FPGA를 이용한 시퀀스 로직 제어용 고속 프로세서 설계

論 文
48A-12-11

The Design of High Speed Processor for a Sequence Logic Control using FPGA

梁 吾*
(Oh Yang)

Abstract - This paper presents the design of high speed processor for a sequence logic control using field programmable gate array(FPGA). The sequence logic controller is widely used for automating a variety of industrial plants. The FPGA designed by VHDL consists of program and data memory interface block, input and output block, instruction fetch and decoder block, register and ALU block, program counter block, debug control block respectively. Dedicated clock inputs in the FPGA were used for high speed execution, and also the program memory was separated from the data memory for high speed execution of the sequence instructions at 40 MHz clock. 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 16 bits or 32 bits respectively. And the real time debug operation was implemented for easy debugging the designed processor. This FPGA was synthesized by pASIC 2 SpDE and Synplify-Lite synthesis tool of Quick Logic company.

The final simulation for worst cases was successfully performed under a Verilog HDL simulation environment. And the FPGA programmed for an 84 pin PLCC package was applied to sequence control system with inputs and outputs of 256 points. The designed processor for the sequence logic control was compared with the control system using the DSP(TMS320C32-40MHz) and conventional PLC system. The designed processor for the sequence logic showed good performance.

Key Words : FPGA, sequence logic controller, VHDL, Verilog HDL, DSP, PLC

1. 서 론

반도체 기술의 비약적인 발달과 디지털 논리회로의 고집적화로 마이크로프로세서(micro-processor)를 이용한 컴퓨터 응용기술은 계속해서 발전하고 있다. 또한 마이크로프로세서의 처리능력은 날로 고기능, 고성능, 고속화 되고있으며 또한, 소자(device)의 크기는 점점 작아지고있고 소비전력도 점점 적어지고 있지만, 성능면에서는 고기능, 고속화를 요구하고있으며 가격은 저가일 것 등이 요구되는 추세에 있다.

이와 같은 요구에 따라 공장 자동화, 공작 기기, 간이로봇, 프로세서 콘트롤, 자동차 조립공정 등과 같이 사용자가 쓰기 쉽도록 프로그래밍할 수 있고 신뢰성이 높은 제어 기기를 요구하게 되었으며 이에 부응할 수 있는 제어 기기가 시퀀스 로직 콘트롤러이며 이를 좀더 확장하여 만들게된 것이 프로그램어블 로직 콘트롤러(programmable logic controller)의 출현을 가져왔다[1,2,3]. 그 동안 프로그램어블 로직 콘트롤러의 설계를 위해 쉽게 사용할 수 있고 구현하기가 쉬운 범용의 마이크로프로세서를 채택하여 기능을 구현하였다. 그러나 마이크로 프로세서나 마이크로 콘트롤러에는 원래의 설계목적이 범용이면서 바이트나 워드 단위의 처리에 적합한 구조로 설계되었기 때문에 프로그램어블 로직 콘트롤러와 같은 1비트 단위의 시퀀스 제어기능을 구현하기 위해서는 하나의 비트(boolean)연산을 수행하기 위해 여러 개의 바이트 또는 워드 단위의 명령어가 필요하고 프로그램 메모리의 용량을 늘려야하는 불편함을 가지고 있다[4,5]. 이상과 같이 시퀀스 제어기능을 구현하기 위해 많은 프로그램 용량이 필요하고 거의 실시간에 가까운 기능을 구현하기 위해서는 프로그램 메모리를 적게 차지하며 실시간에 가까운 고속처리가 필수적인 요소로 등장하게되었다. 본 논문에서는 사용자 프로그램의 용량 증가에 따른 속응제어의 문제점을 해결하고자 시퀀스 로직 제어용 프로세서를 설계하여 시퀀스 명령처리에 대한 고속화를 꾀하고자한다. 시퀀스 명령을 처리하는 기존 제품의 경우에는 비트 명령이 지원되는 마이크로 프로세서만을 사용하는 방법 및 범용의 마이크로프로세서와 하드웨어적으로 시퀀스명령어를 처리할 수 있는 프로세서를 ASIC화하여 처리하는 2가지의 방법이 있다. 먼저 전자의 방법에서는 대부분 컴파일 방법을 사용하여 운전지령 바로 직전에 각각의 시퀀스 명령어를 해독하여 어셈블리 언어로 컴파일 하여 스캔(Scan) 처리하는 방법이며 이와 같은 방법의 처리속도는 수 us가 주류를 이루고 있고, 후자의 경우에는 시퀀스 명령처리 전용의 ASIC을 사

* 正 會 員 : 淸州大 電子·情報通信·半導體工學部
專任講師·工博
接受日字 : 1999年 9月 16日
最終完了 : 1999年 11月 26日

용하여 처리속도를 향상시키는 방법으로써 중소형(최대 I/O:512점)의 PLC의 경우 시퀀스 명령어의 처리속도가 국내외적으로 약 200 ns로 발표되고 있다. 시퀀스 로직 콘트롤러의 경우 처리속도는 제품의 경쟁력을 높이는 중요한 요소가 되고 있기 때문에 본 논문에서는 시퀀스 명령어에 대한 고속처리를 위해 적절한 타이밍설계와 시퀀스 명령어를 페치 하면서 데이터메모리를 읽고 쓰는 방법 등을 이용하여 시퀀스 명령어의 처리속도를 100 ns로 함으로써 처리속도의 고속화를 꾀하는데 주안점을 두었다. 또한 시퀀스 전용의 프로세서를 개발한 후 이를 응용하는 과정에서 디버깅을 필요로 하는 MDS를 개발해야하는데 이에 대한 개발이 어렵다는 판단아래 실시간 디버그 기능을 구현하여 쉽게 디버깅할 수 있도록 하였다. 이와 같은 목적을 구현하기 위해 사용자의 프로그램 메모리(program memory)와 데이터 메모리(data memory)를 분리 설계함으로써 고속화를 실현하며 또한, 사용자의 명령어인 인스트럭션을 고속처리에 적합하도록 설계하여 명령어의 디코딩시간을 줄이고 간단한 방법으로 데이터 메모리를 다루게 함으로써 시퀀스 제어시스템의 성능을 높이고자한다. 아울러 시퀀스 명령어인 불(boolean) 연산은 본 논문에서 제안된 시퀀스 로직 제어용 고속 프로세서가 제어를 담당하며, 바이트(byte) 또는 워드(word) 등의 연산기능과 자기진단 및 통신기능은 히다치(HITACHI)사의 16비트 마이크로 콘트롤러인 H8/510을 사용하였고 시스템 클럭으로는 19.6608MHz를 이용하여 전체적인 사용자 프로그램을 처리하였다[6,7]. 즉, 시퀀스 제어 명령은 본 논문에서 설계된 전용의 프로세서가 담당하며 입·출력의 데이터처리와 자기진단기능, 통신기능 등은 범용의 마이크로 콘트롤러가 담당함으로써 전체적인 성능을 올리고자한다. 마지막으로 본 논문에서 설계된 시퀀스 처리 전용의 프로세서에 대한 초고속 처리능력을 보이기 위해 디지털 신호처리 전용의 프로세서인 DSP(TMS320C32-40MHz)[8] 및 기존의 PLC 시스템과 성능을 비교 평가하여 본 논문에서 구현한 시퀀스 로직 제어용 프로세서가 우수함을 실험을 통해 입증하고자한다. 전체적인 논문의 구성은 2장에서 시퀀스로직 콘트롤러의 설계사양과 전제조건을 설명하고, 시퀀스 로직 콘트롤러의 명령어 코드를 구성하였다. 또한, 시퀀스 로직 제어용 고속 프로세서의 시퀀스 명령에 대한 동작방법에 대해서 고찰하고 각각의 명령에 대한 타이밍을 설명한다. 3장에서는 설계된 프로세서의 합성 및 시뮬레이션을 다루며, 4장에서는 설계된 프로세서를 직접 FPGA로 구현한 후 시퀀스 로직 제어 시스템에 적용하여, DSP를 이용한 제어시스템 및 기존의 PLC로 구현한 것 등과 비교 평가하여 설계된 프로세서의 우수성을 보이며, 마지막으로 5장에서 본 논문의 결론을 맺는다.

