# 실장제어 16 비트 FPGA 마이크로프로세서

차영호* · 조경연* · 최혁환*

## A 16 bit FPGA Microprocessor for Embedded Applications

Young-Ho Cha* · Gyeong-Yeon Cho* · Hyek Hwan Choi*


### 요 약

SoC(System on Chip) 기술은 높은 융통성을 제공하므로 실장제어 분야에서 널리 활용되고 있다. 실장제어 시스템은 소프트웨어와 하드웨어를 동시에 개발하여야 하므로 많은 시간과 비용이 소요된다. 이러한 설계시간과 비용을 줄이기 위해 고급언어 컴파일러에 적합한 명령어 세트를 가지는 마이크로프로세서가 요구된다. 또한 FPGA(Field Programmable Gate Array)에 의한 설계검증이 가능해야 한다.

본 논문에서는 소형 실장제어 시스템에 적합한 EISC(Extendable Instruction Set Computer) 구조에 기반한 16 비트 FPGA 마이크로프로세서인 EISC16을 제안한다. 제안한 EISC16은 짧은 길이의 오프셋과 작은 즉치 값을 가진 16 비트 고정 길이 명령어 세트를 가진다. 그리고 16 비트 오프셋과 즉치 값은 확장 레지스터와 확장 플래그를 사용하여 확장한다.

또한, IBM-PC와 SUN 워크스테이션 상에서 C/C++ 컴파일러 및 응용 소프트웨어를 설계하였다. 기존 16 비트 마이크로프로세서들의 C/C++ 컴파일러들 만들고 표준 라이브러리의 목적 코드를 생성하여 크기를 비교한 결과 제안한 EISC16의 코드 밀도가 높음을 확인하였다.

제안한 EISC16은 Xilinx의 Vertex XCV300 FPGA에서 RTL 레벨 VHDL로 설계하여 약 6,000 게이트로 합성되었다. EISC16은 ROM, RAM, LED/LCD 판넬, 주기 타이머, 입력 키 패드, 그리고 RS-232C 제어기로 구성한 테스트 보드에서 동작을 검증하였다. EISC16은 7MHz에서 정상적으로 동작하였다.



### ABSTRACT

SoC(System on Chip) technology is widely used in the field of embedded systems by providing high flexibility for a specific application domain. An important aspect of development any new embedded system is verification which usually requires lengthy software and hardware co-design. To reduce development cost of design effort, the instruction set of microprocessor must be suitable for a high level language compiler. And FPGA prototype system could be derived and tested for design verification.

In this paper, we propose a 16 bit FPGA microprocessor, which is tentatively-named EISC16, based on an EISC(Extendable Instruction Set Computer) architecture for embedded applications. The proposed EISC16 has a 16 bit fixed length instruction set which has the short length offset and small immediate operand. A 16 bit offset and immediate operand could be extended using by an extension register and an extension flag.


* 부경대학교 전자컴퓨터 정보통신공학부

접수일자: 2001. 12. 14

We developed a cross C/C++ compiler and development software of the EISC16 by porting GNU on an IBM-PC and SUN workstation and compared the object code size created after compiling a C/C++ standard library, concluding that EISC16 exhibits a higher code density than existing 16 microprocessors.

The proposed EISC16 requires approximately 6,000 gates when designed and synthesized with RTL level VHDL at Xilinix's Virtex XCV300 FPGA. And we design a test board which consists of EISC16 ROM, RAM, LED/LCD panel, periodic timer, input key pad and RS-232C controller. It works normally at 7MHz clock.

Keyword

Application-specific processor, embedded system, FPGA processor, EISC processor, SoC(System on Chip)

## Ⅰ. 서 론

마이크로프로세스와 메모리 그리고 각종 주변회로 등 전체 시스템을 하나의 IC에 집적하는 SoC(System on Chip) 설계기술이 실장제어 시스템 개발에 널리 활용되고 있다. 소형 실장제어 시스템에서는 하드웨어 가격과 프로그램 메모리 크기의 제한 때문에 어셈블러를 주로 사용하여 왔다. 그러나 어셈블러에 의한 소프트웨어 개발은 점점 더 복잡해지는 응용프로그램 개발로 인해 어려움이 증가하고 있다. 또한, 실수를 유발할 가능성 또한 증대되고 있기 때문에 많은 개발 시간이 필요할 뿐만 아니라, 유지보수에도 많은 비용이 소요된다. 이러한 문제점으로 C/C++ 등 고급언어 컴파일러에 적합한 실장제어용 마이크로 프로세서 개발이 요구되고 있다.[1]-[4]

