 문제점을 개선한 새로운 제어회로를 설계하고 검증한다. 또한 DIR(dual instruction register)을 사용하여 전체 프로그램 수행시간을 단축한다. 즉, 현재 PC가 지시하는 명령어와 다음 명령어를 함께 인출하여 첫 번째 명령어는 IR1에 두 번째 명령어는 IR2에 적재함으로써 두 번째 명령어를 찾는 시간과 IR에 적재하는 시간을 단축한다.

본 논문의 제 II장에서는 기존의 광컴퓨터인 SPOC의 수행과정과 동작상의 문제점을 설명한다. 그리고 제 III장에서는 새로운 제어회로의 구성과 그 동작에 대하여 상세히 설명한다. 그리고 다중 명령어 레지스터 구조에 대하여 설명한다. 제 IV장에서는 구현된 제어회로의 동작을 검증하고 결과를 분석하고, 제 V장에서 결론을 맺는다.

II. 관련 연구

1. SPOC 구조

$LiNbO_3$ 광스위칭 소자를 사용한 광컴퓨터인 SPOC의 구조는 그림 1과 같다. SPOC는 산술논리장치에서 한 번에 한 비트씩 처리하는 비트단위 직렬방식으로 동작한다. SPOC의 메모리는 일반 컴퓨터와 다른 DLM (delay line memory)이라는 직렬 메모리로 구성되어 있다. 이 메모리는 저장된 데이터를 순환하며 필요한 시점에서 광소자 출력 단자로 데이터를 출력시키는 순차적 구조이다. IR (instruction register)에 저장된 명령어 형식은 그림 2와 같다.

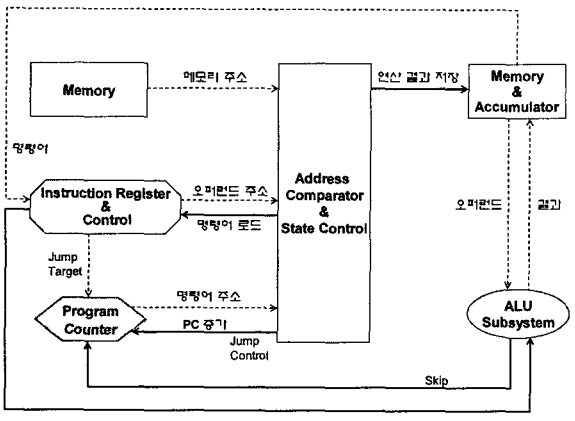


그림 1. SPOC의 구조
Fig. 1. SPOC architecture.

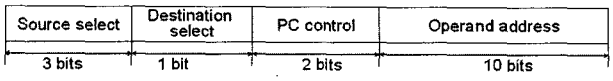


그림 2. 명령어 형식
Fig. 2. Instruction format.

그림 2에서 Source select 비트는 source 오퍼런드와 ALU 연산을 선택한다. Destination select 비트는 ALU 연산 결과를 저장할 Accumulator 또는 메모리를 선택한다. PC control 비트는 순차 실행, 조건 분기, 무조건 분기를 결정한다. Operand address는 10 비트로 구성되지만 SPOC의 메모리 크기가 64 워드이기 때문에 6비트만을 사용한다. SPOC의 특징은 표 1과 같다.

표 1. SPOC의 특징
Table 1. Characteristics of SPOC.

항목	특징
Clock 속도	50 MHz
워드 크기	16 비트
구조	Latchless 비트단위 직렬 워드 주소방식
메모리 크기	64 워드
컴포넌트	62
명령어	Load, Complement, Store, Add, Rotate Right, And, Or, Jump, Conditional Jump

2. SPOC의 전체 실행 과정

SPOC의 전체 실행 과정은 그림 1을 이용하여 설명한다. 그림 1에서 점선은 데이터의 흐름을 의미하고, 실선은 제어동작을 나타낸다. SPOC의 전체 실행 과정은 다음과 같다.

① 주소 비교기(address comparator)가 PC와 메모리의 내용을 비교한다.

② 과정 ①에서 비교한 결과가 일치하면 메모리 발견 신호(MEMF)를 발생시킨다. 그리고 State Control에서 명령어 발견 신호(INSTF)를 발생시킨다.

③ INSTF 신호에 따라 메모리에 저장된 명령어를 IR에 적재한다.