2. 시퀀스 로직 제어용 고속 프로세서 설계

2.1 고속 프로세서의 설계사양

먼저 FPGA를 이용한 시퀀스 로직 제어용 고속 프로세서를 설계하기 위한 설계사양을 표 1에 나타내었다. 표 1에서 기본적인 전제조건은 고속처리가 가능하면서도 경제적이 측면이 요구되고 있다. 이를 위해서는 프로그램 메모리와 데이터 메모리는 8 비트일 것과 32 키로 바이트(kbytes)의 어드레스 공간으로 각각 구성되며 시퀀스 처리에서 70 ~ 80%를 차지하는 외부 데이터 메모리의 읽기와 프로세서 내부 비트 처리를 위한 처리속도가 100 ns로 처리 할 수 있을 것과 사용자 프로그램의 용량은 16 키로 스텝일 것, 프로세서가 처리해야할 접점용량은 6,656점 일 것과 실시간 디버깅 기능을 내장하여 1 스텝운전, 일정번지 브레이크운전, 일정한 어드레스를 읽거나 쓸 때 정지하는 방법(value break run) 등이 중요 전제조건이 되고있다.

표 1 시퀀스 로직 제어용 고속 프로세서의 설계사양
Table 1 Specification of high speed processor for sequence logic control

프로세서 비트 처리용 명령어수		21 종류
프로세서 내부 데이터 버스		16 비트
프로세서 내부 어드레스 버스		18 비트
프로세서 외부 프로그램 메모리 용량		8비트 × 32 kbytes
프로세서 외부 데이터 메모리 용량		8비트 × 32 kbytes
프로그램 메모리 (사용자 프로그램 스텝)		16,384 스텝 (16 ksteps)
프로세서의 비트 처리 접점 용량		6,656 점
처리 속도	외부 데이터 메모리 읽기, 내부 비트 처리	100 ns
	외부 데이터 메모리 쓰기의 처리속도	200 ns
	외부 데이터 메모리의 펄스 처리	800 ns
실시간 디버그(Debug) 기능		1 스텝 운전 모드
		일정번지 브레이크 운전 모드
		Value 브레이크 운전모드
다른 MPU와의 인터페이스 기능		HOLD와 HOLD ACK 기능

2.2 시퀀스 로직 콘트롤러의 명령어 코드

시퀀스 로직 콘트롤러에 대한 명령어의 처리속도를 높이고 명령어의 해독시간을 줄이기 위해서는 일정한 규칙의 명령어 코드가 있어야하는데 본 논문에서는 표 2와 같이 비트 15 ~ 비트 13까지는 명령어의 종류를 나타내며 비트 12 ~ 비트 10까지는 비트 접점의 오퍼랜드를 구분하도록 하였으며 비트 2 ~ 비트 0은 각각 오퍼랜드의 바이트 어드레스에 대한 비트 번호를 나타내도록 하였다. 아울러 명령어의 종류를 마스크한 후 전체의 비트를 우측으로 3번 쉬프트 함으로써 데이터 메모리의 어드레스를 만들도록 하여 명령어에 대한 디코딩 시간을 최대한 줄이도록 하였다. 표 2에서 오퍼랜드 M은 내부메모리를 나타내며 Y와 X는 외부 출력접점과 입력접점을 각각 나타낸다. 또한, B는 통신 등의 데이터링크 접점을, 그리고 L접점은 정전시에도 정전이전의 접점데이터의 상태를 유지하는 불휘발성 접점을 나타낸다. 아울러 T와 C접점은 각각 타이머와 카운터의 접점을 나타낸다. 이와 같은 접점 수는 중형의 시퀀스 로직 콘트롤러에

적합한 용량으로써 프로그램 스텝이 16K 스텝(16,384 steps)을 제어할 수 있는 접점용량이다. 이상과 같은 오퍼랜드중 시스템 정보용 F 및 X 오퍼랜드 등은 직접 셋(set)이나 리셋(reset), 펄스처리를 할 수 없는 명령어이므로 명령어 코드에서 빠져있다. 또한, 시퀀스 제어에서 자주 사용되는 펄스처리용 명령어는 주로 원하는 시퀀스 동작에서 한번만 수행하는 동작이나 어떠한 이벤트가 발생할 때마다 제어동작을 하는 명령어로써 상승에지용 펄스 명령어인 PLS와 하강에지용 펄스명령어인 PLF가 각각 있다. 이러한 PLS와 PLF는 비트 종류 구분을 위해 표 2에서 32비트로 구성하였다. MC와 MCR은 마스터 콘트롤 명령어로써 7개까지의 정수오퍼랜드를 가지며 이에 대한 명령어 코드를 표 3에 나타내었다. 이와 같은 명령어는 시퀀스 명령에 대한 마스터를 콘트롤할 수 있는 명령어로써 MC 명령어는 임의 입력접점이 오프 되면 이하의 명령어를 무처리(NOP)하고, 다시 MCR 명령어를 수행함으로써 마스터 콘트롤을 해제하게된다.

이러한 명령어는 시퀀스제어를 행할 때 효과적으로 시퀀스 명령어를 수행하여 스캔타임을 줄일 수 있으며 범용의 마이크로프로세서로 이러한 명령어를 구현하기 위해서는 여러 개의 바이트 또는 워드 단위의 명령어를 조합해서 사용해야한다. 표 4는 오퍼랜드가 없는 명령에 대한 코드이며 외부로부터 데이터 메모리를 엑세스하지 않고 프로세서 내부의 레지스터나 비트 연산 ALU 등과의 연산처리 동작을 하는 명령어로써 NOP은 무처리 명령이고 NOT은 현재의 비트 연산 결과를 반전하며, ANB와 ORB는 논리 블러간 AND와 OR동작을 하는 명령이고 MPP, MPS, MRD는 각각 메모리 팝, 메모리 푸시, 메모리 리드 등의 동작을 나타내는 명령어이다.