SoC 설계기술은 논리회로 시뮬레이션에 의존하여 설계를 검증한다. 그러나 시뮬레이션으로 인식되지 않는 설계상의 오류가 항상 상존하고 있다. 더욱이 하드웨어와 소프트웨어를 동시에 설계해야 하는 설계에서는 시뮬레이터의 성능이 제한적이어서 올바른 설계검증이 어렵다. 따라서 시뮬레이터를 통하여 검증된 설계는 FPGA(Field Programmable Gate Array)를 통한 시제품 제작 후, 최종적인 설계 검증을 수행하여 설계 오류를 확인하여 개발비용을 줄여야 한다. 따라서 소형 실장제어 분야에 SoC 기술을 효과적으로 활용하려면 FPGA로 구현이 가능하도록 하드웨어가 간단하고, 고급 언어 컴파일러에 효율적인 마이크로 프로세서에 대한 연구가 필요하다.[5]

1970년대 이래로 소형 실장제어 시스템 분야에서 널리 사용되고 있는 Zilog사의 Z80[6]이나 Intel사의 I8051[7]등 8 비트 마이크로프로세스는 고급언어 컴파일러에 비효율적인 구조를 가지고 있으며, Hitachi사의 H-8[8]과 Panasonic사의 MN10200[9]등의 기존 16 비트 마이크로프로세서는 하드웨어가 복잡하여 ASIC (Application Specific IC)에 집적하기에는 어려움이 따른다. 또한 1980년대에 개발된 RISC(Reduced Instruction Set Computer) 구조는 명령어 길이가 32 비트이므로 8 비트나 16 비트 마이크로프로세스에 적합하지 않다. C 컴파일러를 지원하는 8 비트 RISC[10]도 연구되었지만, 22 비트 고정길이 명령어를 사용하므로 코드 밀도가 낮기때문에 많은 메모리를 요구하는 문제점이 있다. 이러한 문제점을 해결하기 위하여 본 논문에서는 EISC(Extendable Instruction Set Comput- er)[11] 구조를 가지는 16 비트 FPGA 마이크로프로세스인 가칭 EISC16을 제안한다.

EISC16은 16 비트 고정길이 명령어를 가지며, 오퍼랜드는 확장 레지스터와 확장 플래그를 사용하여 16 비트 길이로 확장할 수 있다. 또한 8개의 범용레지스터와 4개의 특수목적 레지스터를 가지며, 3 스테이지 파이프라인으로 하드웨어를 구성하였다. 소프트웨어로는 GNU를 이식하여 C/C ++ 컴파일러 및 디버거를 설계하였다.

EISC16 C/C++ 컴파일러의 성능 검증을 위하여 16 비트 마이크로프로세스인 Hitachi사의 H8-300과 Panasonic사의 MN10200의 C/C++ 컴파일러를 동일한 환경에서 작성하고, 벤치 마크 프로그램으로는 Cygnus사에서 개발한 실장제어용 C 표준 라이브러리인 newlib[12]와

SGI사에서 개발한 C++ 표준 라이브러리인 libstdc++ [13]를 컴파일하여 생성된 목적 코드의 크기를 비교하였다.

## II. EISC16의 구조

### 2-1. 레지스터 구조

EISC16은 16 비트 길이의 범용 레지스터 8개와 특수 레지스터 4개를 가진다. 그림 1에 레지스터 구성을 보인다.

R0에서 R7은 범용 레지스터로 연산을 수행할 때 오퍼랜드로 기능하거나 연산 결과를 저장한다. 또한 로드/스토어 명령어에서 인덱스 기능을 가진다. R7은 스택 포인터 레지스터(SP: Stack Pointer)로도 동작한다. 스택은 외부 메모리를 사용하며 주소가 감소하는 방향으로 성장한다. 스택 포인터 레지스터는 항상 최상위 스택(TOS: Top-Of-Stack)을 지시한다.

PC(Program Counter)는 프로그램 카운터로 현재 수행하는 명령어의 주소를 지정한다. EISC16은 16 비트 고정 길이 명령어 구조이므로 프로그램 주소는 항상 짝수 번지를 가진다. 따라서 PC의 최하위 비트는 항상 '0' 이다.

