論文2003-40SD-6-7

FPGA를 이용한 시퀀스 제어용 32비트 마이크로프로세서 설계

(The Design of 32 Bit Microprocessor for Sequence Control Using FPGA)

梁 吾 *

(Oh Yang)

요 약

본 논문은 FPGA를 이용하여 시퀀스 제어용 32비트 마이크로프로세서를 설계하였다. 이를 위해 VHDL을 이용하여 톱-다운 방식으로 마이크로프로세서를 설계하였으며, 고속처리의 문제점을 해결하기 위해 프로그램 메모리부와 데이터 메모리부를 분리하여 설계함으로써 인스트럭션을 페치 하는 도중에 시퀀스 명령을 실행할 수 있는 Harvard 구조로 설계하였다. 또한 마이크로프로세서의 명령어들을 시퀀스제어에 적합하도록 RISC형태의 32 비트 명령어로 고정하여 명령어의 디코딩 시간과 데이터 메모리의 인터페이스 시간을 줄였다. 특히 설계된 마이크로프로세서의 실시간 디버깅 기능을 구현하기 위해 싱글 스텝 런, 일정 프로그램 카운터 브레이크, 데이터 메모리와 일치시 정지 기능 등을 구현함으로써 구현된 프로세서의 디버깅을 쉽게 하였다. 또한, 시퀀스제어에 적합한 펄스명령, 스텝 콘트롤 명령, 마스터 콘트롤 명령 등과 같은 비트 조작 명령과, BIN형과 BCD형 산술명령, 배럴 쉬프트명령 등을 구현하였다. 이와 같은 기능들을 FPGA로 구현하기 위하여 자이링스(Xilinx)사의 V600EHQ240(60만 게이트)과 Foundation 4.2i를 사용하여 로직을 합성하였다. Foundation 합성툴 환경에서 시뮬레이션과 실험에서 성공적으로 수행되었다. 본 논문에서 구현된 시퀀스 제어용 마이크로프로세서의 우수성을 보이기 위해 시퀀스제어용 명령어를 많이 가지고 있는 Hitachi사의 마이크로프로세서인 H8S/2148과 성능을 비교하여 본 논문에서 설계된 시퀀스 제어용 프로세서가 우수함을 확인하였다.

Abstract

This paper presents the design of 32 bit microprocessor for a sequence control using a field programmable gate array(FPGA). The microprocessor was designed by a VHDL with top down method, the program memory was separated from the data memory for high speed execution of sequence instructions. Therefore it was possible that sequence instructions could be operated at the same time during the instruction fetch cycle. In order to reduce the instruction decoding time and the interface time of the data memory interface, an instruction code size was implemented by 32 bits. And the real time debug operation was implemented for easeful debugging the designed processor with a single step run, PC break point run, data memory break point run. Also in this designed microprocessor, pulse instructions, step controllers, master controllers, BIN and BCD type arithmetic instructions, barrel shift instructions were implemented for sequence logic control. The FPGA was synthesized under a Xilinx's Foundation 4.2i Project Manager using a V600EHQ240 which contains 600,000 gates. Finally simulation and experiment were successfully performed respectively. For showing good performance, the designed microprocessor for the sequence logic control was compared with the H8S/2148 microprocessor which contained many bit instructions for sequence logic control. The designed processor for the sequence logic showed good performance.

Keyword : FPGA, VHDL, Microprocessor Sequence logic control

* 正會員, 淸州大學校 情報通信 工學部

(Dept. of Electronic Engineering, Chongju University) 接受日字:2002年10月9日, 수정완료일:2003年5月21日

Ⅰ. 서 론

최근 들어 디지털 논리회로의 고집적화로 디지털 논리회로 분야에 많은 발전을 가져왔고 시스템 온칩(SoC)이나 IP(Intellectual Property)에 대한 많은 관심 속에 비메모리분야에 대한 연구가 활발히 진행되고 있다. 특히 마이크로프로세서의 처리능력은 날로 고기능, 고성능, 고속화를 요구하고 있으며 최적의 시스템을 구축하기 위한 연구가 활발히 진행되고 있다. 또한, 소자(device)의 크기는 점점 작아지고 있고 소비전력 또한 점점 적어지고 있지만 가격은 저가일 것 등이 요구되는 추세에 있다[1]. 이와 같은 요구를 만족하기 위해 그동안 널리 사용되었던 CISC나 RISC형태의 범용 마이크로프로세서를 채택하여 시스템을 구축하였다[2, 3]. 특히 공장 자동화, 공작기기, 간이로봇, 프로세서 콘트롤, 자동차 조립공정 등과 같은 시퀀스제어에서는 임의 접점에 대한 상승 에지나 하강 에지 검출명령과 비트 조작명령, 스텝 콘트롤러(Step controller)와 같은 특수한 명령어가 전체 프로그램 중 80%이상을 차지하고 있기 때문에 시퀀스 전용의 명령어가 필요한 실정이다.

즉, 시퀀스제어를 위해서는 펄스연산이나 세븐 세그먼트(7-segment)와 같은 표시기에 연산결과를 출력하기 위해 BCD형태의 덧셈, 뺄셈, 곱셈, 나눗셈 연산이 필요하고, 단위 공정의 순차제어를 위해 스텝 콘트롤러, 마스터 콘트롤 등이 사용된다. 이를 위해서는 기존의 범용마이크로프로세서를 사용하여 로직연산, 쉬프트연산, 비트 조작연산 등을 수행해야하는 단점이 있다[4]. 이를 해소하기 위해 설계자 전용의 프로세서에 대한 연구가 활발히 증가하고 있고 이를 구현하기 위해 VHDL을 이용한 대용량의 FPGA를 적용하여 Top-down 방식으로 미리 디지털로직을 구현하고 있다[5-8]. [7]에서는 일부 간단한 시퀀스 명령어를 하드웨어적으로 구현한 예를 보였고 [8]에서는 FPGA를 이용한 시퀀스 로직 제어용 고속 프로세서를 설계하여 일부 시퀀스 명령어에 대한 처리시간을 100ns로 고속화를 구현하기도 하였다.

그러나 [8]에서는 21종류의 시퀀스 명령어만을 제시하였고 시퀀스제어에서 자주 사용되는 BCD 연산이나 스텝콘트롤, BIN 데이터변환, 산술 및 논리연산 등은 구현하지 않았으며 일부 시퀀스 명령어에 대해서 고속화의 방안을 제시하였다. 또다른 고속화 방법중 하나는 마이크로프로세서와 코프로세서(coprocessor)를 이용하여 고속화를 꾀하지만 이러한 방법 역시 비용의 상승효과를 가져오기 때문에 적합하지 못한 실정이다[9].

본 논문에서는 시퀀스 명령어에 대한 고속처리를 위해서 적절한 타이밍설계와 프로그램 메모리와 데이터 메모리의 버스를 분리하여 메모리의 접근을 최적화였으며, 시퀀스 처리명령어를 쉽게 디코딩할 수 있도록 명령어를 32 비트의 RISC 구조로 하였다. 특히 사용자 명령어 코드에 대한 병렬처리 방법 및 시퀀스 처리를 위한 비트 ALU를 구현함으로써 여러개의 비트 연산을 병렬로 처리함으로써 시퀀스 명령어에 대한 처리속도의 고속화를 구현하였다. 이와 같은 기능들을 VHDL을 이용하여 FPGA로 구현하기 위하여 Xilinx사의 시스템 게이트가 60만개인 V600EHQ240과 Foundation 4.2i Project Manager를 이용하여 로직을 구현하였다[10, 11].