표 3 정수 오퍼랜드를 갖는 시퀀스 명령어 코드표
Table 3 The code table of sequence instructions with integer operand

MC	0000 1111 0110 0xxx
MCR	0000 1111 0111 0xxx

표 4 오퍼랜드가 없는 시퀀스 명령어 코드표
Table 4 The code table of sequence instructions without operand

NOP	0000 0000 1111 1111
NOT	0000 1111 0000 0000
ANB	0000 1111 0001 0000
ORB	0000 1111 0010 0000
MPP	0000 1111 0011 0000
MPS	0000 1111 0100 0000
MRD	0000 1111 0101 0000

표 2 오퍼랜드가 있는 시퀀스 명령어 코드표
Table 2 The code table of sequence instructions with operands

명령어 \ 오퍼랜드	M (2048 점)	Y (1024 점)	B (1024 점)	L (512 점)	F (512 점)	X (1024 점)	T (256 점)	C (256 점)
LD	0100 0xxx xxxx xxxx	0100 10xx xxxx xxxx	0100 11xx xxxx xxxx	0101 000x xxxx xxxx	0101 001x xxxx xxxx	0101 01xx xxxx xxxx	0101 1000 xxxx xxxx	0101 1001 xxxx xxxx
LDI	0110 0xxx xxxx xxxx	0110 10xx xxxx xxxx	0110 11xx xxxx xxxx	0111 000x xxxx xxxx	0111 001x xxxx xxxx	0111 01xx xxxx xxxx	0111 1000 xxxx xxxx	0111 1001 xxxx xxxx
AND	1000 0xxx xxxx xxxx	1000 10xx xxxx xxxx	1000 11xx xxxx xxxx	1001 000x xxxx xxxx	1001 001x xxxx xxxx	1001 01xx xxxx xxxx	1001 1000 xxxx xxxx	1001 1001 xxxx xxxx
ANDI	1010 0xxx xxxx xxxx	1010 10xx xxxx xxxx	1010 11xx xxxx xxxx	1011 000x xxxx xxxx	1011 001x xxxx xxxx	1011 01xx xxxx xxxx	1011 1000 xxxx xxxx	1011 1001 xxxx xxxx
OR	1100 0xxx xxxx xxxx	1100 10xx xxxx xxxx	1100 11xx xxxx xxxx	1101 000x xxxx xxxx	1101 001x xxxx xxxx	1101 01xx xxxx xxxx	1101 1000 xxxx xxxx	1101 1001 xxxx xxxx
ORI	1110 0xxx xxxx xxxx	1110 10xx xxxx xxxx	1110 11xx xxxx xxxx	1111 000x xxxx xxxx	1111 001x xxxx xxxx	1111 01xx xxxx xxxx	1111 1000 xxxx xxxx	1111 1001 xxxx xxxx
OUT	0001 0xxx xxxx xxxx	0001 10xx xxxx xxxx	0001 11xx xxxx xxxx	0000 001x xxxx xxxx			0000 1000 xxxx xxxx	0000 1100 xxxx xxxx
SET	0010 0xxx xxxx xxxx	0010 10xx xxxx xxxx	0010 11xx xxxx xxxx	0000 010x xxxx xxxx				
RST	0011 0xxx xxxx xxxx	0011 10xx xxxx xxxx	0011 11xx xxxx xxxx	0000 011x xxxx xxxx			0000 1011 xxxx xxxx	
PLS	0101 1010 0000 0xxx 1011 1010 xxxx xxxx	0101 1010 0000 10xx 1011 1010 xxxx xxxx	0101 1010 0000 11xx 1011 1010 xxxx xxxx	0101 1010 0001 000x 1011 1010 xxxx xxxx				
PLF	0101 1011 0000 0xxx 1011 1010 xxxx xxxx	0101 1011 0000 10xx 1011 1010 xxxx xxxx	0101 1011 0000 11xx 1011 1010 xxxx xxxx	0101 1011 0001 000x 1011 1010 xxxx xxxx				

2.3 시퀀스 로직 제어용 프로세서의 내부 구성

시퀀스 로직 제어용 프로세서에 대한 고속 처리를 위해 프로그램 메모리와 데이터 메모리의 버스를 분리하여 설계하였으며, 고속 프로세서를 설계하기 위한 내부의 구성도는 그림 1과 같다.

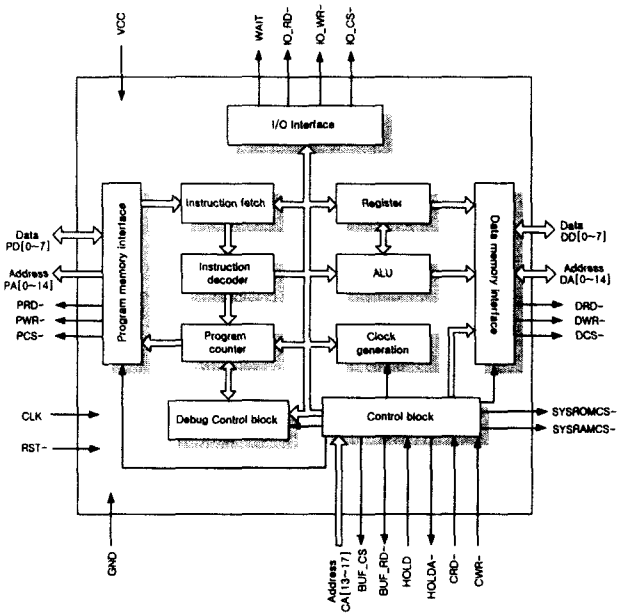


그림 1 시퀀스 로직 제어용 프로세서의 내부 구성도

Fig. 1 Block diagram of the processor for sequence logic control

그림 1에서 콘트롤 블록(control block)에서는 범용 프로세서의 어드레스 버스(address bus)인 CA[13~17]가 연결되며 이 어드레스는 시퀀스 제어용 프로세서 내부에서 선택신호를 만드는 어드레스로 사용된다. 아울러 시퀀스 로직 제어용 프로세서를 제어하는 HOLD와 HOLDA-가 연결된다. 만약 외부로부터 HOLD가 low가 되면 제어용 프로세서가 처리할 수 있는 명령어의 경우에는 HOLDA-가 low로 되며, 만약에 수행 불가능한 명령어일 경우에는 high로되어 범용 프로세서에 제어권을 넘겨주게 된다. 그림 1에서 설계된 프로세서는 총 15비트로 구성된 프로그램 카운터를 기본으로 하여 동작하며 HOLD가 high가 되면 정지하는 구조로 되어 있다. 프로그램 카운터, 프로그램 메모리, 데이터 메모리의 공간을 15비트로 한 것은 경제성을 고려한 32KB의 SRAM에 적합하도록 설계하였다. 또한, CRD-와 CWR-는 각각 범용프로세서의 읽기 신호(RD-)와 쓰기 신호(WR-)가 연결되어 시퀀스 제어용 프로세서의 내부 레지스터를 읽거나 쓸 경우 사용되는 제어신호이다. 디버그 콘트롤 블록은 시퀀스 제어시스템을 디버깅하기 위한 블록으로 디버그 콘트롤 레지스터의 설정에 따라 한 명령어씩 수행하는 싱글 스텝런(single step run), 프로그램 카운터의 내용과 프로그램을 정지하고자하는 어드레스가 일치했을 경우 정지하는 방법(pc break run), 일정한 어드레스를 읽거나 쓸 때 정지하는 방법(value break run) 등으로 설정할 수 있다. 이와 같은 디버그 기능은 실시간 디버그기능을 구현하므로 실제 상황과 동일한 방법으로 실현할 수 있는 장점을 가지고 있다. 클럭 발생부는 외부로부터 40MHz 클럭(CLK)을 입력받아 그림 2와 같이 4개의 상태(Q1, Q2, Q3, Q4)로 분주되며 프로세서 내부처리 명령어와 외부 메모리에 대한 입력명령의 경우에는 T1 ~ T2의 사이클로 분주되고 외부 데이터 메모리의 읽기와 쓰기 동작을 하는 명령어의 경우에는 T1 ~ T4로 각각 분주되며, 외부메모리의 펄스처리 명령어 그룹의 경우에는 T1 ~ T8로 각각 분주된다.