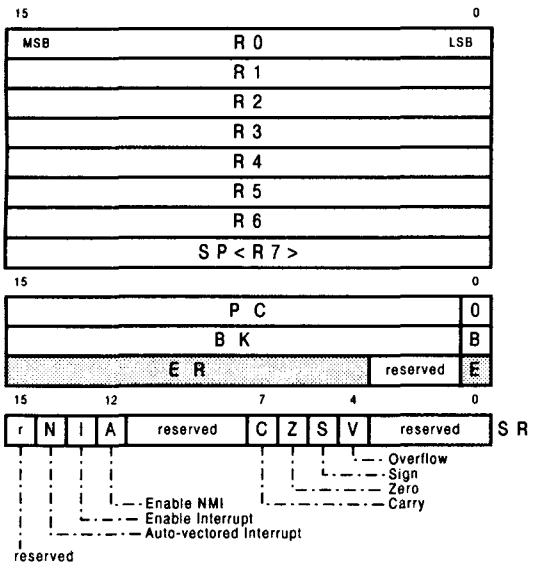



그림 1. 레지스터의 구성
Fig. 1 Programmer's Model

BK(BreaK point)는 프로그램 디버깅을 위하여 실행을 정지할 프로그램 주소를 가지는 레지스터이다. BK의 최하위 비트는 디버깅 실행여부를 나타내는 플래그이다.

ER(Extension Register)은 확장 레지스터로 상위 12 비트는 오퍼랜드를 확장하기 위한 확장 데이터 필드이고, 최하위 비트는 확장 여부를 나타내는 확장 플래그(E Flag)이다.

SR(Status Register)은 프로세스의 상태를 나타내는 레지스터로써, 연산처리 결과 상태를 기억하는 캐리, 제로, 사인 그리고 오버플로의 4 비트 플래그와 3 비트의 인터럽트 제어 플래그로 구성한다.

### 2-2. 명령어 세트

명령어의 디코딩 속도를 증가시키고 하드웨어를 단순하게 구성하기 위하여, 16 비트 고정길이 명령어 형식으로 구성한다. 또한 로드/스토어 명령어 형식으로 모든 연산은 레지스터사이에서만 수행하고, 메모리 입출력은 오직 로드/스토어 명령어로만 수행한다.

명령어 세트와 확장 플래그의 변경 상태를 그림 2에 보인다.

| Instruction | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | E flag |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Load/Store | 0 | OPC | | | offset | | | | | | index | | src/dst | | | | 0 |
| LERI | 1 | 0 | 0 | 0 | immd | | | | | | | | | | | | 1 |
| reg ALU | 1 | 0 | 0 | 1 | OPC | | | src2 | | | src1 | | dst | | | | - |
| immd ALU | 1 | 0 | 1 | i | OPC | | | immd | | | src | | dst | | | | 0 |
| Shift | 1 | 1 | 0 | 0 | 0 | 1 | OPC | | | | count | | | reg | | | - |
| Jump/Call | 1 | 1 | 0 | 1 | OPC | | | | | | offset | | | | | | 0 |
| Misc Inst. | 1 | 1 | 1 | 0 | 0 | 0 | 0 | OPC | | | | | | immd/reg | | | - |
| Move | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | src | | | dst | | | - |
| Load immd | 1 | 1 | 1 | 0 | 1 | immd | | | | | | | | dst | | | 0 |

그림 2. EISC16의 명령어
Fig. 2 EISC16 instruction set

LERI(Load Extension Register Immediate) 명령어는 12 비트 즉치 값을 확장 레지스터의 확장 데이터 필드에 저장하며, 확장 플래그를 '1'로 세트한다. 그리고 뒤따르는 명령어에서 확장 레지스터의 값을 사용하여 오퍼랜드를 확장하면, 확장 플래그는 '0'으로 리세트된다. LERI 명령어는 어셈블러가 오퍼랜드를 확장하는 명령어 전단에 자동적으로 생성한다.

로드/스토어(Load/store) 명령어는 7 비트 변위를

가지는 인덱스 주소모드로 정의한다. 로드/스토어 메모리의 유효주소는 확장 플래그 상태에 따라서 그림 3(a)과 같이 결정한다. 확장 플래그가 '0'이면 명령어의 7 비트 변위를 16 비트로 제로 확장하여서 인덱스 레지스터와 더한다. 또한 확장 플래그가 '1'이면 명령어의 7 비트 변위 중 하위 4 비트와 확장 레지스터의 확장 데이터 필드 12 비트를 왼쪽으로 4 비트 이동시킨 것을 연결하여 16 비트 변위를 구하고, 인덱스 레지스터와 더한다.

