# 단일 칩 8비트 마이크로컨트롤러의 설계 및 구현

# (Design and Implementation of a Single-Chip 8-Bit Microcontroller)

**권성재*, 안정일**, 박성환****
(Sung Jae Kwon, Jung Il Ahn, Sung Hwan Park)

**요 약** 본 논문에서는 마이크로컨트롤러의 기능을 수행하는 데 필수적이며 사용빈도가 높다고 판단되는 총 64개의 명령어를 정의한 후 이를 처리할 데이터패스를 구성해 스테이트 머쉰으로 제어하는 방식으로 VHDL로 설계를 하고 FPGA로 구현했다. 기존의 마이크로컨트롤러 관련 연구에서는 기능적 시뮬레이션까지만 했거나, 인터럽트 기능이 없든지, 하드웨어로 구현을 하지 않았었다. 본 논문에서는 데이터 이동, 논리, 가산 연산 및 분기, 점프 연산을 실행할 수 있도록 해 간단한 연산 및 제어용도에 적합하도록 하였고, 스택, 외부 인터럽트 기능을 지원하도록 해 그 자체로서 완전한 마이크로컨트롤러가 되도록 하였다. 타이밍 시뮬레이션으로 검증 후 제작 과정을 통해, 설계된 마이크로컨트롤러가 정상적으로 동작함을 확인하였다. 심지어 프로그램 ROM까지도 칩 안에 넣어 전체 마이크로컨트롤러를 단일 칩으로 구현하였다. Altera MAX+PLUS II 통합개발환경 하에서 EP1K50TC144-3 FPGA 칩으로 구현을 하였고 최대 동작주파수는 9.39MHz까지 가능했고 사용한 로직 엘리먼트의 개수는 2813개로서 논리 사용률은 97%이었다. 본 연구의 결과는 핵심 기능이 요구되는 마이크로컨트롤러 IP로서도 사용할 수 있고, 모든 코드가 VHDL로 작성되어 있으므로 사용자의 요구에 따라 기능을 추가할 수도 있다.

**핵심주제어** : 설계, 인에이블, FPGA, 구현, 마이크로컨트롤러, 스테이트 머쉰, VHDL

**Abstract** In this paper, we first define a total of 64 instructions that are considered to be essential and frequently used, construct a datapath diagram, determine the control sequence using a finite state machine, and implement an 8-bit microcontroller using FPGA in VHDL. In the past, only functional simulation results of a rudimentary microcontroller were reported, the microcontroller lacked interrupt handling capability, or it was not implemented in hardware. We have designed a self-contained 8-bit microcontroller such that it can perform data transfer, addition, and logical operations, as well as stack and external interrupt operations. Following timing simulation of the designed microcontroller, we implemented it in an FPGA and verified its operation successfully. The design and implementation has been done under the Altera MAX+PLUS II integrated development environment using the EP1K50TC144-3 chip. The maximum operating frequency, the total number of logic elements used, and the logic utilization were found to be 9.39 MHz, 2813, and 97%, respectively. The result can be used as a microcontroller IP, and as needs arise, the VHDL code can be modified accordingly.

**Key Words** : Design, Enable, FPGA, Implementation, Microcontroller, State Machine, VHDL

## 1. 서 론

마이크로프로세서(microprocessor)는 컴퓨터의 중앙처리장치(central processor/processing unit;

* 대진대학교 통신공학과 부교수, 교신저자
** 대진대학교 통신공학과 학부생

CPU)를 단일 칩에 집적시켜 만든 반도체 소자로서, 1971년 Intel이 세계 최초로 4004를 만들었다[1]. 마이크로컴퓨터는 마이크로프로세서에 RAM과 ROM을 추가한 것이며 마이크로컨트롤러는 마이크로컴퓨터에 ADC, DAC, PWM, 포트 등의 제어 블록들을 더 넣은 것이다. 마이크로컴퓨터는 단일 칩으로 된 것도 있으나 여러 개의 칩으로 구성될 수도 있다.

1980년대 초에 마이크로프로세서를 체계적으로 설계하는 방법이 국내에 발표되어 관련 기술의 발전에 많이 이바지하였다[2-5]. 학생들이 TTL 칩만을 사용하여 16개의 명령어를 가지는 소형 CPU를 특정 CAD 툴 없이 수작업에 의해 설계할 수 있는 길을 열어주었다.