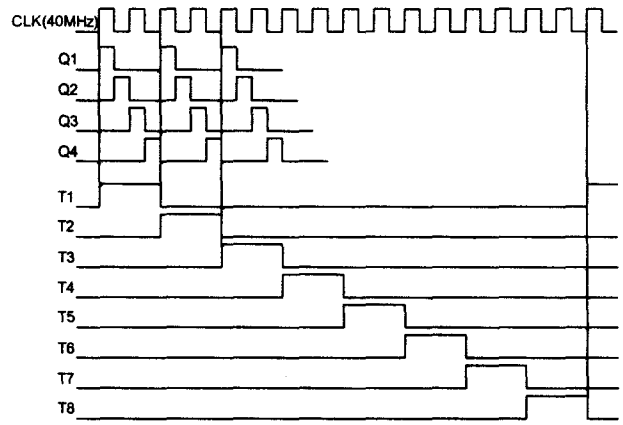


그림 2 클럭 발생부의 타임차트

Fig. 2 Time chart of clock generation block

프로그램 메모리의 외부 인터페이스부는 15비트의 어드레스 버스(address bus)와 8비트의 데이터버스로 구성된다. 여기서 데이터 버스는 메모리의 수를 줄이고 경제성을 고려하기 위해 8비트로 하였으며, 선택신호(chip select)와 읽기/쓰기 제어신호(read/write control signal)가 연결되어 있다. 또한, 인스트럭션 페치부에서는 그림 3과 같이 프로그램 메모리로부터 명령어를 페치한 정보를 일시 저장하기 위한 것으로써 8비트의 Pre_IRH라는 프리페치 인스트럭션 레지스터가 있고 이 레지스터는 T1, T3, T5, T7의 Q4 클럭에서 프로그램 메모리로부터 읽은 데이터를 래치 하는 구조로 되어 있다. 이러한 메모리의 구조는 고속 수행을 위한 구조로써 데이터 메모리를 처리하는 도중에 미리 프로그램 메모리로부터 데이터를 래치하며 T1의 상승에지(rising edge)에서 16비트의 인스트럭션 레지스터에 래치하고 바로 이어 디코더 블록에서 각각의 명령어에 대한 분석을 수행한다.

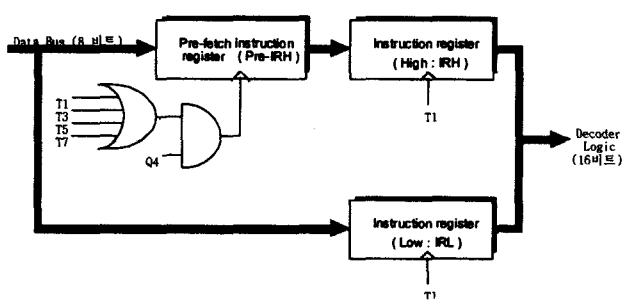


그림 3 인스트럭션 페치 레지스터의 내부 구성도

Fig. 3 Block diagram of instruction fetch registers

2.4 내부처리 명령어의 기능 및 처리방법

외부 메모리를 엑세스하지 않는 내부명령어는 아무런 연산을 하지 않는 NOP(No operation) 명령과 현재 연산된 결과를 반전하는 NOT 명령과 마스터 콘트롤(master control)을 시작하는 MC, 마스터 콘트롤의 끝을 나타내는 MCR로 분류되고, 현재의 시퀀스 연산된 결과를 프로세서 내부의 래지스터에 저장하기 위해 푸시동작(push operation)을 하는 MPS(memory push), 내부의 레지스터에 저장된 연산결과를 읽는 MRD(memory read), 내부의 레지스터에 저장된 결과를 읽고 또한 꺼내는 동작(pop operation)을 하는 MPP(memory pop)로 구성된다. 아울러 논리 블럭간 AND 동작을 하는 ANB 명령과 논리 블록간 OR 동작을 하는 ORB로 구성되며 시퀀스 명령의 종료를 나타내는 END 명령이 있다. 이상과 같은 내부처리 명령어에 대한 동작은 먼저 그림 2와 그림 3에 있는 T1의 상승에지에서는 Pre_IRH에 래치된 8비트의 데이터가 IRH에 래치되며 또한, 프로그램 메모리로부터 흘러나온 8비트의 데이터는 IRL에 각각 래치된다. 아울러 프로그램 카운터를 +1 증가시키며 이 프로그램 카운터의 내용은 프로그램 메모리의 어드레스로 나가며 이때 흘러나온 데이터는 T1의 Q4 상승에지에서 Pre_IRH에 래치시킨 후에 프로그램 카운터의 내용을 하나 증가시킨다. 마지막으로 T2의 Q4 상승에지에서 연산 처리된 결과를 해당 레지스터에 래치한다.

2.5 외부 메모리 읽기 명령어의 기능 및 처리방법

8비트의 외부 데이터 메모리로부터 읽혀진 데이터를 이용하여 LD, AND, OR 동작을 수행하여 내부의 ARG 레지스터에 쓰기 동작을 하는 명령어로써 부정의 동작을 하는 "I"와 결합하는 LDI, ANDI, ORI 명령이 있고 이에 대한 명령어의 래치동작은 내부의 명령어 그룹과 동일하다. 또한, T1의 Q2 상승에지에서 데이터 메모리의 읽기 신호(DRD-)가 low로 되며 이때 흘러나온 데이터는 T2의 Q3 상승에지에서 데이터를 래치한 후 DRD-신호를 high로 만든다. 마지막으로 T2의 Q4 상승에지에서 ALU를 이용한 연산 처리된 결과를 해당 레지스터에 쓰기 동작을 한다. 아울러 마스터 콘트롤 동작시 해당되는 접점이 ON되더라도 마스터 콘트롤 레지스터가 하나라도 OFF되어있으면 LD, LDI의 비트 연산결과는 무조건 "0"이 된다.

2.6 외부 메모리 읽기와 쓰기 명령어의 기능 및 처리방법

하나의 명령어로 외부 데이터 메모리로부터 8비트 데이터를 읽고 또한 ALU를 통한 연산 결과에 따라 쓰기 동작하는 명령어로써 비트 어큐뮬레이터인 ARG의 상태에 따라 해당되는 출력접점을 "0" 또는 "1"로 쓰기 동작을 한다. 이러한 명령은 데이터 메모리의 읽기와 쓰기 동작이 이루어지므로 총 4개의 머신 사이클인 T1 ~ T4까지의 사이클로 이루어지므로 200ns의 수행시간이 요구된다. OUT 명령어는 현재의 연산결과가 1이면 해당 출력접점을 ON하며 반대로 연산결과가 0이면 해당 출력접점을 OFF하는 동작을 수행한다. 또한 SET 명령어는 현재의 연산 결과가 1일 때만 해당 출력접점을 1로 만들며, 반대로 RST 명령어는 현재의 연산결과가 1일 경우 해당 출력접점을 0으로 만든다. 내부동작은 T1의 Q2 상승에지에서 데이터 메모리의 읽기 신호(DRD-)가 low로 되며 이때 흘러나온 데이터는 T2의 Q3의 상승에지에서 읽은 데이터를 래치한 후 DRD-신호를 high로 만든다. 마지막으로 T3의 상승에지에서 데이터 메모리의 쓰기 신호가 low로 되고 연산된 결과를 데이터 버스에 내보낸 후 T3의 Q4의 상승에지에서 쓰기 동작이 이루어진다.