산술/논리(ALU) 명령어는 3 오퍼랜드를 가지는 Add, Add with Carry, Subtract, Subtract with carry, AND, OR 그리고 XOR 7 가지 명령어와 2 오퍼랜드를 가지는 Compare 명령어로 구성된다. 오퍼랜드 구성에 따라서 레지스터와 레지스터 연산과 레지스터와 즉치 값 연산의 2 가지 형식이 있다. 즉치 값 연산 산술/논리 명령어에서 실제적인 즉치 값은 확장 플래그 상태에 따라서 그림 3(b)와 같게 된다. 즉, 그림 2에서 산술/논리 명령어는 4 비트의 즉치 데이터 필드를 가지는데, 확장 플래그가 '0'이면 명령어의 4 비트 즉치 데이터 필드를 16 비트로 부호 확장하여 16 비트 즉치 값을 생성하고, 확장 플래그가 '1'이면 확장 레지스터의 확장 데이터 필드 12 비트를 왼쪽으로 4 비트 이동시키고 명령어의 4 비트 즉치 값을 연결하여 16 비트 즉치 값을 생성한다.

프로그램 제어 명령어인 Jump/Call 명령은 14개의 조건분기 명령어와 무조건 분기 명령어 그리고 서부루틴 콜 명령어로 구성하며, 8 비트 변위의 프로그램 카운터 상대 어드레싱 모드를 가진다. 분기 목적 주소는 확장 플래그 상태에 따라서 그림 3(c)와 같이 결정한다. 확장 플래그가 '0'이면 명령어의 8 비트 변위를 16 비트로 사인 확장하여서 프로그램 카운터와 더한다. 그리고 확장 플래그가 '1'이면 명령어의 8 비트 변위 중 하위 4 비트와 확장 레지스터의 확장 데이터 필드 12 비트를 왼쪽으로 4 비트 이동시킨 것을 연결하여 16 비트 변위를 형성하고, 프로그램 카운터와 더한다.

즉치 로드(Load immd) 명령어는 즉치 값을 레지스터에 로드하는 명령어로 8 비트의 즉치 필드를 가진다. 실제 로드되는 즉치 값은 확장 플래그 상태에 따라서 명령어의 8 비트 즉치 값을 사인 확장한 값이 되거나, 명령어의 8 비트 즉치 필드의 하위 4 비트 즉치 값과 확장 레지스터의 확장 데이터 필드의 12 비트 값을 연결한 값이 된다.

이들 확장 플래그 상태에 따라서 오퍼랜드를 확장하는 명령어는 항상 확장 플래그를 '0'으로 리세트시킨다.

### 2-3. 예외 처리

예외 처리는 리세트, 마스크불가 인터럽트, 인터럽트 그리고 소프트웨어 인터럽트를 설정하였다. 표 1에 예외처리 처리 순서도를 보인다.

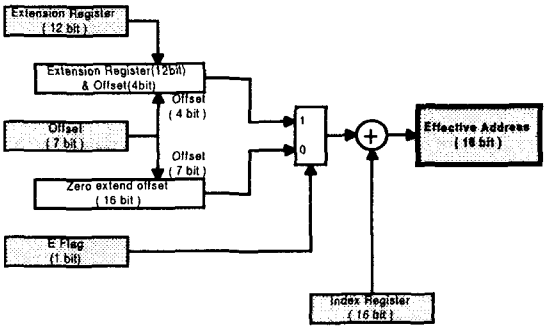



그림 3. (a) Load/Store의 유효주소
Fig. 3 (a) Effective address of Load/Store

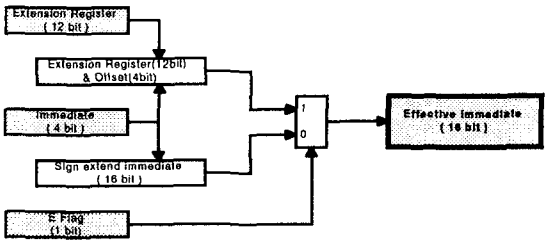



그림 3. (b) 산술/논리연산의 즉치 값
Fig. 3 (b) Effective immediate of Arithmetic/Logic immediate operation