최근 들어 국내에서도 많은 마이크로프로세서들이 개발되고 있다. 19개의 명령어를 가지는 간략형 8비트 프로세서가 Xilinx Foundation, Active-HDL, Xilinx FPGA를 사용하여 합성되었다[6]. 하지만 최대 동작주파수, 게이트 카운트 등의 관련 성능 데이터는 논문에 나타나 있지 않다. 필수적인 외부 인터럽트 기능도 없다.

VHDL을 이용하여 Parwan CPU[7]를 모델링하고 시뮬레이션 한 논문이 발표되었다[8]. 또한 명령어 셋이 간단한 파이프라인 컴퓨터를 VHDL로 코딩하고 시뮬레이션 하여 동작을 확인하였다[9]. 하지만 시뮬레이션까지만 하였고 구현 결과는 없다.

한편 코드 밀도가 높은 확장명령형 16비트 컴퓨터의 설계 및 구현 논문도 발표된 바 있다[10]. 사용한 툴 및 FPGA의 종류를 명시하지 않았다. 단지 12,000개의 게이트가 소요되었고 8MHz까지 동작한다고 되어 있다. 32비트인 경우의 연구 결과도 발표되었다[11].

최근에는 특정 용도에 적합한 프로세서들이 많이 개발되고 있으며[12,13] 프로세서의 전력소비를 줄이기 위한 연구도 많이 하고 있다[14].

요즘 많이 사용되고 있는 마이크로컨트롤러로서는 Intel의 8051, Atmel의 AVR, Microchip Technology의 PIC 계열 등을 들 수 있다[15-17]. 본 논문에서는 8051 마이크로컨트롤러 명령어 집합 중 필수적이라고 판단되는 일부를 실행하는 8비트 마이크로컨트롤러를 VHDL과 FPGA를 사용하여 개발하고자 한다. 본 논문의 특징은 모든 설계, 합성, 시뮬레이션을 홈페이지에서 내려 받아 무료로 사용할 수 있는 MAX+PLUS II 통합개발환경[18]을 통해 구현하였고, 간단한 제어용도에 필수적인 명령어들을 정의했고, 설계 과정을 자세히 명시했을 뿐만 아니라, 명령어 정의에서부터 출발해 이를 구현하는 모든 블록들을 VHDL로 코딩하고 시뮬레이션 및 실험을 통해 동작을 완벽하게 검증했다는 점이다. 또한 프로그램 ROM까지도 칩 내부에 통합시켜 단일 칩으로 완전한 8비트 마이크로컨트롤러를 만드는 것을 목적으로 한다.

각각의 명령어들은 스테이트 머쉰(state machine)에 의해 제어되어 원하는 결과를 얻도록 동작하게 된다. 각 명령어는 기본적으로 ST0~ST13까지의 총 14개의 상태를 가진다. 각 상태마다 스테이트 머쉰이 인에이블 신호를 발생시켜 동작시키게 한다. PC(프로그램 카운터)를 동작시키는 인에이블 신호에 의해 명령코드 및 데이터가 들어 있는 ROM의 번지가 증가되게 되며, 해당 번지의 명령어 또는 데이터가 출력된다. 출력된 명령코드는 명령어 레지스터/디코더에 의해 그 명령코드를 각각의 블록으로 보내어 스테이트 머쉰의 제어 하에 동작하게 한다.

2절은 구현할 마이크로컨트롤러의 명령어를 정의하고 구성 블록들을 다룬다. 3절에서는 시뮬레이션을 통해 각각의 명령어 및 블록이 제대로 동작하는지를 확인한다. 몇 가지 연산 프로그램 및 LED 제어 프로그램도 시뮬레이션 하였다. 4절에서는 설계된 마이크로컨트롤러를 FPGA로 구현하여 실험을 통해 검증한다.

## 2. 명령어 정의 및 하드웨어 설계

8비트 마이크로컨트롤러의 경우 단일 바이트로 명령어를 구성한다면 최대 256개의 명령어를 가질 수 있으나 본 마이크로컨트롤러에서는 기본적이며 많이 필요하다고 판단되는 총 64개의 명령어를 구현하기로 결정하였으며, 본 마이크로컨트롤러의 명령어는 기존의 8비트 마이크로컨트롤러인 8051(AT89C51)의 명령어[15,16]를 기반으로 구현하였다. 따라서 연산 코드(op code)를 구현하는 방식은 차이가 있을 수 있지만 연산 코드를 나타내는 니모닉(mnemonic)은 완전히 동일하도록 하였

고 같은 결과를 얻을 수 있도록 하였다. 포트는 입력 포트와 출력 포트를 따로 두었기에 포트 입출력 관련 명령어 니모닉 2개만이 8051과 다르다. 머쉰 코드(machine code)는 우리가 임의로 정의하였기에 8051과는 다르다. 상용 8051은 255개의 명령어를 지원한다[15,16]. 본 논문의 마이크로컨트롤러와 상용 8051과의 다른 점을 표 1에서 비교를 한다. 표에서 볼 수 있듯이 상용 8051만큼 다양한 명령어 및 주변장치는 없지만 기본적인 마이크로컨트롤러로서의 기능을 할 수 있도록 설계하였다.