2.7 펄스처리 명령어의 기능 및 처리방법

펄스처리 명령어에는 상승에지 펄스 출력과 하강에지 펄스출력이 있으며 상승에지 펄스출력인 PLS는 입력조건이 OFF에서 ON될 때 단지 1 스캔(scan)동안만 출력이 ON되는 명령어이며, 하강에지 펄스출력인 PLF는 입력조건이 ON에서 OFF될 때 단지 1 스캔(scan)동안만 출력이 ON되는 명령어이다. 펄스처리 명령어 대한 내부 동작은 총 8개의 상태로 구분되고 T1의 상승에지에서는 Pre_IRH에 래치된 8비트의 데이터가 IRH에 래치되며 또한, 프로그램 메모리로부터 흘러나온 8비트의 데이터는 IRL에 각각 래치된다. 아울러 프로그램 카운터를 +1 증가시키며 이 프로그램 카운터의 내용은 프로그램 메모리의 어드레스로 나가고, 또한 T1의 Q3 상승에지에서 데이터 메모리의 펄스처리를 위한 읽기 신호(DRD-)가 low로 되며 이때 흘러나온 데이터는 T2의 Q3 상승에지에서 읽은 데이터를 래치하며 현재와 이전의 상태에 대한 펄스를 체크하여 T3의 Q4에서 쓰기 신호인 DWR-을 이용하여 쓰기 동작을 수행한다. 아울러 T7의 Q3에서 비트 영역의 데이터를 읽고 이에 대한 연산한 후 T8의 Q4에서 쓰기 동작을 한다.

3. 합성 및 시뮬레이션

본 논문에서 제안된 각각의 모듈을 톱다운(top-down) 방식으로 프로그램 메모리 및 데이터 메모리 인터페이스부, 인스트럭션 페치부 및 디코더부, I/O 인터페이스부, 비트 ALU 및 래지스터부, 콘트롤 블럭등 각각의 모듈을 설계하여 VHDL로 기술하였다[9]. 이와 같이 설계 및 기술된 하드웨어 표현언어(VHDL)를 퀵로직(Quick Logic)사에서 제공되는 Synplify-Lite의 로직합성 툴(synthesis tool)을 이용하여 로직을 합성하였으며 핀의 형태는 84핀 QFP 형태이다. 또한, 2007 시리즈중 최고속도를 갖고 7,000게이트(gates)에 해당되는 QL2007-2PF84C를 사용하였다[10,11]. 이때 배치(place) 및 배선(route)의 과정을 거쳐 FPGA의 사용율이 480개의 셀중 366개를 사용함으로써 76.3%가 됨을 확인하였다. 아울러 테스트 벡터(test vector)를 원활히 작성하고 시스템 레벨(system level)상에서 시뮬레이션을 하기 위해 Verilog HDL[12]을 사용하여 테스트 벡터를 작성한 후 포스트 시뮬레이션(post simulation)을 하였다. 그림 4는 배치(place) 및 배선(route)을 마친 후 포스트 시뮬레이션에 대한 결과로써 외부 데이터 메모리의 액세스 없이 프로세서 내부에서만 처리되는 내부처리 명령에 대한 시뮬레이션 결과를 나타내고 있다. 이 그림에서 HOLD_ACK가 low가 되는 것은 현재 프로세서가 명령어를 처리하고있음을 나타내고있고, 시스템 클럭(CLK_40M)은 40MHz가 입력되며 처리되는 명령어는 외부의 데이터 메모리를 액세스하지 않는 MC K0 명령어이고 인스트럭션 레지스터가 "0x0f50"으로써 머신 사이클은 50ns 폭을 갖는 2개의 신호인 T1 ~ T2로 수행이 완료되고 처리

속도인 Tdelta는 100ns임을 알 수 있다. 아울러 프로세서 내부 처리 명령어이므로 데이터 메모리의 읽기(DMEM_RD)와 쓰기(DMEM_WR)신호는 high임을 알 수 있다.

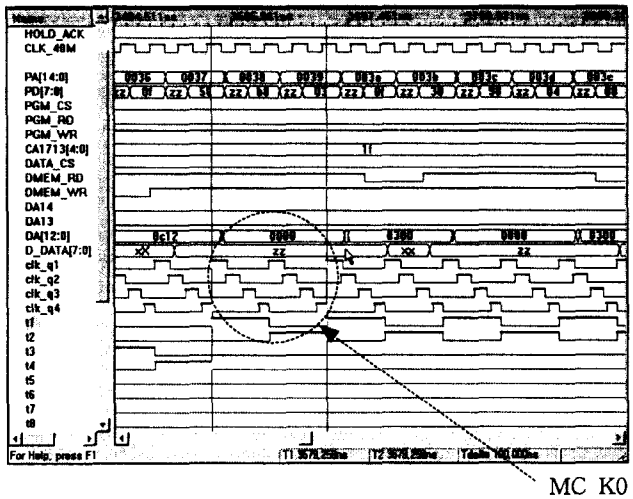


그림 4 내부처리 명령에 대한 시뮬레이션 결과
Fig. 4 Simulation result of internal instruction

그림 5는 외부 데이터 메모리를 읽기만 하는 명령과 읽기와 쓰기 동작을 하는 명령어에 대한 처리를 나타내는 시뮬레이션 결과이다.

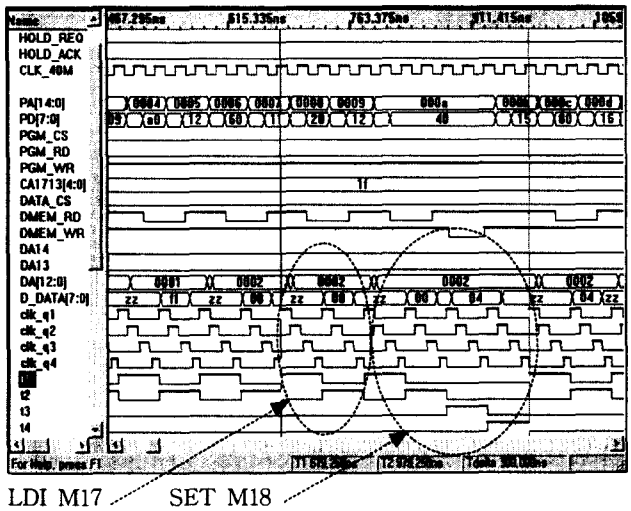


그림 5 외부 메모리 입력과 쓰기 명령어에 대한 시뮬레이션 결과
Fig. 5 Simulation result of read and write instruction from external memory