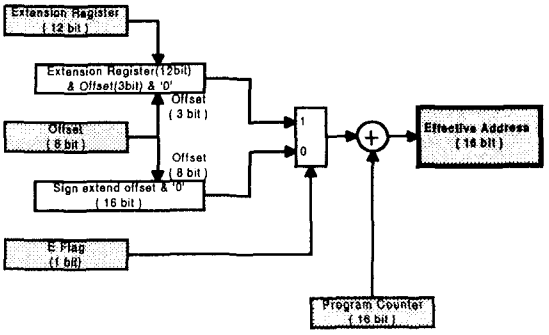



그림 3. (c) 조건분기와 Call의 유효주소
Fig. 3 (c) Effective address of Conditional branch and Call

리세트가 발생하면 상태 레지스터와 확장 플래그 및 디버깅 플래그를 리세트시키고, 리세트 벡터 값에 따른 분기를 수행한다. 마스크불가 인터럽트는 수행중인 프로그램 카운터 값과 상태 레지스터와 확장 레지스터를 스택에 대피시키고 상태 레지스터의 인터럽트 플래그를 리세트시켜서 중복 인터럽트 발생을 금지시킨 후, 마스크 불가 인터럽트 벡터 값에 따른 분기를 수행한다.

인터럽트는 자동 벡터와 외부 벡터의 두 가지 방식으로 인터럽트 서비스 루틴으로 분기한다. 자동 벡터는 항상 일정한 인터럽트 벡터 값을 가지며, 외부 벡터는 8 비트 벡터 값을 인터럽트를 요구한 외부 장치 또는 외부 인터럽트 제어기가 공급하며, 공급된 벡터 값에 따른 분기를 수행한다.

예외 처리 우선 순위는 리세트, 마스크불가 인터럽트, 소프트웨어 인터럽트의 순서로 설정한다.

표 1. 예외처리
Table. 1 Exception Processing

| |
|---|
| [RESET]<br>[1] Clear %SR, %BK.b0, %ER.b0<br>[2] Get %PC from 0x0000 |
| [NMI]<br>[1] Push %PC, %SR, %ER<br>[2] Clear %SR.b14, %SR.b13, %ER.b0<br>[3] Get %PC from 0x0002 |
| [INT]<br>[1] Push %PC, %SR, %ER<br>[2] Clear %SR.b13, %ER.b0<br>[3] If ( %SR.b12 is 0) Get %PC from 0x0004<br>[4] Get interrupt vector V<br>at interrupt-acknowledge cycle<br>[5] Get %PC from 000V X 2 |
| [SWI]<br>[1] Push %PC, %SR, %ER<br>[2] Clear %SR.b13, %ER.b0<br>[3] Get %PC from 000N X 2<br>where N is SWI vector |
| Interrupt Vector : 0x0000 - 0x00FF<br>0x0000-0x0001 v00 = Reset vector<br>0x0002-0x0003 v01 = NMI auto-vector<br>0x0004-0x0005 v02 = INT auto-vector<br>0x0006-0x001F v03 - v0F = SWI<br>0x0020-0x01FF v10 - vFF = Interrupt vector |

## 4. 데이터 패스 구조

EISC16의 데이터 패스 블럭도를 그림 4에 보인다.

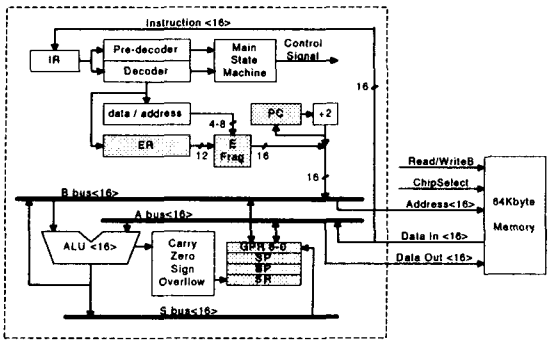



그림 4. EISC16의 데이터 패스 블럭도
Fig. 4 Data path of EISC16

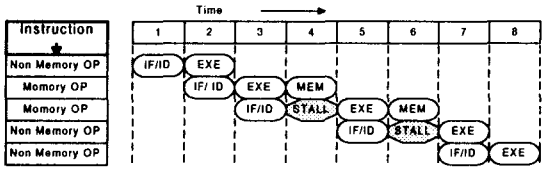



그림 5. EISC16의 파이프라인 동작
Fig. 5 EISC16 pipeline operation

명령어는 메모리로부터 인출되어서 명령어 레지스터(IR)에 저장된다. 명령어는 프리 디코더와 디코드를 거쳐서 해석되어 주상태기계(main state machine)에 입력된다.