또한, 시퀀스 전용의 프로세서를 설계한 후 이를 응용하는 과정에서 디버깅을 필요로 하는 MDS를 개발해야하는데 이에 대한 개발이 복잡하기 때문에 실시간 디버그 기능을 구현하여 쉽게 디버깅할 수 있도록 하였고, 이에 대한 운용은 윈도즈환경에서 실시간 디버깅 기능을 구현하였다. 아울러 설계된 마이크로프로세서의 우수성을 평가하기 위해 산업용 제어기기로 널리 사용되고 있는 내부 32비트/외부 16비트인 HITACHI사의 마이크로 콘트롤러인 H8S/2148[12, 13]과 비교 평가하여 본 논문에서 구현한 시퀀스 로직 제어용 프로세서가 우수함을 실험을 통해 입증하고자한다.

전체적인 논문의 구성은 2장에서 시퀀스 로직 콘트롤러의 명령어와 그래픽표현 설계사양과 전제조건을 설명하고, 시퀀스 로직 콘트롤러의 명령어 코드를 구성하였다. 또한, 시퀀스 로직 제어용 고속 프로세서에 대한 내부블록 구조를 고찰하다. 3장에서는 설계된 프로세서의 시뮬레이션과 각각의 명령에 대한 시뮬레이션 결과를 검토하고, 4장에서는 설계된 프로세서를 직접 FPGA로 구현한 후 시퀀스 로직 제어 시스템에 적용하여 성능을 비교 평가하여 설계된 프로세서의 우수성을 보이며, 마지막으로 5장에서 본 논문의 결론을 맺는다.

Ⅱ. 시퀀스 로직 제어용 마이크로프로세서 설계

1. 시퀀스 로직 명령어의 종류와 그래픽표현

공장자동화나 산업용 시퀀스제어에 많은 사용되는 명령어는 비트 처리 명령어와 워드처리 명령어로 나누어지고 있고 명령어 수는 수백 종에 이르며 이중 대표적으로 많이 사용되는 명령어를 나타내면 <표 1>과 같다[14].

표 1. 시퀀스 로직 명령어에 대한 기호와 표현
Table 1. Symbol and representations for sequence logic instructions.

명령어	기 호	기 능
LD	⊣⊢	논리 연산 시작
LDI	⊣/⊢	논리 부정 연산 시작
AND	⊣⊢	논리 적
ANDI	⊣/⊢	논리 적 부정
OR	⊣⊢	논리 화
ORI	⊣/⊢	논리 화 부정
ANB		논리블럭 AND
ORB		논리블럭 OR
LDP	⊣↑⊢	논리 연산 시작 상승펄스
LDF	⊣↓⊢	논리 연산 시작 하강펄스
ANDP	⊣↑⊢	논리 적 상승펄스
ANDF	⊣↓⊢	논리 적 하강펄스
ORP	⊣↑⊢	논리 화 상승펄스
ORF	⊣↓⊢	논리 화 하강펄스
MPS	⊣⊢	분기 개시
MRD	⊣⊢	분기
MPP	⊣⊢	분기 종료
INV	⊣/⊢	연산결과 반전

명령어	기 호	기 능
OUT	( )	접점 출력
SET	SET D	접점 SET
RST	RST D	접점 RESET
PLS	PLS D	상승에지에서 1스캔 출력
PLF	PLF D	하강에지에서 1스캔 출력
MC	MC n	마스터 콘트롤 시작
MCR	MCR n	마스터 콘트롤 끝
NOP		무처리
LD=	LD= S1 S2	비교연산 및 논리 시작
AND=	AND= S1 S2	비교연산 및 논리 적
OR=	OR= S1 S2	비교연산 및 논리 화
MOV	MOV S D	데이터 이동
BCD	BCD S D	BIN을 BCD로 변환
BIN	BIN S D	BCD를 BIN로 변환
ROL	ROL D n	왼쪽으로 n회 회전
ROR	ROR D n	오른쪽으로 n회 회전
CALL	CALL n	n번지 프로그램 호출
JMP	JMP n	JME n 번지로 점프

2. 마이크로프로세서의 설계사양

① 워드 명령어 종류

명령어 종류		어드레싱 모드	
ADD	W, L	#imm,Rd	Rs,Rd
SUB	W, L	#imm,Rd	Rs,Rd
MUL	B, W, L	#imm,Rd	Rs,Rd
DIV	B, W, L	#imm,Rd	Rs,Rd
AND	W, L	#imm,Rd	Rs,Rd
XOR	W, L	#imm,Rd	Rs,Rd
CMP	W, L	#imm,Rd	Rs,Rd
SHLL	W, L	#imm,Rd	Rs,Rd
SHAR	W, L	#imm,Rd	Rs,Rd
ROR	W, L	#imm,Rd	Rs,Rd
NOT	W, L		Rs,Rd
BCD	W, L		Rs,Rd
BADD	W, L		Rs,Rd
LD	B, W, L	@addr, Rd @Rs, Rd @Rs+, Rd @Rs-,Rd @(Rs+offset), Rd	
CLR	W, L	@addr @Rd @Rd+ @Rd-	
NOP			
RET			
RETC		Fn	
RETN		Fn	
PUSH	W, L	Rs	

명령어 종류		어드레싱 모드	
ADDC	W, L	#imm,Rd	Rs,Rd
SUBB	W, L	#imm,Rd	Rs,Rd
MULS	B, W, L	#imm,Rd	Rs,Rd
DIVS	B, W, L	#imm,Rd	Rs,Rd
OR	W, L	#imm,Rd	Rs,Rd
MOV	W, L	#imm,Rd	Rs,Rd
CMPS	W, L	#imm,Rd	Rs,Rd
SHLR	W, L	#imm,Rd	Rs,Rd
ROL	W, L	#imm,Rd	Rs,Rd
RCL	W, L	#imm,Rd	Rs,Rd
NEG	W, L		Rs,Rd
BIN	W, L		Rs,Rd
BSUB	W, L		Rs,Rd
ST	B, W, L	Rs, @addr Rs, @Rd Rs, @Rd+ Rs, @Rd- Rs, @(Rd+offset)	
JMP		addr	
JMPC		Fn, addr	Fn, Rd
JMPN		Fn, addr	Fn, Rd
DJNZ		Rd, addr	
CALL		addr	
CALLC		Fn, addr	Fn, Rd
CALLN		Fn, addr	Fn, Rd
POP	W, L	Rd	

위의 명령어에서 B, W, L은 각각 8비트, 16비트 32비트 연산을 나타내며 Fn 상태레지스터의 플래그번호로 총 16개로 구성된다. 또한, 레지스터는 16비트로 16개가 있으며 2개씩 조합하여 32비트로 8개가 있고 R15, R16은 스택포인터(SP)로 사용된다. 아울러 명령어 뒤의 C, N은 정논리와 부논리를 각각 나타낸다.