<표 1> 상용 8051과 본 논문의 마이크로컨트롤러 간의 기능 비교

| 항 목 | 상용 | 본 논문 |
|---|---|---|
| 명령어 | 255개 | 64개 |
| 포트 | 4개 | 1개 |
| 외부 인터럽트 | 2개 | 1개 |
| 스택 | 있음 | 있음 |
| 타이머 | 있음 | 없음 |

본 논문에서 설계한 마이크로컨트롤러의 어드레스 버스의 폭은 16비트이고 데이터 버스의 폭은 8비트이다. 물론 명령어의 길이는 기본적으로 8비트이지만 즉치 데이터(immediate data) 또는 번지를 필요로 하는 명령어의 길이는 각각 총 2바이트 및 총 3바이트이다. 기본적으로 마이크로프로세서는 프로그램이 시키는 대로 계속 일만 할 따름이다. 즉, 페치(fetch), 해독(decode), 실행(execute)의 3가지 사이클을 계속 반복한다. 그러므로 마이크로프로세서의 설계는 페치해온 명령어를 해독해 해당 작업을 수행해주도록 하면 된다.

표 2는 본 논문에서 구현할 마이크로컨트롤러의 명령어 집합이다. 2바이트 즉치(immediate) 데이터 명령어인 MOV A, DATA, 3바이트 명령인 JMP ADDR, CALL ADDR, JNZ ADDR, RET, RETI를 제외한 모든 경령어는 1바이트 길이의 명령어이다. 표 2의 명령어를 실행하기 위한 하드웨어의 블록도는 그림 1과 같이 주어진다. 기능으로 보면 마이크로컨트롤러는 ALU와 컨트롤 유니트의 두 부분으로 구성된다. 마이크로컨트롤러의 설계는 데이터 패스를 먼저 설계한 다음 이를 제어하는 컨트롤 로직을 만드는 순서로 설계한다.

컨트롤은 다이크로프로그래밍, 한 스테이트 당 한 개의 플립플롭 등으로 할 수 있는데 본 논문에서는 스테이트 머쉰을 사용하여 설계를 진행한다. 각 블록도는 스테이트 머쉰에 의해 발생하는 인에이블 신호에 의해 동작하며 마이크로컨트롤러 설계 시 가장 중심되는 블록은 스테이트 머쉰이다. 일단 프로그램 카운터가 다음에 실행한 번지를 증가시키고 ROM에서는 각 주소에 들어있는 데이터를 내보내게 된다. ROM에서 출력되어 나오는 것은 명령코드나 데이터, 또는 점프할 번지 값이 되기도 한다. 이것 역시 스테이트 머쉰에서 발생되는 인에이블 신호에 따라 결정되게 된다. 명령어 레지스터/디코더는 ROM의 값을 받아서 명령코드로 만들고 그 값이 명령코드에 따라 다른 값을 내는 블록들(그림 1의 MUX, 스테이트 머쉰, ALU)에 들어가 각 명령에 맞게 동작하게 한다. 각각의 레지스터는 각 명령어에 따라 그 블록의 인에이블 신호가 활성화 되어 동작하게 된다.

본 마이크로컨트롤러 설계에서는 레지스터를 R0~R7까지의 8개로 하였으며 스택 구조를 갖는 레지스터를 설계함으로써 마이크로컨트롤러의 중요한 부분이라고 할 수 있는 내부 및 외부 인터럽트를 추가하였다. 마이크로컨트롤러의 설계는 먼저 명령어를 작성하고 그것에 맞게 전체 블록도를 작성하고 각 명령어와 스테이트 머쉰을 정한 후에 각각의 블록도를 설계하고 합성을 하는 순서로 진행한다.