파이프라인은 메모리 입출력을 수반하지 않는 명령어는 2 스테이지, 메모리 입출력을 수반하는 명령어는 3 스테이지로 구성한다. 첫 번째 스테이지는 명령어를 인출하고 프리 디코더를 수행하며, 두 번째 스테이지는 명령어를 해석하고, 오퍼랜드를 인출하고, 연산을 수행한다. 또한 메모리 입출력 명령어에서는 메모리 번지를 계산한다. 그리고 연산명령에서는 연산결과를 레지스터에 저장한다. 세 번째 스테이지에서는 메모리 입출력을 수행한다.

단일 메모리 구조를 가지므로 메모리 입출력과 명령어 인출을 동시에 수행할 수 없다. 따라서 메모리 입출력 스테이지에서는 명령어 인출이 대기 상태로 된다. 그림 5에 메모리 입출력 스테이지에서 다음 명령어들이 대기 상태로 되는 파이프라인 동작을 보인다.

이러한 파이프라인 동작은 주상태기계에 의하여 제어된다. 주상태기계는 23가지 상태를 가지는 동기순서회로이며 그 상태를 표 2에 보인다.

표 2. EISC16의 주상태기계
Table. 2 EISC16 Main State Machine

| State | Description |
|---|---|
| State-0 | Reset State |
| State-1 | Exception Vector Load State |
| State-2 | Halt State |
| State-3 | Hold State |
| State-4 | Instruction fetch |
| State-5 | ALU execution and fetch next instruction |
| State-6 | Instruction fetch and calculate load/store memory address |
| State-7 | Memory read/write and calculate load/store memory address |
| State-8 | ALU execution and Memory write |
| State-9 | ALU execution |
| State-10 | Memory read/write |
| State-11 | Prepare for Push/Pop and Memory read/write |
| State-12 | Prepare for Push/Pop |
| State-13 | Push/Pop operation |
| State-14 | CALL instruction execution |
| State-15 | CALLR instruction execution |
| State-16 | Special machine cycle state |
| State-17 | Interrupt Acknowledge state |
| State-18 | PUSH %PC execution |
| State-19 | PUSH %SR execution |
| State-20 | PUSH %ER execution |
| State-21 | Clear %SR state |
| State-22 | Load %PC state |

## Ⅲ. 소프트웨어

소프트웨어는 IBM-PC와 SUN 워크스테이션에서 FSF(Free Software Foundation)의 GCC를 이식하였으며 자세한 것을 표 3에 보인다. C/C++ 크로스 컴파일러 및 표준 라이브러리와 어셈블러 및 소스 라인 디버거를 작성하였으며, 개발 환경은 V-IDE를 이식하였다. 실시간 운영체제는 uC/OS를 이식하였다.

코드 밀도 비교를 위하여 16 비트 마이크로프로세스인 Hitachi사의 H8-300과 Panasonic사의 MN10200의 크로스 C/C++ 컴파일러를 동일한 환경에서 작성하였다. 벤치 마크 프로그램으로는 Cygnus사에서 개발한 실장제어용 C 표준 라이브러리인 newlib와 SGI사에서 개발한 C++ 표준 라이브러리인 libstdc++를 컴파일하여 생성된 목적 코드의 크기를 비교하여 결과를 표 4에 보인다. Newlib은 ANSI C 표준 라이브러리와 부동소수점 라이브러리로 구분되며 이들을 분리하여 나타내었다. 이들 라이브러리는 C/C++ 언어에서 많이 사용하는 표준화된 프로그램들로 구성되어 있으며, 전문가들이 작성하였고, 컴파일된 목적 코드가 300K 바이트에 이르는 것으로 충분히 크기 때문에 벤치 마크 프로그램으로 적당하다.