② 비트 명령어 종류

명령어 종류	어드레싱 모드	명령어 종류	어드레싱 모드
BLD	#bit, @addr #bit, @Rd Rs, @addr Rs, @Rd	BLDI	#bit, @addr #bit, @Rd Rs, @addr Rs, @Rd
BAND		BANDI	
BOR		BORI	
BXOR		BXORI	
BOUT		BOUTI	
BSET		BRST	
BLDP		BLDF	
BANDP		BANDF	
BORP		BORF	
PLS_CK		PLF_CK	
LD_BF		ST_EF	
SET		RST	
LDS	Step_addr, Step_no	LDSI	Step_addr, Step_no
ANDS		ANDSI	
ORS		ORSI	
OUTS		SETS	
BMOV	Bit_no, Bit_no	INV	
MC	kk:4	MCR	kk:4
ANB		ORB	
MPS		MRD	
MPP		MOVF	#imm, Flag Rs, Flag
ANDF	#imm, Flag Rs, Flag	ORF	#imm, Flag Rs, Flag
XORF	#imm, Flag Rs, Flag		

위의 명령어에서 Flag 레지스터는 16비트로 구성되고 Step_no는 00~99의 범위를 가지며 100가지의 공정을 단계별로 설정할 수 있는 구조로 하였다. 또한, kk:4는 마스터 콘트롤을 위한 정수로 0~15로 설정할 수 있다. FPGA를 이용한 시퀀스 제어용 32비트 마이크로프로세서를 설계시 명령어의 길이는 32비트로 구성하여 RISC형태로 하였고 명령어의 종류는 시퀀스처리를 위해 비트 처리용 118종과 워드처리를 위해 164종으로 구성된다. 설계된 FPGA의 편리한 디버깅을 위해 한 명령어씩 수행하는 1 스텝 운전(single step run), 수행중인 프로그램 카운터(PC)의 내용과 프로그램을 정지하고자하는 어드레스가 일치했을 경우 정지하는 프로그램 카운터 브레이크 운전(pc break run), 정지하고자 하는 데이터 어드레스가 일치하고 읽기 또한 쓰기 신호가 일치했을 경우 정지하는(Data Memory Address break run) 방법 등의 기능이 구현된다. 이와 같은 디버그 기능은 실시간 디버그 기능을 구현하므로 실제 상황과 동일한 방법으로 실현할 수 있는 장점을 가지고 있다. 또한 시퀀스 제어기에서는 타이머가 필수적으로 사용되기 때문에 32비트 타이머 1개를 두었으며 다른 MPU와 인터페이스를 위해 외부로부터 프로그램 운전/정지 기능을 부가하여 편리성을 도모하였다.

3. 마이크로프로세서의 구성 및 설계

시퀀스 제어용 32비트 마이크로프로세서를 VHDL을 이용하여 톱-다운(Top-down) 방식으로 구현하기 위한 내부구성을 <그림 1>과 같이 총 15개의 블록으로 구성하였다. 시퀀스 제어용 마이크로프로세서에 대한 고속처리를 위해 프로그램 메모리와 데이터 메모리의 버스를 분리하여 설계하였다. 프로세서의 프로그램 메모리는 32비트로 하였고 데이터 메모리는 16비트로 각각 구성하였다. <그림 1>에서의 PM 블록에서는 인스트럭션 레지스터를 32비트로 구성하기 위해 16비트 메모리 2개를 사용하였고 이를 위해 WR 신호를 상위 16비트인 PM_WRH-와 하위 16비트인 PM_WRL-로 각각 분리하였다.

DM 블록에서는 DM_16BIT가 '0'일 때는 8비트 2개의 메모리로 '1'일 때는 16비트 1개로 각각 인터페이스하게 하였으며 8비트 2개로 사용할 때 쓰기에는 상위와 하위번지를 분리하기 위해 DM_WRH-와 DM_WRL-신호를 사용하였다. 또한 16비트 메모리를 사용할 때 상위 8비트와 하위 8비트를 제어하기 위해 LB-와 UB-신호를 사용하였으며 쓰기 신호를 위해 DM_WRH-와 DM_WRL-를 이용함으로써 SRAM의 활용도를 높였다. 또한, 비트 처리용 ALU(BALU)와 워드처리용 ALU(WALU) 코어가 각각 있으며 워드처리용 ALU 코어는 기본적으로 32비트로 처리되며 특별히 32비트와 32비트의 곱셈결과와 32비트 나누기 32비트시 나눗셈결과는 64비트의 결과를 얻도록 하여 내부버스는 32비트와 64비트가 혼용되어 설계되었다. CPU 블록에서는 CS_SEL 신호로 범용프로세서에서 직접 어드레스가 디코드된 CS0, CS1, CS2, CS3와 디코드 되지 않은 어드레스 신호를 선택할 수 있게 설계하였다. CS 블록에서는 범용 프로세서의 어드레스 버스가 연결되며 시퀀스 제어용 프로세서의 디코드 신호로 사용되고 또한, 외부에 연결되는 플래시메모리나 입출력 디바이스 등 예비 선택신호를 발생하도록 하였다.

시퀀스 제어에 필수적으로 사용되는 Timer INT 블록에서는 32비트 타이머가 구성되고 인터럽트 수행을 위한 레지스터들을 제어하고 있으며, 인터럽트 발생시 인터럽트 루틴을 수행하게 한다. CLK 블록에는 시퀀스 제어용 프로세서를 제어하는 HOLD와 GA_RUN이 연결된다. 만약 외부로부터 HOLD의 입력신호가 Low가

되면 프로세서가 처리할 수 있는 명령어의 경우에는 GA_RUN을 High로 출력하고 명령어를 수행한다. 그러나 FPGA가 수행 불가능한 명령어 또는 해독할 수 없는 명령어일 경우에는 GA_RUN을 Low로 출력하여 범용 프로세서에 제어권을 넘겨주게 된다.

<그림 1>에서 설계된 프로세서는 총 20비트로 구성된 프로그램 카운터를 기본으로 하여 동작하며 HOLD가 High가 되면 정지하는 구조로 되어있다. 또한 내부 클럭 발생을 위해 외부 클럭 50 MHz를 입력하였다.

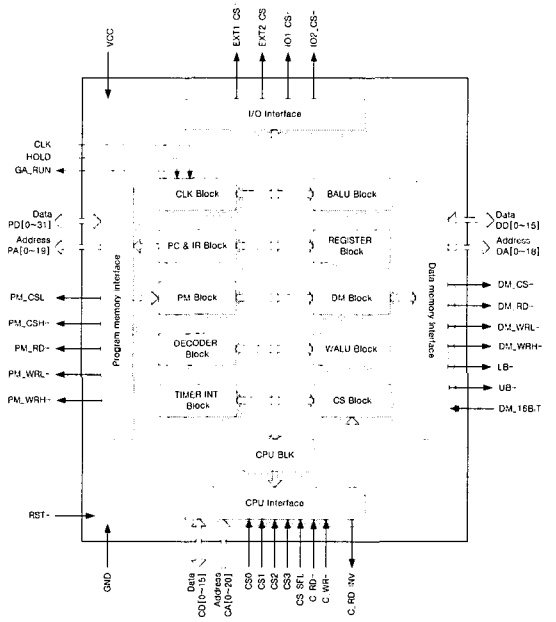


그림 1. 시퀀스 제어용 마이크로프로세서의 내부 구성도
Fig. 1. Block diagram of microprocessor for sequence logic control.