그림 5로부터 먼저 읽기 동작에 대한 인스트럭션 레지스터는 "0x6011"로써 LDI M17의 동작을 의미하고 이를 해석하면 비트의 번호는 1번이며 데이터 메모리의 어드레스(DA)는 2번이며 읽혀진 데이터는 "00"임을 알 수 있다. 이때 데이터 메모리의 상태를 읽기 위해 DMEM_RD 신호가 출력되고 명령어의 수행 사이클은 2 사이클에 수행된다. 아울러 읽기와 쓰기 동작에 대한 시뮬레이션으로 수행 코드는 "0x2012"로써 SET M18의 동작을 의미하고 현재의 비트 연산결과가 1이므로 데이터 메모리의 2번 어드레스의 3번째 비트에 1인 "04"를 쓰고 있음을 알 수 있다. 아울러 명령어의 수행 사이클은 T1 ~ T4로 4사이클에 수행됨을 알 수 있다. 수행시간은 데이터 메모리의 읽기 명령의 경우에는 Tdelta는 100ns이고 읽기와 쓰기 동작을 하는 명령어인 경우에 Tdelta는 200ns임을 알 수 있다. 그림 6은 펄스처리 명령어에 대한 시뮬레이션 결과를 나타내고 있다. 현재 수행중인 명령어는 PLS M17이며 이에 대한 OP코드는 32비트의 길이를 가지며 "0x5a00ba11"로 T1 ~ T4 구간에서는 펄스처리 알고리즘에 의해 DA13과 DA14가 high가되어 펄스의 상위 어드레스를 형성하고 현재의 프로그램 카운터의 최하위 3비트는 펄스메모리 비트의 번호를 의미하고 프로그램 카운터를 8로 나눈 값("03")과 상위의 어드레스인 DA13과 DA14를 결합하여 펄스의 어드레스를 형성한다. 또한, T5 ~ T8의 구간에서는 이전펄스의 상태와 현재의 펄스상태입력을 이용하여 만약 상승에지가 검출되면 M17을 1로 쓰고 반대로 상승에지의 상태가 검출되지 않으면 0으로 쓰기 동작을 한다. 이와 같은 펄스처리 시간은 총 8사이클로써 처리속도가 400ns임을 그림 6으로부터 알 수 있다.

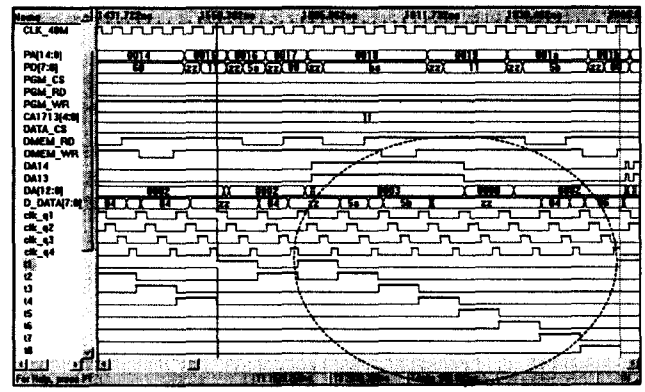


그림 6 펄스처리(PLS M17) 명령어에 대한 시뮬레이션 결과
Fig. 6 Simulation result of pulse instruction(PLS M17)

4. 실험 및 결과

설계된 FPGA에 대한 타당성을 검토하며 또한, 이 FPGA를 이용한 시퀀스로직 콘트롤러 시스템에 적용하여 FPGA를 이용한 시퀀스로직 제어용 프로세서의 성능을 실험을 통해 평가하고자한다. 이를 위해 전체 제어 시스템의 구조를 그림 7과 같이 구성하였다. 그림 7에서 H8/510은 HITACHI사의 범용 마이크로 콘트롤러이며 외부 클럭은 시리얼 통신시 19,200bps의 전송속도에 적합한 19.6608MHz를 사용하였고 이에 대한 시스템 ROM은 27C256을, 시스템 RAM은 UM61256FK-15를 각각 사용하였다. 아울러 시퀀스 제어용 프로세서의 프로그램 메모리와 데이터 메모리는 액세스타임이 15ns인 RAM(UM61256FK-15) 2개를 사용하였으며 FPGA가 동작 중에도 시리얼 통신이나 자기진단을 위해 74HC245와 74HC541 버퍼를 사용하였다. 또한, 시퀀스제어의 1 스캔이 완료되면 입력 카드로부터는 입력을 읽고 출력 카드로 연산된 데이터를 출력할 수 있는 구조로 입출력을 각각 설계하였다. 즉, I/O 인터페이스부를 이용하여 디지털

입력은 128점의 DC24V입력, 디지털 출력은 128점의 릴레이 출력을 각각 설계하였다. 또한 정상전압의 경우에는 5V를 출력하며 정전시에는 2V 이상을 프로그램 메모리와 데이터 메모리에 전압을 공급하여 데이터를 유지시키는 L영역을 위해 3.6V용 바테리를 이용하였다.

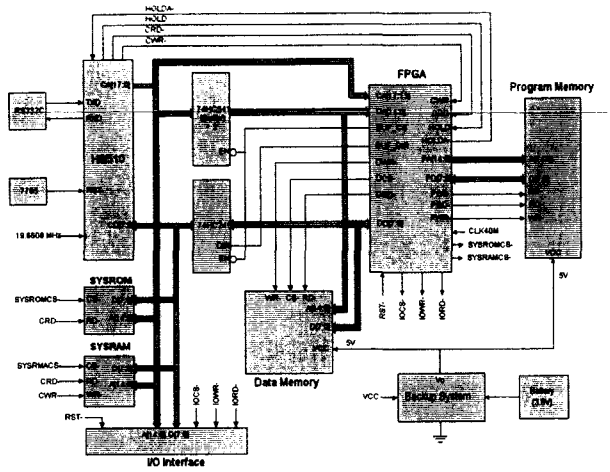
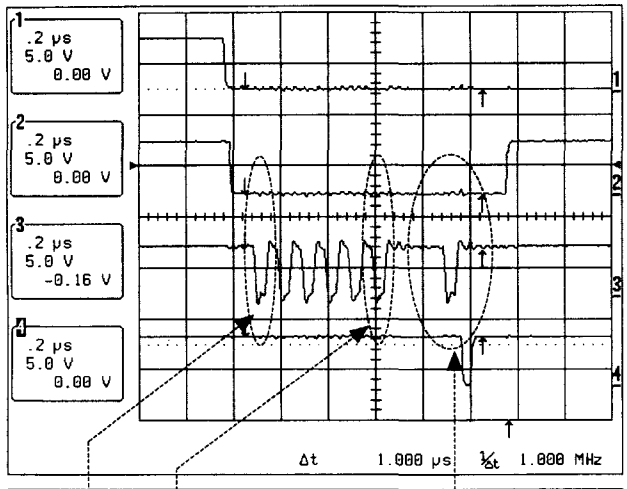


그림 7 프로그램어블 로직 콘트롤 시스템의 구성
Fig. 7 Configuration of sequence logic control system

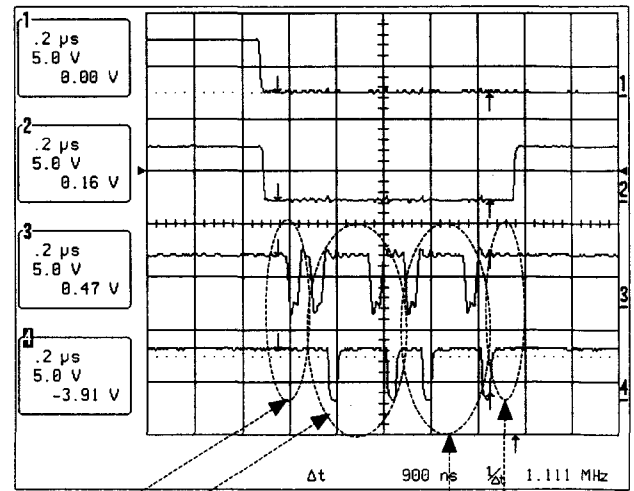


LD	M1	;100ns
ANDI	M2	;100ns
LDI	M3	;100ns
AND	T4	;100ns
LD	L20	;100ns
AND	C12	;100ns
ORB		;100ns
ANB		;100ns
OUT	P30	;200ns
END		;100ns

그림 8 내부처리와 입력 및 쓰기 명령에 대한 실험결과
Fig. 8 Experimental result of internal, read and write instructions