④ IR에 적재된 명령어의 오퍼런드 주소를 이용하여 메모리에서 오퍼런드를 인출하고 오퍼런드 발견 신호

(OPRF)를 발생시킨다.

⑤ OPRF 신호에 따라 IR에 적재된 명령어의 OPcode를 번역한다. Source select 비트에 따라 ALU의 연산 결과를 선택하고, destination select 비트에 따라 연산 결과를 저장할 장소를 선택한다. 그리고 PC 제어 비트는 Jump 또는 Skip을 선택한다.

이러한 SPOC의 전체 실행 과정은 명령어 인출과정(①-③)과 오퍼런드 인출과정(④, ⑤)으로 분류한다.

3. SPOC의 문제점

SPOC는 설계상으로 또는 비트 단위 직렬(bit-serial) 방식으로 동작하는 구조 때문에 다음과 같은 문제점을 가진다.

첫째, 메모리 사이클의 낭비가 많다. SPCO는 기본적으로 랜덤 액세스가 불가능한 DLM을 기억 장치로 채택하고 있기 때문에 원하는 데이터, 명령어를 얻기 위해서는 DLM의 출력 단자에서 원하는 결과가 출력될 때까지 대기해야 한다. 이것은 SPOC의 성능을 저하시키는 큰 요인이다. 하지만 비트 단위 직렬 방식의 SPOC로는 설계시 감수해야 하는 문제점이다. 또한, DLM의 구조를 bit-parallel로 변경하려면 광스위칭 소자의 수가 선형적으로 증가한다. 일반적으로 DLM에서 1 워드를 저장하려면 2개의 광스위칭 소자가 필요하다. 광스위칭 소자가 고가인 점을 고려한다면 제작비용이 많이 소요된다는 문제점이 있다.

둘째, 오퍼런드가 필요 없는 명령어의 경우에도 오퍼런드를 인출하기 위해 메모리에 접근해야 한다. 즉, SPOC는 2-state 컴퓨터로 오퍼런드가 필요 없는 명령어도 오퍼런드 인출 사이클을 실행하도록 설계되어 있다.

셋째, 산술논리장치에서 불필요한 연산 수행이 많다는 것이다. 즉, 2-state 구조인 SPOC는 오퍼런드 인출시에 Opcode를 번역하여 어떤 명령어인지를 알 수 없다. 따라서 원하는 연산을 선택하여 수행할 수 없기 때문에 산술논리장치에서 모든 연산을 수행한 후, Opcode를 해석하여 원하는 연산 결과만을 선택하는 문제점이 있다.

III. 회로 구현

본 논문에서는 SPOC의 문제점을 개선한 새로운 회로를 설계하고 구현한다. 그리고 프로그램 수행시간을 단축하기 위해 DIR 구조를 사용한다.

1. 상태 제어 로직

상태 제어 로직(State Control Logic)은 불필요한 오퍼런드 인출 상태를 제거하고, DIR 구조의 프로그램 실행 제어를 위한 회로이다. 상태 제어 로직은 컴퓨터의 실행 상태에 대한 정보를 가지며 상태 변경을 제어하는 기능을 하는 회로로써 그림 3과 같은 구조를 가진다.

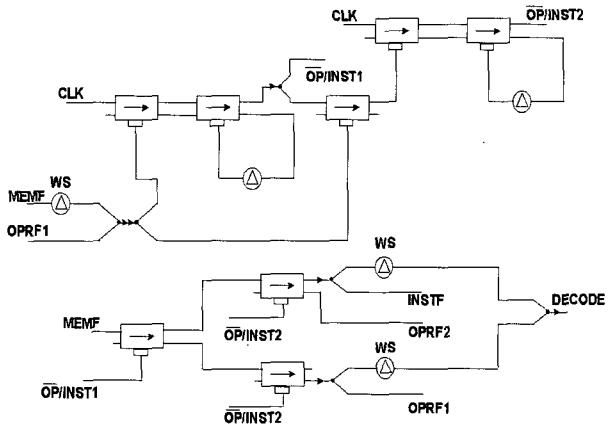


그림 3. 상태 제어 로직
Fig. 3. State control logic.