4. 시퀀스명령에 대한 고속처리 구현

시퀀스제어를 위해서는 순수한 비트 동작용 명령어가 약 30%, 펄스처리용 명령어가 50%, 나머지 20%는 기타 산술연산 명령어가 차지하고 있다. 특히 산술명령에서는 BCD 데이터 형태가 주류를 이루고 있으나 범용의 마이크로프로세서에서는 이러한 명령어가 구비되어있지 못하기 때문에 프로그램이 증가하고 수행속도가 느려지는 단점을 가지고 있다. 본 논문에서는 이와 같이 많이 사용되는 시퀀스 명령어를 직접 설계하여 하드웨어적으로 구현함으로써 사용자프로그램을 고속으로 처리하고 아울러 명령어를 구현할 때 사용되는 명령어의 길이를 줄이고자 하였다. 대부분 사용자 프로그램은 수10~수100 킬로바이트의 프로그램 메모리가 필요하여 입력유닛으로부터 데이터를 읽고 시퀀스연산을 한 후 출력유닛에 출력데이터를 내보내는데 소요되는 시간을 스캔 시간이라 하며 이 시간을 줄이는 것이 매우 중요하다. 이러한 이유는 속응제어와 빠른 입출력 제어를 위해 가능한 각각의 명령처리 시간을 줄이는 것이 전용 마이크로프로세서를 설계하는 이유이기도 하다. 이와 같은 속응제어를 위해 <그림 2>의 내부구조와 같이 프로그램 메모리와 데이터 메모리를 분리하여 설계함으로써 <그림 3(b)>와 같이 시퀀스 프로그램을 페치 하면서 동시에 데이터 메모리를 읽거나 쓸 수 있는 독립된 버스를 갖는 것이 기본적인 고속화 방법이다. 또한, 282종의 명령어를 디코딩할 때 디코딩시간을 줄이기 위해 시퀀스 제어명령의 특성에 적합하도록 OP 코드를 작성하여 시퀀스 명령어 해독시간을 줄였다. 또한, 비트처리 명령의 경우 최하위 4비트를 이용하여 비트의 번호를 나타내었고 특히 명령처리에서 길게 소요되는 또 다른 요소로 데이터메모리를 형성할 때 소비되는 시간을 줄이기 위해 외부 데이터 메모리를 인터페이스 하는 명령어의 코드를 별도로 분리하여 인스트럭션 래치와 동시에 데이터 메모리를 형성하도록 설계하여 고속화를 실현하였다.

아울러 시퀀스제어의 고속처리를 위해 비트 ALU를 별도로 설계하였다. 본 비트 ALU에서는 펄스처리 명령의 경우 다음과 같이 하드웨어적으로 병렬동작이 수행되도록 구현하여 고속화를 꾀하였다.

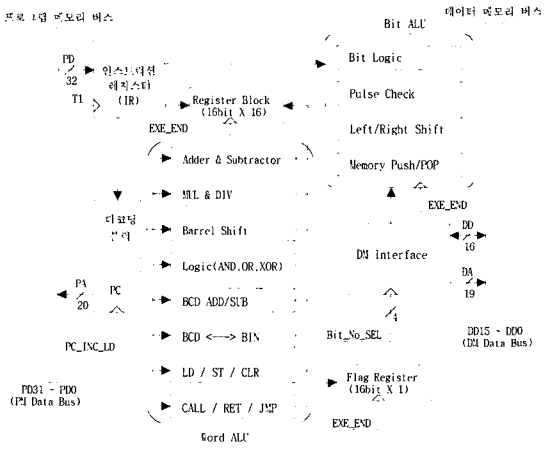
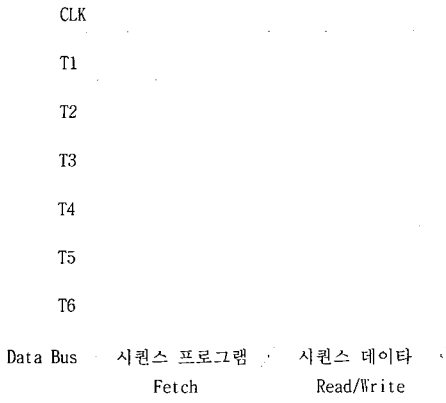
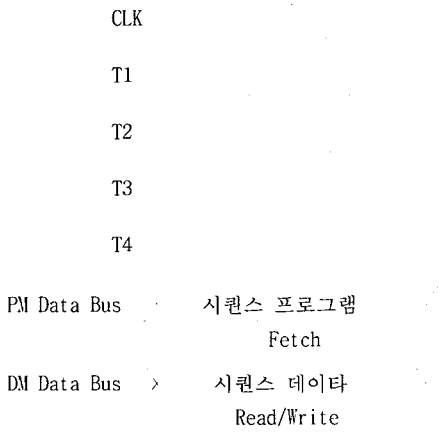


그림 2. 시퀀스 제어용 마이크로프로세서의 구조
Fig. 2. Structure of microprocessor for sequence logic control.



(a) H8S/2148의 타이밍도



(b) 본 논문에서 설계된 프로세서의 타이밍도

그림 3. 시퀀스 제어를 위한 마이크로프로세서의 타이밍도

Fig. 3. Timing diagram of microprocessor for sequence control.

Data Memory.비트번호 → Pulse Bit

Bit Result를 반전 → Data Memory.비트번호

Bit Result와 Pulse Bit를 AND, OR, XOR 연산 → Bit Result

즉, 비트 데이터의 읽기, 비트 데이터의 쉬프트와 비트 마스크 및 데이터 메모리에 쓰기 동작 등을 병렬로 처리하여 하나의 명령어로써 여러 가지의 동작을 병렬로 연산하도록 구현함으로써 범용의 마이크로프로세서를 사용하여 시퀀스제어명령을 구현할 때보다 2~10여 배 이상 빠르게 수행 가능토록 구현하였다.

Ⅲ. 시뮬레이션 및 결과 검토

본 논문에서 제안된 기능을 FPGA로 구현하기 위해 프로그램 메모리 인터페이스부, 데이터 메모리 인터페이스부, I/O 인터페이스부, 비트 ALU, 워드 ALU 및 레지스터부, 인스트럭션 및 디코더부, 타이머부 등 각각의 모듈을 설계하여 VHDL로 기술하였다. 이와 같이 기술된 VHDL을 Xilinx사에서 제공되는 Foundation V4.2i를 이용하여 로직을 합성 및 시뮬레이션 하였고 하드웨어 구현을 위해 시스템 게이트가 600,000 게이트에 해당되며 핀의 형태는 240핀 HQFP 형태인 V600EHQ을 사용하였다. 이때 배치 및 배선(P & R)의 과정을 통해 FPGA의 사용율은 6,912개의 셀중 5,536개를 사용함으로써 80%가 됨을 확인하였다. 아울러 동작 주파수는 50 MHz를 사용하였고 프로그램메모리와 데이터메모리의 액세스시간은 실제 구현될 SRAM과 동일하게 55 ns로 가정하여 시뮬레이션을 하였다.