구성 블록들 중에서 중요한 몇 가지를 자세히 살펴본다. 우선 스테이트 머쉰은 마이크로컨트롤러를 설계하는 데 있어서 가장 중요한 블록이다. 그림 2의 상태도에서처럼 각각의 명령어에 따라 데이터패스를 활성화시켜주는 인에이블(enable) 신호를 만들어주기 때문이다. 클락 신호의 상승 에지(rising edge)에서 활성화되지만 인에이블 신호를 결정하는 것은 각 명령어의 명령코드이다. 본 논문에서 사용하는 스테이트 머쉰은 ST0부터 ST18까지 총 19개의 상태로 구성되는 유한 스테이트 머쉰(FSM; finite state machine)이다. 일반 명령어의 경우에는 ST0부터 ST13까지의 상태를 클락 신호의 상승 에지에서 한 상태씩 옮겨가지만 점프(또는 분기) 명령어 등의 경우에는 클락 신호의 상승 에지에서 상태가 ST0에서부터 ST18까지 한 단계씩 이동한다.

다음으로 프로그램 카운터(program counter;

PC)는 프로그램의 수행을 제어하는 카운터로서 명령어를 페치할 번지를 가리키고 특별한 경우가 아닌 경우 자동으로 명령어의 길이만큼 증가하여 다음 번지를 지정할 준비를 한다. 여기서는 ROM의 번지를 지정하고 다음 명령어가 수행되기 전에 자동적으로 그 다음 번지를 지정한다. enable0은 PC 값을 하나씩 증가시키고 enable5는 ROM에서 출력되는 점프할 번지를 받아들이고 enable8은 점프할 번지부터 실행되도록 출력 값을 바꾸는 역할을 한다. enable13은 내부 인터럽트 시 스택에 저장된 복귀주소를 내보내는 역할을 하며 enable15는 외부 인터럽트 시 인터럽트 벡터를 내보내는 역할을 한다.

또한 ROM(read-only memory)은 각 주소에 명령코드(OP 코드) 또는 피연산자(operand)를 저장하고 있다. 입력 주소는 16비트이고 출력 데이터는 8비트이다. ROM은 FPGA 외부에 연결할 수도 있으나 ROM은 기본적으로 민텀의 합(sum of minterms)으로 표현되는 조합논리이므로 본 논문에서는 FPGA 내부에서 구현하였다.

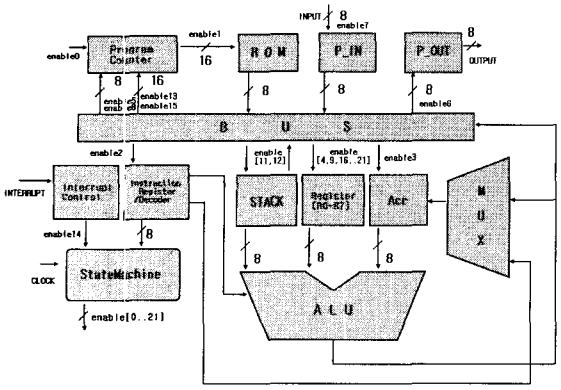
한편 스택(stack)은 인터럽트 발생 시 현재 돌아가고 있는 프로그램의 Acc 값과 R0~R7 값, 주소를 저장하는 레지스터이다. 인터럽트가 끝난 후 다시 인터럽트 발생 전의 프로그램으로 돌아갈 때 스택에서 복귀 주소를 불러오며 실행됐던 인터럽트 전 프로그램의 Acc 값과 R0~R7 값도 불러와 실행하던 프로그램을 계속 실행할 수 있게 하는 역할을 한다.

그리고 EAREG(Enable All Interrupt Register)는 외부 인터럽트를 사용할 때 설정해주는 레지스터로서 이 값이 1일 경우에만 외부 인터럽트가 동작하게 된다. INT는 외부 인터럽트 요청 신호와 EAREG 신호를 AND시켜 이 값이 1일 경우에 인터럽트 동작 신호를 내보내게 된다. 외부 인터럽트 요청 신호의 하강 에지(falling edge)에서 인터럽트가 걸리도록 하였다.

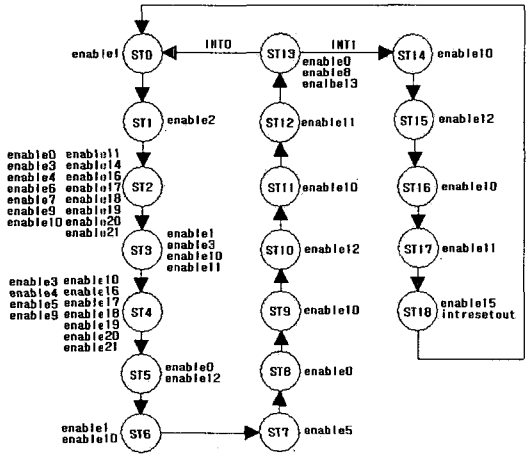
마이크로컨트롤러를 구성하는 각 블록은 VHDL로 작성하였고 그림 3의 전체 블록도는 그래픽 디자인 포맷(gdf)인 스키매틱 형태로 구성하였다. 마이크로컨트롤러의 전체 블록도를 그림 4와 같이 한 개의 심벌로 나타내어 다른 디자인에서 한 개의 블록으로 임포트(import)할 수 있도록 해주는 계층적 설계도 가능하다. DATA_IN 단자는 P_IN 포트의 입력 단자이고 DATA_OUT 단자는 P_OUT 포트의 출력 단자이다. RESET 핀은 마이크로컨트롤러를 리셋해주고 CLK은 마이크로컨트롤러를 동기적으로 구동하는 클락 신호 입력이며 INTERRUPT 핀은 외부 인터럽트 요청 입력이다.