그림 3의 상단 부분에서는 $\overline{OP}$/INST1과 $\overline{OP}$/INST2를 사용하여 컴퓨터의 현재 상태를 알 수 있다. $\overline{OP}$/INST1과 $\overline{OP}$/INST2의 조합으로 표 2와 같은 제어신호를 얻을 수 있다.

표 2. 제어 상태 비트와 제어 신호의 관계
Table 2. Relations of control state bit and signal.

$\overline{OP}$/INST1	$\overline{OP}$/INST2	Control Signal
1	1	INSTF
0	1	OPRF1
1	0	Not Used
0	0	OPRF2

하단 부분에서는 MEMF 신호를 결정하는 기능을 제공한다. MEMF 신호가 INSTF이면 실제 명령어가 메모리에 존재한다는 의미이고, OPRF1 또는 OPRF2이면 DIR 구조에서 IR1 또는 IR2에 적재된 명령어의 오퍼런드를 찾았다는 의미이다. 이 신호들을 이용하여 DIR 구조의 전체 프로그램 실행 순서를 제어한다. 즉, INSTF 신호와 OPRF1이 발생하고 WS 클럭 후에 DECODE 신호를 발생시킨다. 만약 오퍼런드가 필요 없는 명령어의 경우, 명령어를 찾은 후에 명령어를 번역할 수 있는 상태에서 지체 없이 DECODE 신호를 발생시키면 오퍼런드 인출을 위한 메모리 접근 없이 바로 실행할 수 있다.

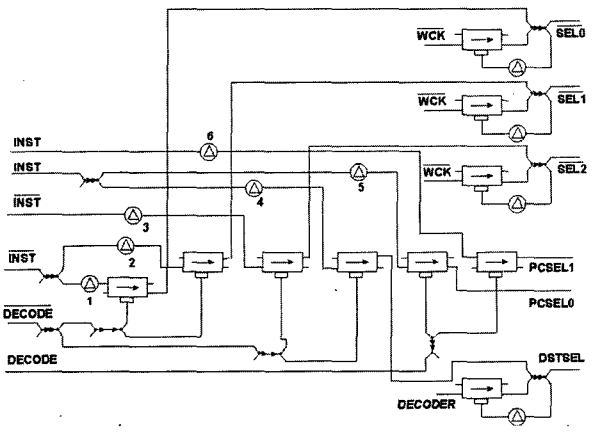


그림 4. 명령어 번역기
Fig. 4. Instruction decoder.

2. 명령어 번역기

명령어 번역기는 상태 제어 로직과 함께 불필요한 오퍼런드 인출 상태를 제거하는 기능을 제공한다. Opcode는 Source select, PC Control, Destination select로 구성되며, 이러한 정보를 번역하는 회로는 그림 4와 같다.

그림 4는 명령어가 IR에 적재된 후, 상태 제어 로직으로 부터 DECODE 신호를 받으면 바로 Opcode를 번역한다. 즉, Opcode를 먼저 번역함으로써 오퍼런드가 필요 없는 명령어의 경우, 오퍼런드 인출 과정을 생략한다.

3. 명령어 선택부

명령어 선택부는 산술논리장치에서 수행하는 불필요한 연산을 제거하기 위해 추가된 회로로써 그림 5와 같다.

그림 5는 Opcode의 Source select 비트를 번역한 결과 신호인 $\overline{SEL0} \sim \overline{SEL2}$를 전송 받는다. 각 $\overline{SEL}$신

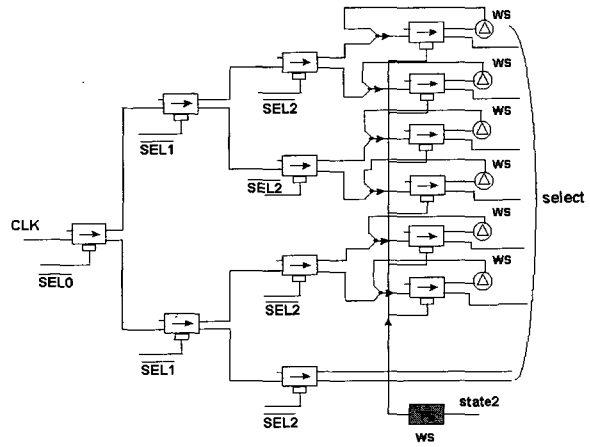


그림 5. 명령어 선택부
Fig. 5. Instruction selector.