<그림 4>에서는 ADD, SUB 명령어의 시뮬레이션 결과는 나타내었다. 여기서 .W와 .L은 각각 16비트와 32비트를 나타내고 있고 GA_RUN이 High가 되는 것은 현재 프로세서가 명령어를 처리하고 있음을 나타내고 있다.

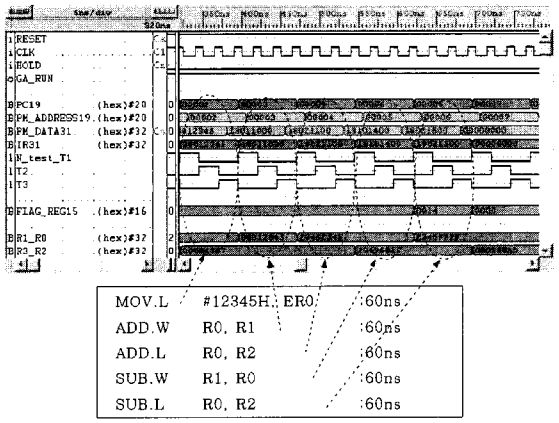


그림 4. 덧셈, 뺄셈 명령어에 대한 시뮬레이션 결과

Fig. 4. Simulation result for ADD and SUB instructions.

<그림 5>는 32비트 배럴 쉬프트 명령에 대한 각각의 시뮬레이션 결과를 보이고 있다. 이러한 배럴 쉬프트는 FPGA 내부 처리 명령어로써 명령어 수행 시간은 각각 60ns이다.

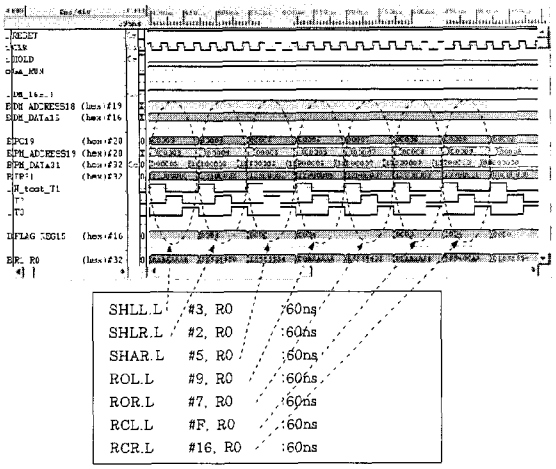


그림 5. 배럴 쉬프트 명령어에 대한 시뮬레이션 결과
Fig. 5. Simulation result for barrel shift instructions.

<그림 6>은 곱셈에 대한 시뮬레이션 결과로써 8비트, 16비트, 32비트 곱셈에 대한 시뮬레이션이며 각각의 명령에 대한 시뮬레이션 결과는 16비트 32비트 및 64비트의 결과를 출력하는 구조로 되어있다.

이러한 곱셈 명령어는 시스템 게이트 수를 줄이기 위해 한 클럭마다 쉬프트와 덧셈알고리즘을 이용하였고 8비트의 경우 12 클럭, 16 비트의 경우 20 클럭, 32 비트의 경우 36클럭이 필요하다.

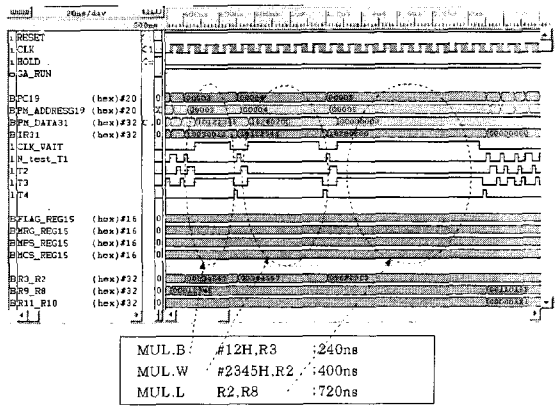


그림 6. 곱셈 명령어에 대한 시뮬레이션 결과
Fig. 6. Simulation result for MUL instructions.

<그림 7>은 비트 처리 명령어 중 펄스처리 명령어에 대한 시뮬레이션 결과를 보이고 있다. 처음에 수행되는 명령어는 상승 에지를 체크한 후 연산된 결과를 플래그레지스터와 외부 데이터 메모리에 쓰기 동작을 수행함을 보이고 있다. 읽기와 쓰기 동작이 연속해서 이루어지고 명령어의 수행시간은 7개의 머신 사이클이 필요하여 140ns에 처리된다.

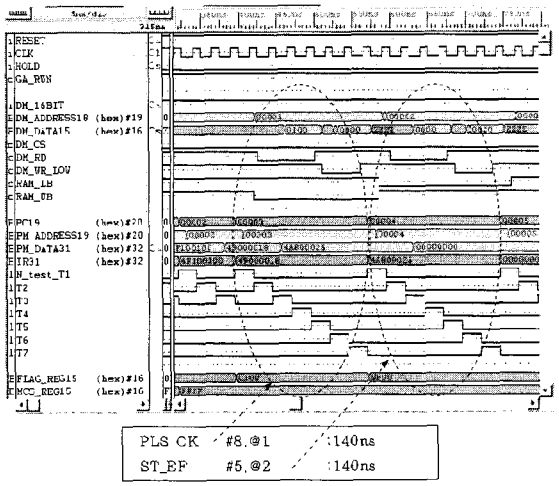


그림 7. 펄스 명령어에 대한 시뮬레이션 결과
Fig. 7. Simulation result for pulse instructions.

Ⅳ. 실험 및 검토

본 논문에서 설계된 시퀀스 제어용 32비트 마이크로프로세서의 성능 평가를 위해 <그림 8>과 같이 시스템을 구성하였다.

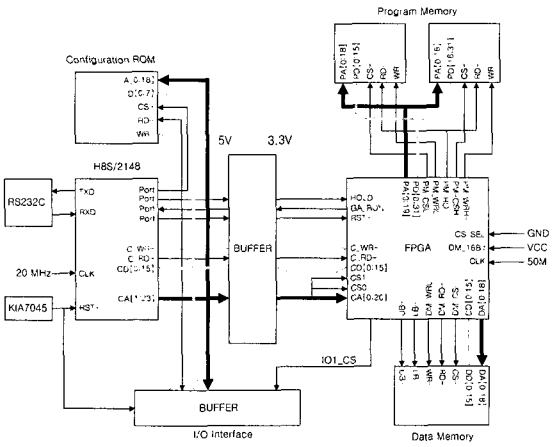


그림 8. 마이크로프로세서의 성능평가를 위한 전체 구성도
Fig. 8. Configuration for benchmarking of microprocessor.

<그림 8>에서 H8S/2148은 Hitachi사의 범용 마이크로콘트롤러이며 본 논문에서 설계된 프로세서와 성능평가 대상으로 선정하였으며 H8S/2148은 시퀀스제어에

적합하도록 비트처리 명령과 워드처리용 명령어가 많이 구비된 마이크로프로세서이며 널리 사용되고 있다. 외부 클럭은 시스템의 최대성능을 보이기 위해 20MHz를 사용하였다. 소자들의 전압 레벨을(3.3V와 5V) 변환해 주기 위해 74LVC164245를 사용하였으며, 백업전류가 수 uA이며 처리속도가 55ns인 16비트 CMOS SRAM(K6T4016V3C-TB55)를 프로그램 메모리로 2개, 데이터 메모리로 1개를 각각 사용하였다. 또한 시퀀스 제어의 1스캔이 완료되면 입력 카드로부터는 입력을 읽고 출력 카드로 연산된 데이터를 출력 할 수 있는 구조로 입출력을 각각 설계하였다.