표 3. EISC16의 크로스 소프트웨어
Table. 3 EISC16 cross software

| Host platform | Window-95, Window-NT, Linux, SUN |
|---|---|
| Object file format | ELF |
| Cross assembler | Binutils-2.9.1 |
| Cross loader | Binutils-2.9.1 |
| Binary utilities | Binutils-2.9.1 |
| Cross C compiler | GCC-2.95.2 |
| Cross C++ compiler | LIBSTDC++-2.10.0 |
| C library | Cygnus NEWLIB-1.8.2 |
| C++ library | GCC-2.95.2 |
| Cross debugger | GDB-5.0 |
| Simulator | GDB-5.0 |
| Real Time OS | uC/OS-2.0 |
| Work bench | V-IDE |

표 4. 목적코드의 크기 비교표
Table. 4 Object code size comparison table

| Processor | Newlib-1.8.2 (libc) | Newlib-1.8.2 (libm) | Libstdc++ -2.10.0 | Total |
|---|---|---|---|---|
| EICS16 | 60,899 (100) | 66,630 (100) | 135,649 (100) | 263,178 (100) |
| H8-300 | 68,940 (126) | 93,088 (140) | 187,300 (138) | 357,244 (136) |
| MN10200 | 70,243 (115) | 64,100 (96) | 163,587 (121) | 297,930 (113) |

unit is byte

표 4에서 목적 코드의 크기는 바이트 단위이며 괄호 속의 값은 EISC16 코드의 크기를 100으로 하였을 때 상대적인 코드 크기이다.

벤치 마크 비교 결과 EISC16의 코드 밀도는 H8-300에 비하여 27%, MN10200에 비하여 12% 정도 높게 나타났다. 따라서 EISC16이 C/C++ 등 고급언어 컴파일러에 적합한 구조임을 알 수 있다.

## Ⅳ. 하드웨어 구현 및 검증

하드웨어는 RTL 레벨의 VHDL로 설계하여 Xilinx사의 Virtex XCV300에서 구현하였으며, 약 6,000 게이트가 소요되었다. 표 5에 블록별 게이트 소요를 보

인다. 레지스터부가 약 3,300 게이트로 가장 많은 게이트를 차지하고, 주상태기계를 포함하는 제어부에 약 1,700 게이트가 소요되었다.

구현된 하드웨어의 동작 검증을 위하여 그림 6과 같은 테스트 보드를 제작하였다. 테스트 보드는 각각 32K 바이트 ROM, RAM과 LCD/LED 디스플레이, 키패드, 타이머 및 RS-232C 통신 기능을 가진다. 타이머에서 주기적으로 인터럽트를 발생하고, 인터럽트 서비스 루틴에서 LED에 그 상태를 표시하여 예외 처리 기능을 검증하였다. 또한 RS-232C를 통하여 PC와 연결하여 프로그램을 다운 로드하여서 수행시키고 그 결과를 LCD에 나타내어서 프로그램이 정상적으로 동작하는 것을 검증하였다. 제작된 EISC16은 7MHz 클럭에서 정상 동작하였다.

표 5. 게이트 수
Table. 5 Gate counter

| Block | Gate Counter |
|---|---|
| Register File | 3305 |
| Main Control Block | 1683 |
| Data Input Block | 477 |
| ALU Block | 528 |
| Total | 6084 |

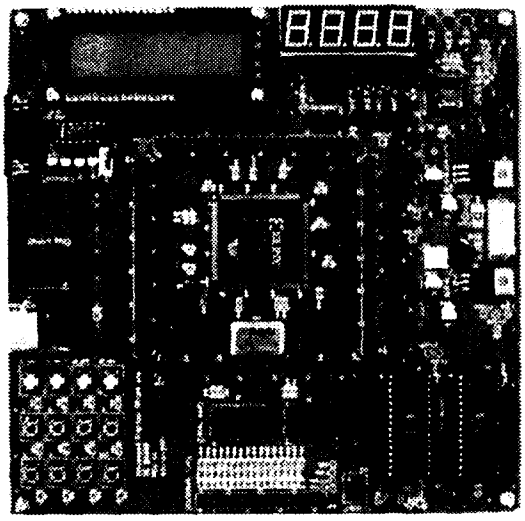


그림 6. EISC16의 테스트보드
Fig. 6 EISC16 test board

## V. 결 론

본 논문에서는 EISC(Extendable Instruction Set Computer) 구조를 가지는 16 비트 FPGA 마이크로프로세스인 가칭 EISC16을 제안하였다.

EISC16은 16 비트 고정길이 명령어를 가지며, 오퍼랜드는 확장 레지스터와 확장 플래그를 사용하여 확장하여 모든 길이의 오퍼랜드를 표현할 수 있다. 또한 8개의 범용레지스터와 4개의 특수목적 레지스터를 사용하고, 3 스테이지 파이프라인으로 하드웨어를 구성하였으며, C/C++ 컴파일러 및 디버거는 GNU를 이식하였다.