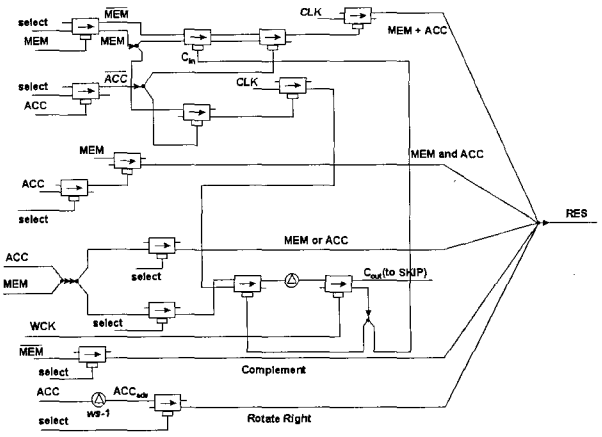


그림 6. 산술논리장치
Fig. 6. Arithmetic Logic Unit.

호들은 산술논리장치의 특정 연산을 선택하기 위한 select 신호를 생성한다. 명령어 선택부의 각 select 단자는 특정 연산을 선택하는 산술논리장치의 선택부에 연결된다.

4. 산술논리장치(ALU)

기존의 산술논리장치에 특정 연산을 선택하여 실행할 수 있는 선택회로를 추가한 산술논리장치는 그림 6과 같다. 명령어 선택부의 각 select 단자는 산술논리장치의 특정 연산과 연결되어 있기 때문에 수행하고자 하는 연산에 오퍼런드를 전송함으로써 연산을 선택하여 수행한다.

5.Dual Instruction Register(DIR)

이 회로는 컴퓨터의 수행 속도를 향상시키기 위해서 제안한 구조이다. 기존 회로에 IR을 하나 더 추가함으로써 한 번에 2개의 명령어를 인출하여 IR1과 IR2에 각각 적재한다. 즉, 현재 PC가 지시하는 명령어와 다음 명령어를 함께 인출하여 첫 번째 명령어는 IR1에 두 번째 명령어는 IR2에 적재한다.

DIR 구조를 적용한 타이밍 다이어그램은 그림 7과 같다. 1 워드가 16 비트이기 때문에 특정 명령어를 찾

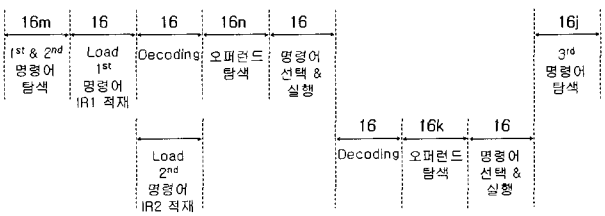


그림 7. 타이밍 다이어그램
Fig. 7. Timing diagram.

는데 위치에 따라 16m 클럭이 소요된다. 찾은 명령어를 IR1에 적재하는데 16 클럭이 소요된다. 바로 다음 명령어를 IR2에 적재하는 동안 IR1에 적재된 명령어를 번역한다. 이때 순차실행과 분기의 경우는 다르다. 순차실행의 경우, DECODE 신호에 따라 명령어를 번역하는 동안 다음 명령어를 IR2에 적재한다. 그리고 명령어 실행이 끝나면 IR2에 적재한 명령어를 실행한다. Jump 또는 Skip과 같은 분기의 경우, IR1에 적재한 명령어가 Jump 또는 Skip 이면 IR2에 적재한 명령어를 실행하지 않는다. 이러한 DIR의 장점은 두 번째 명령어를 찾는 시간과 IR에 적재하는 시간을 줄일 수 있다는 것이다.

6. 주소 비교기(Address comparator)

PC의 내용, 또는 명령어의 오퍼런드 주소와 메모리에 저장된 내용을 비교하는 기능을 제공하는 주소 비교기 회로는 그림 7과 같다. 만약 비교 내용이 일치하면 MEMF 신호를 발생시킨다. $\overline{OP}$/ $\overline{INST1}$ 가 1 이면 PC와 비교하고 0 이면 INST와 비교를 한다. 이 회로는 DIR 구조에서 분기를 제어한다. Jump나 Skip 명령어의 경우, IR2에 적재된 명령어의 오퍼런드 주소와 비교하는 것이 아니라 PC의 내용과 비교함으로써 IR2에 적재된 명령어를 실행하지 않는다.