<그림 9>는 덧셈, 뺄셈, 배럴 쉬프트, 마스터 컨트롤, 메모리 푸시 팝과 관련된 명령어 수행에 대한 실험결과이다.

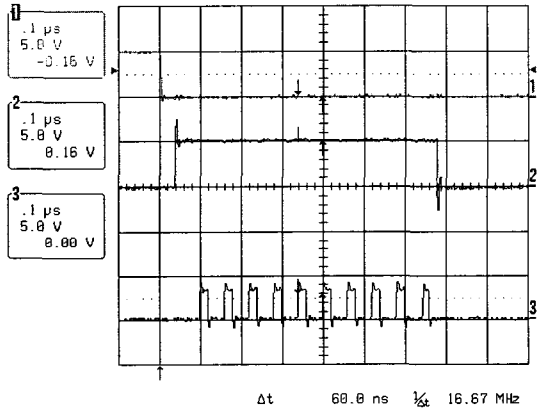
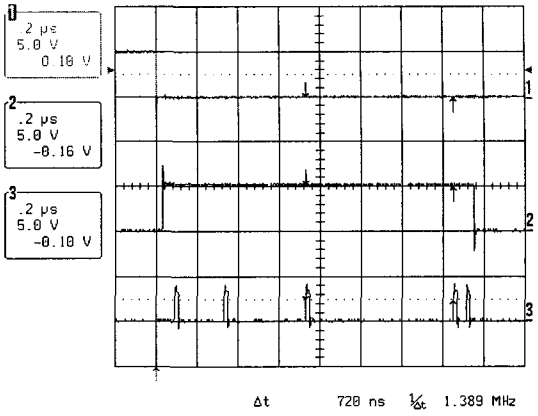


그림 9. 워드 명령어와 비트 명령어에 대한 실험결과
Fig. 9. Experimental result for word and bit instructions.

<그림 9>에서 채널 1은 FPGA의 명령을 수행하라는 지령(HOLD)으로 Low시 수행되고 채널 2는 마이크로프로세서가 수행될 때 High를 출력하는 GA_RUN 신호이며 채널 3은 각각의 명령어 종료(EXE_END)를 관측한 파형이다. 다른 프로세서와 통신을 위해 FPGA 정지 명령(END)을 수행하도록 하였다. <그림 9>에서 수행된 명령어는 다음과 같다.

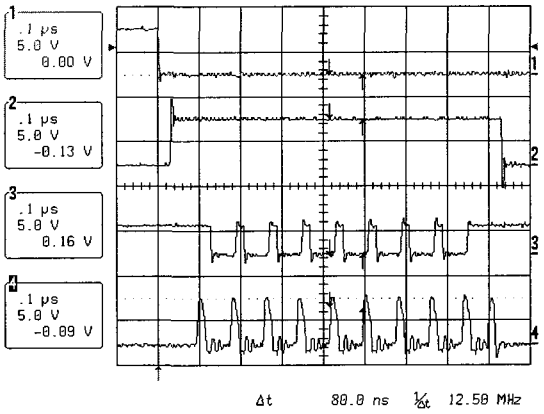
ADD	SUB	SHLL	ROL	
MC	MCR	MPS	MPP	END

<그림 10>은 8비트(.B), 16비트(.W), 32비트(.L) 곱셈 명령어에 대한 수행결과이다.



MUL.B	MUL.W	MUL.L	END

그림 10. 곱셈 명령어에 대한 실험결과
Fig. 10. Experimental result for MUL instructions.



BLD	BLDI	BAND	BANDI	BOR
BORI	BXOR	BXORI	END	

그림 11. 비트 읽기 명령에 대한 실험결과
Fig. 11. Experimental result for bit read instructions.

<그림 11>은 외부데이터 메모리 읽기 명령어이기 때문에 채널 3의 DM_RD- 신호가 매 명령어마다 나타나고 있다. 채널 4는 명령어의 종료 신호(EXE_END)를 나타내고 있다.

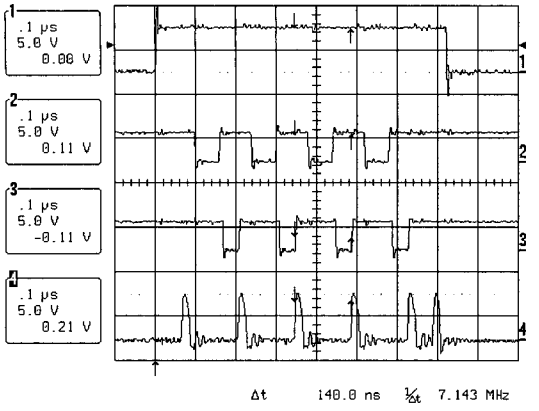


그림 12. 펄스 명령에 대한 실험결과
Fig. 12. Experimental result for pulse instructions.

표 2. 참고문헌 [8]의 각각 명령어에 대한 처리시간 비교표
Table 2. Execution time table of each instructions for reference[8].

명령어	[8]에서 설계된 프로세서의 경우		DSP(TMS320C32)를 이용한 경우				MCU(H8/325)를 이용한 경우		기존PLC의 경우 (ASIC 사용)	
			컴파일 방법		인터프리터 방법					
	수행 속도	명령어 길이	수행 속도	명령어 길이	수행 속도	명령어 길이	수행 속도	명령어 길이	수행 속도	명령어 길이
LD	100ns	2 바이트	850ns	28 바이트	1.9us	4 바이트	2.7us	8 바이트	200ns	4 바이트
LDI	100ns	2 바이트	850ns	28 바이트	1.9us	4 바이트	2.7us	8 바이트	200ns	4 바이트
OR	100ns	2 바이트	600ns	20 바이트	1.9us	4 바이트	2.1us	6 바이트	200ns	4 바이트
ORI	100ns	2 바이트	600ns	20 바이트	1.9us	4 바이트	2.1us	6 바이트	200ns	4 바이트
AND	100ns	2 바이트	550ns	16 바이트	1.9us	4 바이트	2.1us	6 바이트	200ns	4 바이트
ANDI	100ns	2 바이트	550ns	16 바이트	1.9us	4 바이트	2.1us	6 바이트	200ns	4 바이트
OUT	200ns	2 바이트	750ns	28 바이트	2.0us	4 바이트	3.6us	10 바이트	300ns	4 바이트
SET	200ns	2 바이트	700ns	28 바이트	1.8us	4 바이트	4.8us	12 바이트	300ns	4 바이트
RST	200ns	2 바이트	700ns	28 바이트	1.8us	4 바이트	4.8us	12 바이트	300ns	4 바이트
MC	100ns	2 바이트	500ns	20 바이트	1.6us	4 바이트	2.4us	8 바이트	800ns	12 바이트
MCR	100ns	2 바이트	700ns	20 바이트	1.9us	4 바이트	2.4us	8 바이트	200ns	4 바이트
PLS	400ns	4 바이트	1,050ns	44 바이트	3.4us	4 바이트	24us	8 바이트	800ns	12 바이트
PLF	400ns	4 바이트	1,050ns	44 바이트	3.4us	4 바이트	24us	8 바이트	800ns	12 바이트
NOP	100ns	2 바이트	50ns	4 바이트	1.3us	4 바이트	0.6us	2 바이트	200ns	4 바이트
NOT	100ns	2 바이트	50ns	4 바이트	1.3us	4 바이트	0.6us	2 바이트	200ns	4 바이트
ANB	100ns	2 바이트	250ns	4 바이트	1.9us	4 바이트	1.2us	4 바이트	200ns	4 바이트
ORB	100ns	2 바이트	250ns	4 바이트	1.6us	4 바이트	1.2us	4 바이트	200ns	4 바이트
MPS	100ns	2 바이트	250ns	4 바이트	1.6us	4 바이트	1.2us	4 바이트	200ns	4 바이트
MRD	100ns	2 바이트	250ns	4 바이트	1.6us	4 바이트	1.2us	4 바이트	200ns	4 바이트
MPP	100ns	2 바이트	250ns	4 바이트	1.6us	4 바이트	1.2us	4 바이트	200ns	4 바이트
END	100ns	2 바이트	250ns	4 바이트	1.3us	4 바이트	0.6us	2 바이트	200ns	4 바이트