그림 7의 시스템 구성으로부터 실험한 결과 파형은 그림 8과 같다. 이 그림에서 1번 채널은 HOLD를 나타내고 2번 채널은 HOLDA-를 나타내며 채널 3과 4는 DRD-와 DWR-를 각각 나타낸다. 이에 대한 시퀀스로직의 명령은 그림 8의 오른쪽에 나타내었다. 입력명령, 입력반전명령, 출력명령을 조합한 경우로써 각각의 명령에 대한 수행시간은 외부 데이터 메모리로부터 읽기 동작을 하는 LD, ANDI, LDI, AND, ANDI는 100ns이며, 또한 시퀀스 전용의 프로세서 내부의 동작을 하는 ORB, ANB는 100ns가 소요된다. 아울러 출력명령인 OUT는 외부 메모리로부터 읽기와 쓰기 동작을 수행하므로 200ns가 소요됨을 실험 결과로부터 알 수 있었다. 또한, 시퀀스명령의 끝을 나타내는 END 명령은 OP 코드를 디코드하고 나서 프로세서를 정지시키는 명령으로 100ns가 소요된다. 이때 HOLDA-신호를 더 이상 처리할 수 없는 경우를 범용의 마이크로 콘트롤러인 H8/510에 알려주는 신호이고 이 신호가 high가되어 다른 프로세서에 액세스권을 넘겨줌으로써 모든 시퀀스 동작은 완료된다.

그림 9는 입력명령과 상승에지 펄스명령 및 하강에지 펄스명령에 대한 실험 결과 파형이다. 그림 9의 파형에서 1번 채널은 HOLD를 나타내고 2번 채널은 HOLDA-를 나타내며 채널 3과 4는 외부 데이터 메모리의 읽기 신호인 DRD-와 쓰기 신호인 DWR-를 각각 나타낸다. 이에 대한 시퀀스로직의 명령은 그림 9의 오른쪽에 나타내었다. 입력명령인 LD M1에 대한 상승에지 펄스인 PLS M2는 명령어의 처리시간이 400ns가 되고 아울러 하강에지 펄스명령인 PLF B12 역시 400ns임을 결과 파형으로부터 알 수 있다. 펄스의 명령에서는 앞에 나오는 읽기 신호와 쓰기 신호는 펄스영역을 읽고 쓰는 영역이며 뒤에 나오는 DRD-, DWR-는 접점영역 즉, M2와 B12에 해당되는 접점의 읽기와 쓰기를 각각 나타낸다.



LD	M1	;100ns
PLS	M2	;400ns
PLF	B12	;400ns
END		;100ns

그림 9 펄스 명령에 대한 실험결과
Fig. 9 Experimental result of pulse instructions

그림 10은 시퀀스 로직 제어용 프로세서에서 구현한 모든 명령어를 조합하여 간단한 시퀀스 제어시스템을 구성하고 이에 대한 실험결과 파형을 보이고 있다. 그림 10에서 채널 1은 HOLD신호를 나타내며 채널 2는 HOLDA-, 채널 3은 DRD-, 채널 4는 DWR-를 각각 나나내고 있다.

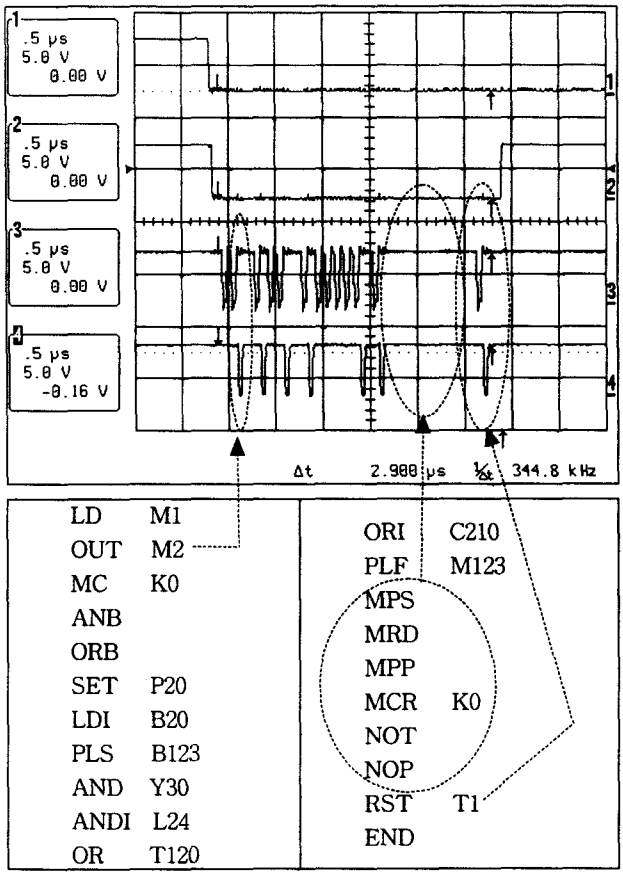


그림 10 모든 시퀀스 명령에 대한 실험결과
Fig. 10 Experimental result of all sequence instructions

본 실험에서 사용한 명령어는 모두 21 종류의 명령어가 사용되며 시퀀스 제어명령의 마지막에서는 "END"명령으로 시퀀스 명령이 종료되고 다음 스캔(scan)의 시작인 LD M1이 시작되어 계속해서 시퀀스 프로그램이 반복해서 수행되는 구조를 가지고 있다. 기본적으로 시퀀스 제어용 프로세서 내부의 명령인 MC, MCR, ANB, ORB, MPS, MRD, MPP, NOT, NOP, END 등은 100ns로 처리되고 또한, 외부 데이터 메모리의 읽기 명령인 LD, LDI, OR, ORI, AND, ANDI의 경우에도 100ns에 처리된다. 그러나 외부 데이터 메모리에 읽고 쓰는 출력명령인 OUT, SET, RST의 경우에는 200ns로 처리되고 또한 펄스처리명령인 PLS, PLF의 경우에는 400ns로 처리된다. 이상과 같은 전체의 시퀀스 명령 조합을 그림 10의 파형 우측에 나타내었다. 이러한 시퀀스 명령어에 대한 동작을 살펴보면 프로세서 내부처리 명령어인 경우에는 외부에 읽기 또는 쓰기 신호가 발생되지 않고, 읽기 명령어는 읽기 신호만을 출력하며, 출력명령어의 경우에는 읽기와 쓰기 신호가 출력됨을 알 수 있다.

아울러 펄스 명령어의 경우에는 처음에 나오는 읽기 신호는 펄스처리영역을 읽기 위한 신호가 출력되고 현재의 비트 어큐뮬레이터의 상태를 펄스영역에 쓰기 동작을 수행한다. 아울러 2번째 나오는 읽기 신호는 비트 접점의 내용을 읽기 위한 신호이고 이를 이용하여 상승에지에 대한 검출 알고리즘을 수행하여 해당 비트 접점에 쓰기 동작을 수행한다