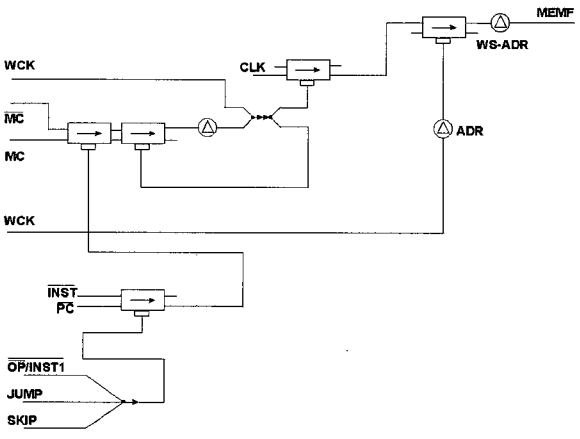


그림 8. 주소 비교기
Fig. 8. Address comparator.

7. 개선된 CPU 의 전체 수행 과정

개선된 CPU의 블록 다이어그램은 그림 9와 같다. 실선으로 표시된 것은 데이터의 흐름을 의미하고, 점선은 제어 신호를 의미한다.

먼저 PC가 지시하는 명령어를 메모리에서 찾는다. 명령어가 메모리에 존재하면 MEMF 신호를 발생시키

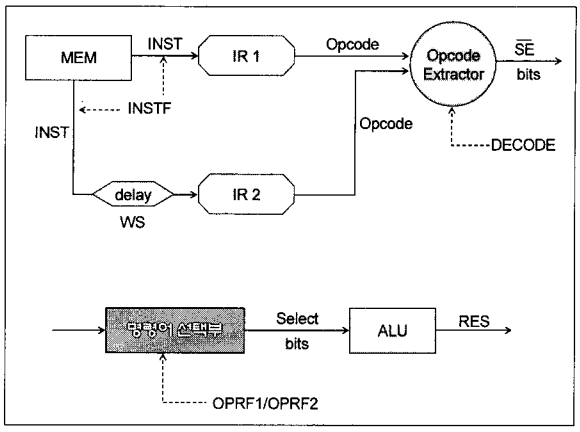


그림 9. CPU 블록 다이어그램
Fig. 9. CPU block diagram.

고, 상태 제어 로직에 의해 INSTF 신호로 번역된다. 이 신호에 따라 메모리에 저장된 명령어를 IR에 적재한다. R1에는 첫 번째 명령어가 적재되고, IR2에는 IR1에 명령어가 적재된 후에 WS 클럭이 발생하면 다음 명령어를 적재한다. WS 클럭이 발생한 후에 DECODE 신호에 따라 IR1에 적재된 명령어를 번역하게 한다. 그리고 IR1에 적재된 명령어의 오퍼런드 주소에서 오퍼런드를 찾으면 MEMF 신호를 발생시킨다. 상태 제어 로직은 MEMF 신호를 OPRF1 신호로 전환한다. OPRF1 신호에 따라 명령어 선택부는 select신호를 생성하여 산술논리장치에 전송함으로써 원하는 연산을 선택하여 수행한다.

IV. 회로 검증

본 논문에서는 참고문헌[13]에서 개발한 시뮬레이터를 사용하여 입력 데이터에 대한 출력 결과를 분석함으로써 구현한 회로의 정확한 동작 여부를 검증한다.

1.시뮬레이터의 구성과 동작 방식

시뮬레이터 모듈은 입력과정과 동작 검증과정, 그리고 출력과정으로 구성된다. 입력과정에서는 광소자의 연결 관계와 입력 데이터를 묘사한 텍스트 파일이 작성

표 3. 시뮬레이션 데이터표현 형식
Table 3. Data format of simulation.

항목	표현 형식
시뮬레이션 시간	period 100
입력단의 입력신호	input MEMF 0 1 0 1
출력단	output out1

된다. 시뮬레이션에 필요한 데이터들에 대한 표현 형식은 표 3과 같다.

표 3에서 시뮬레이션 시간은 시뮬레이션 하는 클럭 수를 의미하며 표현형식에서 'period 100'은 100 클럭 동안 시뮬레이션을 수행한다는 의미이다. 입력단의 입력은 'input' 키워드 뒤에 입력단 이름과 입력 데이터를 순차적으로 나열한다. 입력 데이터는 'period'에서 정한 클럭 수만큼 0과 1로 표현한다.