<그림 12>는 펄스 명령어에 대한 실험결과 파형이다. 채널 1은 명령어를 수행하고 있음을 나타내는 GA_RUN 신호이고, 채널 2는 DM_RD-, 채널 3은 DM_WR_LOW-, 채널 4는 명령어의 종료 신호를 각각 나타낸다.

표 3. 본 논문에서 설계한 프로세서와 H8S/2148의 워드명령어 수행시간 비교
Table. 3. Execution time comparisons of designed processor and H8S/2148 for word instructions.

명령어 종류		본 논문의 프로세서			H8S/2148		
		수행시간	싸이클 수	수행 코드수	수행시간	싸이클 수	수행 코드수
ADD, SUB	16비트	60ns	3	4 바이트	150ns	3	2 바이트
	32비트	60ns	3	4 바이트	150ns	3	2 바이트
AND, OR, XOR, MOV,NEG, NOT	16비트	60ns	3	4 바이트	150ns	3	2 바이트
	32비트	60ns	3	4 바이트	300ns	6	4 바이트
CMP(부호 없는 비교)	16비트	60ns	3	4 바이트	150ns	3	2 바이트
	32비트	60ns	3	4 바이트	150ns	3	2 바이트
CMPS(부호 있는 비교)	16비트	60ns	3	4 바이트	2850ns	57	24 바이트
	32비트	60ns	3	4 바이트	2850ns	57	24 바이트
MUL(부호 없는 곱셈)	8비트	240ns	12	4 바이트	650ns	13	2 바이트
	16비트	400ns	20	4 바이트	1050ns	21	2 바이트
	32비트	720ns	36	4 바이트	5400ns	108	16 바이트
MULS(부호 있는 곱셈)	8비트	240ns	12	4 바이트	800ns	16	4 바이트
	16비트	400ns	20	4 바이트	1200ns	24	4 바이트
	32비트	720ns	36	4 바이트	6000ns	120	24 바이트
DIV(부호 없는 나눗셈)	8비트	240ns	12	4 바이트	700ns	14	2 바이트
	16비트	400ns	20	4 바이트	1100ns	22	2 바이트
	32비트	720ns	36	4 바이트	5600ns	112	16 바이트
DIVS (부호 있는 나눗셈)	8비트	240ns	12	4 바이트	850ns	17	4 바이트
	16비트	400ns	20	4 바이트	1250ns	25	4 바이트
	32비트	720ns	36	4 바이트	6200ns	124	24 바이트
LD	8비트	100ns	5	4 바이트	600ns	12	6 바이트
	16비트	100ns	5	4 바이트	600ns	12	6 바이트
	32비트	160ns	8	4 바이트	750ns	15	8 바이트
ST	8비트	80ns	4	4 바이트	600ns	12	6 바이트
	16비트	80ns	4	4 바이트	600ns	12	6 바이트
	32비트	140ns	7	4 바이트	750ns	15	8 바이트
PUSH	16비트	80ns	4	4 바이트	350ns	7	2 바이트
	32비트	140ns	7	4 바이트	650ns	15	4 바이트
POP	16비트	100ns	5	4 바이트	350ns	7	2 바이트
	32비트	160ns	6	4 바이트	650ns	13	4 바이트
ROTL, ROTR, ROTXL, ROTXR,SHAR, SHLL, SHLR, SHLR	16비트	60ns	3	4 바이트	150ns	3	2 바이트
	32비트						
NOP		60ns	3	4 바이트	150ns	3	2 바이트
JMP, JMPC, JMPN		120ns	6	4 바이트	300ns	6	2 바이트
DJNZ		120ns	6	4 바이트	600ns	12	6 바이트
CALL, CALLC, CALLN		140ns	7	4 바이트	300ns	6	2 바이트
RET, RETC, RETN		160ns	8	4 바이트	600ns	12	2 바이트
BCD, BIN	16비트	160ns	8	4 바이트	9150ns	183	120 바이트
	32비트	240ns	12	4 바이트	14550ns	291	200 바이트
BADD(BCD 덧셈)	16비트	60ns	3	4 바이트	3300ns	66	52 바이트
	32비트	60ns	3	4 바이트	4950ns	99	74 바이트
BSUB(BCD 뺄셈)	16비트	60ns	3	4 바이트	3300ns	66	52 바이트
	32비트	60ns	3	4 바이트	4950ns	99	74 바이트

본 논문에서 설계된 시퀀스 제어용 마이크로프로세서의 종합적인 성능을 비교하기 위해 참고문헌 [8]에서의 처리속도를 <표 2>에 나타내었고, Hitachi사의 H8S/2148과 본 논문에서 설계된 프로세서의 성능 평가를 위해 비교 실험한 결과를 <표 3>과 <표 4>에 각각 나타내었다. <표 3>에서는 시퀀스 명령어 중 자주 사용

표 4. 본 논문에서 설계한 프로세서와 H8S/2148의 비트 명령어 수행시간 비교

Table 4. Execution time comparisons of designed processor and H8S/2148 for bit instructions.