이제 본 논문에서 설계된 시퀀스 제어 로직용 프로세서의 우수성을 보이기 위해 신호처리 전용의 프로세서인 DSP를 이용한 경우와 기존의 PLC 시스템과 성능을 비교 평가하고자 한다. 이를 위해 DSP는 TMS320C32-40를 이용하며 고속처리를 위해 프로그램 메모리와 데이터 메모리를 초고속 RAM인 UM61256FK-15(액세스 타임 : 15ns)를 4개 사용하여 32비트로 설계하고, 외부 클럭을 40MHz로 입력하여 설계된 시퀀스 전용 프로세서와 클럭을 동일하게 하였다. 아울러 비트 연산결과를 내부레지스터 R0에 할당하고 마스터 콘트롤 레지스터를 R1에 그리고, 메모리 푸시, 리드, 팝 동작을 위한 레지스터를 R2에 각각 할당하여 고속화를 꾀하였다. 이와 같은 시스템을 설계한 후 시리얼(serial) 통신을 이용하여 프로그램 메모리에 원하는 사용자의 프로그램인 시퀀스 제어용 프로그램을 다운로드(down load)한 후 RUN 운전 지령을 하여 각각의 명령어에 대한 처리속도를 오실로스코프로 계측하여 표 5와 같은 실험결과를 표 5와 같이 얻었다. 표 5에서 DSP를 이용한 경우는 처리속도를 보다 빠르게 하기 위해 어셈블리언어를 이용하였고 명령어 수행 방법을 컴파일 방법과 인터프리터 방법으로 각각 구분하여 실험하였다. 이때 컴파일 방법은 고속으로 명령어의 처리하는 방법으로 최초의 운전 지령시 사용자 프로그램을 먼저 DSP 코드에 접합하도록 컴파일 한 후 프로그램 메모리에 저장하여 이를 수행하는 방법이며, 인터프리터 방법은 사용자 프로그램 메모리로부터 각각의 시퀀스 프로그램을 페치(fetch)하여 이를 각각의 명령어 수행 루틴으로 분기하여 처리하는 방법이다. 컴파일 방법은 고속화를 꾀할 수 있으나 메모리의 용량이 본 논문에서 설계된 길이보다 약 10배정도 많고 처리속도는 약 5배정도가 더 길게 되었다. 그러나, 메모리의 크기를 줄이기 위해 인터프리터 방법을 사용하면 메모리는 약 2배정도 길며 처리속도는 약 18배정도 더 길게됨을 실험결과로부터 알 수 있다.

이제 본 논문에서 설계된 시퀀스 제어용 프로세서와 기존의 PLC 시스템과 성능을 비교 평가한다. 이를 위해 기존의 PLC 시스템을 2가지로 분류한다. 즉, 빠른 비트 연산 명령어를 지원하는 MCU(Micro Controller Unit)인 H8/325를 사용한 경우와 ASIC을 사용한 각각의 경우에 대한 명령어 처리속도를 비교 평가하고자한다.

먼저 기존의 MCU를 사용한 경우에는 비트 연산결과를 MCU내의 캐리 플래그로 하였으며, MRG 레지스터는 R0로 설정하여 시퀀스 명령어의 수행속도를 빠르게 하기 위해 컴파일 방법을 사용하였다. 또한 기존의 ASIC을 사용한 경우에 대한 명령어 처리속도를 비교 평가하였다. 이러한 표 5의 결과로부터 알 수 있듯이 본 논문에서 구현된 프로세서가 MCU를 사용한 경우보다는 약 10배 이상 빠름을 알 수 있고 아울러 ASIC을 사용한 경우보다는 약 2배 이상 빠른 것을 알 수 있다.

표 5 각각의 명령어에 대한 처리시간 비교표

Table 5 Execution time table of each instruction

명령어	본 논문에서 설계된 프로세서의 경우		DSP(TMS320C32)를 이용한 경우				MCU(H8/325)를 이용한 경우		기존 PLC의 경우 (ASIC 사용)	
			컴파일 방법		인터프리터 방법					
	수행 속도	명령어 길이	수행 속도	명령어 길이	수행속도	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

5. 결 론

본 논문에서는 FPGA를 이용하여 산업용 자동제어 장치로 널리 사용되고있는 시퀀스 로직 제어용 고속 프로세서를 설계하였다. 이를 위해 VHDL을 이용하여 FPGA를 설계하였으며 FPGA의 설계시 고속처리의 문제점을 해결하기 위해 프로그램 메모리와 데이터 메모리부를 분리 설계함으로써 프로그램 메모리로부터 인스트럭션을 페치하는 도중에 시퀀스명령을 실행하는 프리페치 구조를 피하였으며, 클럭 전용핀을 활용하여 40MHz에서도 동작할 수 있도록 하였다. 아울러 프로세서의 명령어들을 시퀀스 제어에 적합하도록 16비트 또는 32비트로 고정하여 명령어의 디코딩시간과 외부 데이터 메모리의 인터페이스 시간을 줄임으로써 고속화를 실현하였다. 아울러 1스텝운전, PC 브레이크운전, 일정값을 읽거나 쓸 때 정지하는 실시간 디버깅 기능을 구현하여 디버깅을 쉽게 하였다. 이와 같은 기능들을 FPGA로 구현하기 위하여 퀵로직(Quick Logic)사의 pASIC 2 SpDE와 Synplify-Lite 합성툴을 이용하여 로직을 합성하였고, 또한 Verilog HDL 환경에서 최종 시뮬레이션을 성공적으로 수행되었다. 본 논문에서 설계된 FPGA를 84핀 PLCC 형태의 FPGA로 프로그래밍 한 후 256점의 입력과 출력을 갖는 프로그램어블 로직 콘트롤러의 제어시스템에 적용하였다. 본 논문에서 설계된 프로세서의 우수성을 보이기 위해 DSP(TMS320C32-40MHz)를 이용한 시퀀스 제어시스템과 기존의 MCU(H8/325) 및 ASIC을 사용한 제어시스템과 성능을 비교하여 본 논문에서 설계된 시퀀스 제어용 프로세서가 동일 클럭(40MHz)을 사용한 DSP보다 컴파일의 경우 약 5배정도 빠르며 인터프리터 방식의 경우보다는 약 18배정도 빠름을 확인하였고 아울러 MCU를 사용한 경우보다는 10배 이상 빠르며 ASIC을 사용한 경우보다는 2배 이상 빠름을 확인하였다.

참 고 문 헌

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

[2] 안재봉, "PLC 응용기술핸드북," 도서출판 기술, 1993.

[3] A. R. Al-Ali, I. M. EL-Amin, "PLC-Based Motor Protection," in Proceeding of the 5th International Conference on Control, Automation, Robotics and Vision, pp.985-988, 1998.

[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] N. Aramaki, Y. Shimorkawa, S. Kuno, T. Saitoh, H. Hashimoto, "A New Architecture for High-Performance Programmable Logic Controller," in Proceeding of the 23th International Conference on Industrial Electronics, Control, and Instrumentation, Vol. 1, pp.187-190, 1997.

[6] HITACHI Semiconductor, "H8/500 Series Programming Manual : user's manual," 1996.

[7] HITACHI Semiconductor, "H8/510 Series Hardware Manual : user's manual," 1996.

[8] Texas Instruments,"TMS320C3X User's Guide," 1993.

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

[10] Quick Logic, "Quick Works User's Guide with SpDE Reference," 1996.

[11] Quick Logic, "Synplify-Light Verilog and VHDL Synthesis User's Guide for Quick Works version 5.1," 1995.

[12] Quick Logic, "The VERILOG *Golden* REFERENCE Guide," 1997.

저 자 소 개

양 오 (梁 熬)

1959년 10월 24일생. 1983년 한양대 전기공학과 졸업. 1985년 동 대학원 전기공학과 졸업(석사). 1997년 한양대 전기공학과 졸업(공박). 1985년~1997년 LG 산전연구소 책임연구원. 1997년~현재 청주대 전자·정보통신·반도체 공학부 전임강사.

Tel : (0431) 229-8440, Fax : (0431) 229-8440

E-mail : ohyang@chongju.ac.kr