2. 회로 검증

본 논문에서 설계한 각 회로에 필요한 입력 데이터를 입력하면 정확한 출력 결과를 얻을 수 있는지 검증한다.

• 상태 제어 로직 검증: 상태 유지 회로와 상태 제어 회로로 나누어 검증한다. 상태 유지 회로는 입력 데이터로 MEMF 신호를 매 워드 클럭 마다 입력하여 정확히 상태 유지 비트를 출력하는지 검증한다. 상태 제어 회로는 입력 데이터로 상태 유지 비트를 입력한다. 상태 유지 비트에 따라 정확한 제어 신호 INSTF, OPRF1, OPRF2를 출력하는지 검증한다.

그림 10의 결과는 입력 신호 MEMF에 따라 $\overline{OP}$/INST1과 $\overline{OP}$/INST2의 신호가 각각 1010와 1100로 출력된 것이다.

• 명령어 번역기 검증: 입력 데이터로 임의의 명령어를 입력하여 해당 명령어의 각 Opcode를 정확히 출력하는지 검증한다.

그림 11은 입력 데이터 INST=1001101101001101을 입력한 경우의 출력 결과이다. 이 명령어의 Opcode=100110 이므로 Source select=100, Destination select=1, PC control=10 이다. 따라서 그림 11의 결과에서 두 번째 워드 기간동안에 Source select 비트 R_SEL0,

```
Filename: state.ols
Circuit analyzing...
period  input  output  switch  device  node
  64      2       2       5      11     20
Simulating...
period
64
Output display
 ‾/INST1 :
1111111111111111 0000000000000000
1111111111111111 0000000000000000
 ‾/INST2 :
1111111111111111 1111111111111111
0000000000000000 0000000000000000
```

그림 10. 상태 제어 로직의 시뮬레이션 결과
Fig. 10. Simulation result of state control logic.

```
Filename: Instruction.ols
Circuit analyzing...
period  input  output  switch  device  node
  64      6       6      10      40     56
Simulating...
period
64
Output display
R_SEL0 :
0000000000000000 0000000000000000
0000000000000000 0000000000000000
R_SEL1 :
0000000000000000 1111111111111111
0000000000000000 0000000000000000
R_SEL2 :
0000000000000000 1111111111111111
0000000000000000 0000000000000000
PC_SEL0 :
0000000000000000 1000000000000000
0000000000000000 0000000000000000
PC_SEL1 :
0000000000000000 0000000000000000
0000000000000000 0000000000000000
DST_SEL :
0000000000000000 1111111111111111
0000000000000000 0000000000000000
```

그림 11. 명령어 번역기의 시뮬레이션 결과
Fig. 11. Simulation result of instruction decoder.

```
Filename: select.ols
Circuit analyzing...
period  input  output  switch  device  node
 128      4       8      13      26     44
Simulating...
period
128
Output display
OUT1 :
1111111111111111 0000000000000000
0000000000000000 0000000000000000
0000000000000000 0000000000000000
0000000000000000 0000000000000000
OUT2 :
0000000000000000 1111111111111111
0000000000000000 0000000000000000
0000000000000000 0000000000000000
0000000000000000 0000000000000000
OUT3 :
0000000000000000 0000000000000000
1111111111111111 0000000000000000
0000000000000000 0000000000000000
0000000000000000 0000000000000000
OUT4 :
0000000000000000 0000000000000000
0000000000000000 1111111111111111
0000000000000000 0000000000000000
0000000000000000 0000000000000000
OUT5 :
0000000000000000 0000000000000000
0000000000000000 0000000000000000
1111111111111111 0000000000000000
0000000000000000 0000000000000000
OUT6 :
0000000000000000 0000000000000000
0000000000000000 0000000000000000
0000000000000000 1111111111111111
0000000000000000 0000000000000000
OUT7 :
0000000000000000 0000000000000000
0000000000000000 0000000000000000
0000000000000000 0000000000000000
1111111111111111 0000000000000000
OUT8 :
0000000000000000 0000000000000000
0000000000000000 0000000000000000
0000000000000000 0000000000000000
0000000000000000 1111111111111111
```

그림 12. 명령어 선택부의 시뮬레이션 결과
Fig. 12. Simulation result of Instruction selector.