또한 EISC16 C/C++ 컴파일러의 성능 검증을 위하여 기존 16 비트 마이크로프로세스의 컴파일러를 동일한 환경에서 작성하고, C/C++ 라이브러리를 컴파일하여 생성된 목적 코드의 크기를 비교하였다. 벤치 마크 비교 결과 EISC16의 코드 밀도는 기존 16 비트 마이크로프로세서에 비하여 12-27% 높게 나타났다.

또한 제안한 EISC16은 RTL 레벨의 VHDL로 설계하여 Xilinx 사의 Virtex XCV300에서 합성하여 약 6,000 게이트로 구현되었고, EISC16을 포함하는 테스트 보드를 제작하여 7MHz 주파수에서 정상적으로 동작하는 것을 확인하였다.

본 논문에서 제안한 16 비트 FPGA 마이크로프로세서인 EISC16은 FPGA에 실장이 가능한 수준으로 하드웨어가 간단하며, C/C++ 등 고급 언어 컴파일러에 효율적이므로 소형 실장제어 시스템에 폭 넓은 활용이 기대된다.

감사의 글

본 연구는 2001년도 반도체설계교육센터로부터의 부분적인 지원에 의하여 이루어진 연구로서, 관계부처에 감사 드립니다.

## 참 고 문 헌


[1] Rajesh K. Gupta and Yervant Zorizn. "Introducing Core-Based System Design", IEEE Design & Test of Computers, vol. 14, Issue 4, pp 15-25, Dec. 1997.

[2] Steven J.E. Wilton and Resve Saleh, "Programmable Logic IP Cores in SoC Design: Opportunities and Challenges", Custom Integrated Circuits, 2001, IEEE Conference on, pp 63-66, May, 2001.

[3] Rafael P. L. et al., "HW-SW Co-Design and Verification of a Multi-Standard Video and Image Codec", Quality Electronic Design, 2001 International Symposium on, pp 393-398, 2001.

[4] Zhang, Ma and Yao, " A Software/ Hardware Co-Design Methodology For Embedded Micro processor Core Design", IEEE Trans. on Consumer Electronics, vol. 45, no. 4, pp 1241-1246, Nov. 1999.

[5] Michael Gschwind et al., "FPGA Prototyping of a RISC Processor Core for Embedded Applications", Very Large Scale Integration (VLSI) Systems, IEEE Trans. vol. 9, pp 241 -250, April 2001.

[6] Z80 Family Databook, Zilog, 1994.

[7] Embedded Controller Handbook, Intel, 1988.

[8] http://semiconductor.hitachi.com/H8/h8_family.html

[9] http://www.panasonic.co.jp/semicon/e-index.html

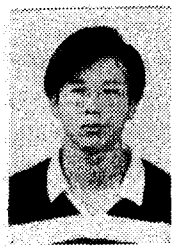
[10] Piguet, C. et al., "Low-Power Design of 8-b Embedded CoolRisc Microcontroller Cores", IEEE Journal of solid-state circuits, vol. 32, no. 7, pp 1067-1078, July 1997.

[11] 조경연 " 확장 명령어 32 비트 마이크로 프로세스에 관한 연구", 전자공학회지, vol. 36, no. 5, pp 11-20, May 1999.

[12] http://sources.redhat.com/newlib

[13] http://gcc.gnu.org


## 저 자 소 개

차영호(Young-Ho Cha)
부경대학교 전자공학과 공학사
부경대학교 전자공학과 공학석사
1999년-현재 부경대학교 전자공학과 박사과정
※관심분야 : CAD, VLSI 설계

조경연(Gyeong-Yeon Cho)
인하대학교 컴퓨터공학과 공학사
인하대학교 컴퓨터공학과 공학석사
인하대학교 컴퓨터공학과 공학박사
1991년-현재 부경대학교 부교수, 에이디칩스(ADC) 기술고문
※관심분야 : 전산기구조, 반도체회로설계, 시스템프로그래밍

최혁환(Hyek-Hwan Choi)
경북대학교 전자공학과 공학사
아리조나주립대학교 전자공학과 공학석사
아리조나주립대학교 전자공학과 공학박사
1994년-현재 부경대학교 부교수
※관심분야 : 반도체 소자 모델링, RF 집적회로설계