명령어 종류	본 논문의 프로세서			H8S/2148		
	수행시간	싸이클 수	수행코드수	수행시간	싸이클 수	수행코드수
BLD. BLDI	80ns	4	4 바이트	250ns	5	8 바이트
BOR. BORI	80ns	4	4 바이트	200ns	4	6 바이트
BAND. BANDI	80ns	4	4 바이트	200ns	4	6 바이트
BXOR. BXORI	80ns	4	4 바이트	200ns	4	6 바이트
BOUT. BOUTI	140ns	7	4 바이트	300ns	6	8 바이트
BSET. BRST	140ns	7	4 바이트	400ns	8	10 바이트
ANDB. ORB	60ns	3	4 바이트	100ns	2	4 바이트
BLDP.BLDF	140ns	7	4 바이트	1600ns	32	34 바이트
BANDP. BANDF	140ns	7	4 바이트	1600ns	32	30 바이트
BORP. BORF	140ns	7	4 바이트	1550ns	31	28 바이트
LD BF	80ns	4	4 바이트	250ns	5	8 바이트
ST EF	140ns	7	4 바이트	300ns	6	8 바이트
PLS CK. PLF CK	140ns	7	4 바이트	2000ns	40	36 바이트
SET. RST	140ns	7	4 바이트	300ns	6	8 바이트
LDS. LDIS(스텝명령)	80ns	4	4 바이트	550ns	11	10 바이트
ANDS. ANDIS (스텝명령)	80ns	4	4 바이트	450ns	9	10 바이트
ORS. ORIS(스텝명령)	80ns	4	4 바이트	450ns	9	10 바이트
OUTS(스텝명령)	80ns	4	4 바이트	450ns	9	8 바이트
SETS(스텝명령)	160ns	8	4 바이트	1050ns	21	22 바이트
BMOV	60ns	3	4 바이트	100ns	2	4 바이트
MC. MCR	60ns	3	4 바이트	150ns	3	6 바이트
INV. MRD	60ns	3	4 바이트	50ns	1	2 바이트
MPS. MPP	60ns	3	4 바이트	100ns	2	4 바이트
MOVF	60ns	3	4 바이트	100ns	2	4 바이트
ANDF	60ns	3	4 바이트	100ns	2	4 바이트
ORF	60ns	3	4 바이트	100ns	2	4 바이트
XORF	60ns	3	4 바이트	100ns	2	4 바이트

되는 워드명령어에 대한 비교표이고 <표 4>는 비트 명령어에 대한 수행시간을 비교하였다. <표 2>에서 기본적인 시퀀스 명령어는 100ns로 처리되며 펄스명령어는 400ns로 처리되었다. 그러나 본 논문에서 구현된 마이크로프로세서는 기본적인 시퀀스 명령어는 60ns~80ns로 처리되나 펄스명령어의 경우는 140ns로 수행됨으로써 2.8배 이상 빠름을 알 수 있다.

V. 결 론

본 논문에서는 FPGA를 이용하여 PLC의 시퀀스 제어를 위한 마이크로프로세서를 설계하였다. 설계시 고속처리의 문제점을 해결하기 위해 프로그램 메모리 버스와 데이터 메모리 버스는 하버드 구조로 분리하여 외부 프로그램 메모리로부터 명령어의 페치와 외부 데이터메모리 액세스를 동시에 수행하도록 설계하였다. 또한, 명령어의 디코딩 시간을 줄이기 위해 시퀀스 제어 명령에 적합하도록 마이크로프로세서의 명령어를 구성하였고 이를 32 비트 RISC 구조로 설계하였다. 또한 시퀀스제어에 주로 사용되는 펄스처리명령이나 BCD 연산 등의 명령어를 병렬로 처리함으로써 명령어의 길이와 수행시간을 단축할 수 있는 구조로 프로세서를 설계하였다. 아울러 프로세서의 디버깅을 위해 프로그램 카운터일치, 데이터메모리 일치, 1개씩 명령어 수행 등과 같은 디버깅 기능을 부가하여 프로세서의 검증을 쉽게 하였다. 이를 구현하기 위해 VHDL을 사용하여 FPGA를 설계하였으며 Xilinx사의 240핀 HQFP 형태인 V600EHQ240 디바이스와 Foundation V4.2i 합성툴을 이용하여 시뮬레이션을 성공적으로 수행하였다.

본 논문에서 설계된 프로세서의 우수성을 보이기 위해 비트 처리 명령과 워드처리 명령어가 많이 있어 산업용 제어기기로 널리 사용되고 있는 내부 32비트, 외부 16비트인 Hitachi사의 H8S/2148과 비교하여 비트 명령어에서는 5배, 워드 명령어일 경우는 2배 이상 빠름을 실험을 통해 확인하였고, 특히 BCD 연산전용의 ALU를 설계함으로써 범용의 마이크로프로세서인 H8S/2148보다 약 50배 이상 빠르게 수행되어 본 논문에서 설계된 마이크로프로세서의 우수성을 확인하였다.

참 고 문 헌

[1] F. Bonfatti, G. Gadda, P. Daniela Monari, "Reusable software Design for Programmable Logic controllers," ACM SIGPLANT Notices, Vol.30, No.11, pp. 31~40, 1995.

[2] M. Morris Mano, "Computer System Architecture", 1997.

[3] Koo. KH et al, "Implementation of a RISC microprocessor for programmable logic controllers", Microprocess Microsys, 2000.

[4] K. Koo, W. H. Kown, "Predicting Execution Time of Relay Ladder Logic for Programmable Logic Controllers," in Proceeding of the 1996 IEEE Conference on Emerging Technologies and Factory Automation, Vol.2, pp. 670~676, 1996.

[5] Yoshiyuki Shimokawa, Toshiyuki Matsushita, Hideo Furuno, Yoh Shimanuki, "A High-Performance VLSI Chip of Programmable Controller and Its Language for Instrumentation

and Electric Control", in Proceeding of the IECON'91 International Conference on Industrial Electronics, Control and Instrumentation, Vol.2, pp. 884~889, 1991.

[6] Bruce W. Bomar, "Implementation of Microprogrammed Control in FPGAs", in Proceeding of the IEEE Transactions on Industrial Electronics, Vol.49, NO.2, 2002.

[7] Miyazawa, I., Nagao, T., Fukagawa, M., Itoh, Y., Mizuya, T., Sekiguchi, T., "Implementation of Ladder Diagram for Programmable Controller Using FPGA", in Proceeding of the 1999 7th IEEE International Conference on Emerging Technologies and Factory Automation, Vol.2, pp. 1381~1385, 1999.

[8] 양 오, "FPGA를 이용한 시퀀스 로직 제어용 고속 프로세서 설계", 대한전기학회 논문지 48권 12호, pp. 1554~1563, 1999

[9] N. Aramaki, Y. Shimokawa, S. Kuno, "A New Architecture for High-Performance Programmable Logic Controller," in Proceeding of the 23rd International conference on Industrial Electronics, Control, and Instrumentation, Vol.1, pp. 187~190, 1997.

[10] Xilinx, "VirtexTM-E 1.8V Fidle Programmable Gate Arrays", 2001.

[11] R. Lipsett, C. Schaefer, "VHDL : Hardware Description and Design", KALA, 1991.

[12] HITACHI Semiconductor, "Hitachi Single-Chip Microcomputer H8S/2144 Series, H8S/2148 Series Hardware Manual", 1997.

[13] HITACHI Semiconductor, "Hitachi Single-Chip Microcomputer H8S/2144 Series, H8S/2148 Series Programming Manual", 1997.

[14] Mitsubishi, "Programming Manual : Common Instructions", 1999.

저 자 소 개

梁 吾(正會員) 第39卷 SC編 2號 參照

1983년 2월 : 한양대학교 전기공학과 졸업. 1985년 2월 : 한양대학교 대학원 전기공학과 졸업(석사). 1985년 1월~1997년 8월 : LG산전 연구소 책임연구원. 1997년 2월 : 한양대학교 대학원 전기공학과 졸업(박사). 1997년 9월~현재 : 청주대학교 정보통신공학부 조교수. <주관심분야 : 디지털 논리회로 및 ASIC 설계, DSP 응용 제어>