R_SEL1, R_SEL2가 각각 011로, select 비트는 1, PC control 비트 PC_SEL0, PC_SEL1이 각각 10으로 정확하게 출력됨을 볼 수 있다.

• 명령어 선택부 검증: 입력 데이터로 Opcode의 Source select 부분을 번역한 $\overline{SEL0} \sim \overline{SEL2}$ 을 입력한다. 출력 데이터는 산술논리장치의 특정 연산을 선택하는 select 신호이다. 이때 원하는 출력 단자로 select 신호를 출력하는지 검증하였다.

그림 12는 $\overline{SEL0} \sim \overline{SEL2}$ 의 값을 000, 001,...111로 입력했을때 각 출력단자가 정확하게 선택되었음을 볼 수 있다. 즉, $\overline{SEL0} \sim \overline{SEL2}$ 의 비트가 000이면 출력단자 OUT1이 선택되고, 111이면 출력단자 OUT8이 선택되고 있음을 볼 수 있다.

• 산술논리장치 검증: 임의의 메모리 오퍼런드와, accumulator 오퍼런드, 명령어 선택부의 출력인 select 신호를 입력하여 정확한 연산 결과를 출력하는지 검증한다. 연산은 매 워드 클럭 마다 각 연산을 수행하고, 수행하고자 하는 연산만 수행되는지를 검증한다.

그림 13은 메모리 오퍼런드(MEM)=0010101101001010, accumulator 오퍼런드(ACC)=0001011001001101일 때, ADD, AND, OR, Complement연산을 수행한 결과이다. 첫 번째 워드 기간 동안에 ADD 연산 결과인 0011110010100000이 출력되고, 두 번째 워드 기간 동안에 AND 연산 결과인 0000001001001000이 출력된다. 그리고 세 번째 워드 기간 동안에 OR 연산 결과인 0011111101001111이 출력되고, 네 번째 워드 기간 동안에 complement 연산 결과인 1101010010110101이 출력됨을 볼 수 있다.

```
Filename: alu.ols
Circuit analyzing...
period  input  output  switch  device  node
  64     10      2       15      26     44
Simulating...
period
64
Output display
COUT :
0000000000000000 1000000000000000
0000000000000000 0000000000000000
RES :
0011110010100000 0000001001001000
0011111101001111 1101010010110101
```

그림 13. 산술논리장치의 시뮬레이션 결과
Fig. 13. Simulation result of ALU.

```
Filename: address.ols
Circuit analyzing...
period  input  output  switch  device  node
  64     8       1       5       12     22
Simulating...
period
64
Output display
MEMF :
0000000000000000 1000000000000000
1000000000000000 1000000000000000
```

그림 14. 주소 비교기의 시뮬레이션 결과
Fig. 14. Simulation result of address comparator.

• 주소 비교기 검증: 입력 데이터는 임의의 명령어, memory counter 값, PC 값이다. 출력 데이터는 memory counter 값과 명령어, 또는 memory counter 값과 PC 값이 일치했다는 신호 MEMF이다.

그림 14는 메모리 카운터(MC)=1011001011000000, PC= 100100110 1000000, 명령어(INST)=1011001011010100일 때, 그 결과를 나타낸다, 이때 메모리 카운터(MC)의 값과 명령어(INST)의 값이 앞에서 10개 비트가 일치한다. 따라서 시뮬레이션의 두 번째 워드 기간에서 MEMF 신호가 발생됨을 볼 수 있다.

V. 결 론

본 논문에서는 기존의 광컴퓨터인 SPOC의 문제점을 개선한 회로를 구현하고 검증하였다.

SPOC는 오퍼런드가 필요 없는 명령어도 오퍼런드를 찾기 위해 메모리에 접근하는 문제점이 있다. 본 논문에서는 오퍼런드를 찾기 전에 미리 명령어를 해독함으로써 오퍼런드가 필요 없는 명령어는 오퍼런드 인출과정 없이 실행하도록 회로를 개선하였다. 또한, SPOC는 원하는 연산을 선택하여 수행하지 못하므로 산술논리장치에서 불필요한 연산을 많이 수행한다. 본 논문에서는 명령어 선택부와 산술논리장치에 선택회로를 추가하여 특정 연산을 선택하여 수행함으로써 산술논리장치의 불필요한 연산을 제거하였다. 그리고 기존 회로에 1개의 IR을 더 추가한 DIR 구조를 사용함으로써 전체적인 프로그램 수행 시간을 단축하였다. 특히, 순차실행의 경우는 분기의 경우보다 더 많은 수행시간을 단축할 수 있었다.

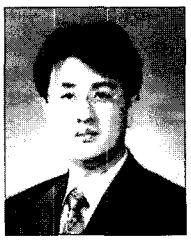
향후 연구에서는 본 논문에서 개선한 회로와 기존의 SPOC의 회로 소자 수, 동작 속도 등의 성능을 비교 분석할 수 있는 시뮬레이터를 구현하는 것이다.

참 고 문 헌

[1] W. H. Cathey, K. Wagner, and W. J. Miceli, "Digital Computing with Optics", Proc. IEEE, Vol. 77 No. 10, pp. 1558 - 1572, Oct. 1989,
[2] J. L. Johnson, "Architectural Relationships Involving

Symbolic Substitution," Appl. Opt., Vol. 27, No. 3, pp. 529-533, Feb. 1988.
[3] A. F. Benner, J. Bowman, T. Erkkila, R. J. Feuerstein, V. P. Heuring, H. F. Jordan, J. Sauer, and T. Soukup, "Digital Optical Counter Using Directional Coupler Switches," Appl. Opt., Vol. 30, No. 29, pp. 4179-4189, Oct. 1991.
[4] A. D. McAulay, J. Wang, and X. Xu, "Optical Adder that Uses Spatial Light Rebroadcasters," Appl. Opt., Vol. 31, No. 26, pp. 5584-5591, Sep. 1992.
[5] W. P. Heuring, H. F. Jordan, and J. P. Pratt, "Bit-serial Architecture for Optical Computing," Appl. Opt., Vol. 31, No. 17, pp. 3213-3224, Jun. 1992.
[6] J. P. Pratt, and V. P. Heuring, "Designing digital optical computing systems: power distribution and cross talk", Appl. Opt., Vol. 31 No. 23, pp. 4657 - 4661, Aug. 1992,
[7] C.E. Love, H.F. Jordan, "SPOC - A Stored Program Optical Computer," IEEE Potentials Vol. 13, No. 4, Oct./Nov. 1994.
[8] T. Main, R. J. Feuerstein, H. F. Jordan, V. P. Heuring, J. Feehrer, and C. E. Love, "Implementation of a general-purpose stored-program digital optical computer", Appl. Opt., Vol. 33 No. 8, pp. 1619 - 1628, Mar. 1994.
[9] H. F. Jordan, V. P. Heuring, and R. Feuerstein, "Optoelectronic Time-of-Flight Design and Demonstration of an All-optical, Stored Program, Digital Computer", Proc. of IEEE Special Issues on Optical Computing, Vol. 82 No. 11, Nov. 1994.
[10] Guilfoyle, "Digital Optical Computing Architectures for Compute Intensive Applications ," Proc. 1994 Int'l Conf. Optical Computing, 1994.
[11] Guilfoyle et al., "Smart Optoeletronic Architecture for Data/Knowledge-Based Applications," Proc. Third Int'l Conf. on Electronics Circuits and Systems, Vol. 2, pp. 566-569, 1996.
[12] Guilfoyle, John M. Hessenbruch, Richard V. Stone, "Free-Space Interconnects for High-Performance Optoelectronic Switching," Computer, Vol. 31, No. 2, pp. 69-75, Feb. 1998.
[13] 박종현, 이원주, 전창호, "광스위칭 소자에 기반한 산술논리연산회로의 설계", 컴퓨터산업교육학회논문지, 제3권, 제2호, pp. 148 - 158, Feb. 2002.

저 자 소 개



이 원 주(정회원)
1989년 한양대학교 컴퓨터공학과 학사.
1991년 한양대학교 컴퓨터공학과 석사.
2004년 한양대학교 컴퓨터공학과 박사.
2006년 현재 두원공과대학 인터넷프로그래밍과 부교수.
<주관심분야 : 광컴퓨터, Grid 컴퓨팅, 병렬처리 시스템, 모바일 컴퓨팅 >