<표 2> 실행할 명령어 집합의 정의

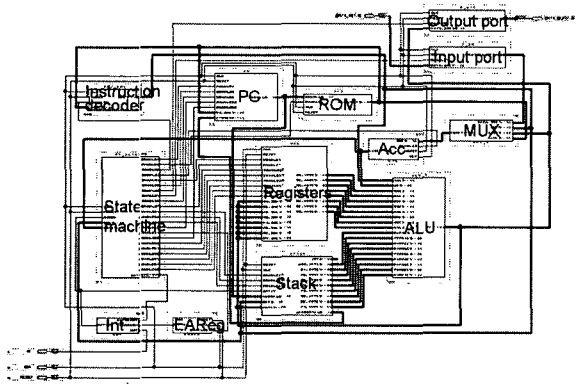
| 연산 코드 | 니모닉 | 연산 코드 | 니모닉 |
|---|---|---|---|
| NOP | 00000000 | MOV A, P_IN | 01110000 |
| CLR A | 00010000 | MOV P_OUT,A | 10000000 |
| MOV A, R0 | 00100000 | ADD A, R0 | 10010000 |
| MOV A, R1 | 01000000 | ADD A, R1 | 10010001 |
| MOV A, R2 | 00100001 | ADD A, R2 | 10010010 |
| MOV A, R3 | 00100010 | ADD A, R3 | 10010011 |
| MOV A, R4 | 00100011 | ADD A, R4 | 10010100 |
| MOV A, R5 | 00100100 | ADD A, R5 | 10010101 |
| MOV A, R6 | 00100101 | ADD A, R6 | 10010110 |
| MOV A, R7 | 00100110 | ADD A, R7 | 10010111 |
| MOV R0, A | 00110000 | ORL A, R0 | 10100000 |
| MOV R1, A | 01010000 | ORL A, R1 | 10100001 |
| MOV R2, A | 00110001 | ORL A, R2 | 10100010 |
| MOV R3, A | 00110010 | ORL A, R3 | 10100011 |
| MOV R4, A | 00110011 | ORL A, R4 | 10100100 |
| MOV R5, A | 00110100 | ORL A, R5 | 10100101 |
| MOV R6, A | 00110101 | ORL A, R6 | 10100110 |
| MOV R7, A | 00110110 | ORL A, R7 | 10100111 |
| MOV A, DATA | 01100000 | CPL A | 10110000 |
| INC A | 11000000 | DEC A | 11010000 |
| JMP ADDR | 11100000 | JNZ ADDR | 11110000 |
| PUSH A | 11110001 | POP A | 11110110 |
| PUSH R0 | 11110010 | POP R0 | 11110101 |
| PUSH R1 | 11110011 | POP R1 | 11110110 |
| PUSH R2 | 11111010 | POP R2 | 00000001 |
| PUSH R3 | 11111011 | POP R3 | 00000010 |
| PUSH R4 | 11111100 | POP R4 | 00000011 |
| PUSH R5 | 11111101 | POP R5 | 00000100 |
| PUSH R6 | 11111110 | POP R6 | 00000101 |
| PUSH R7 | 11111111 | POP R7 | 00000110 |
| CALL ADDR | 11110111 | RET | 11111000 |
| SETB EA | 11111001 | RETI | 10001000 |



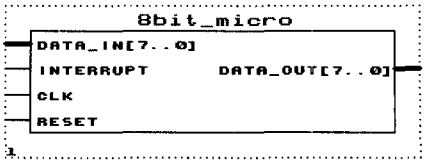

<그림 1> 설계한 마이크로컨트롤러의 구조




<그림 2> 스테이트 머쉰의 상태도




<그림 3> 설계한 마이크로컨트롤러의 전체 블록도




<그림 4> 설계한 마이크로컨트롤러의 전체 블록도를 하나의 심벌로 표시

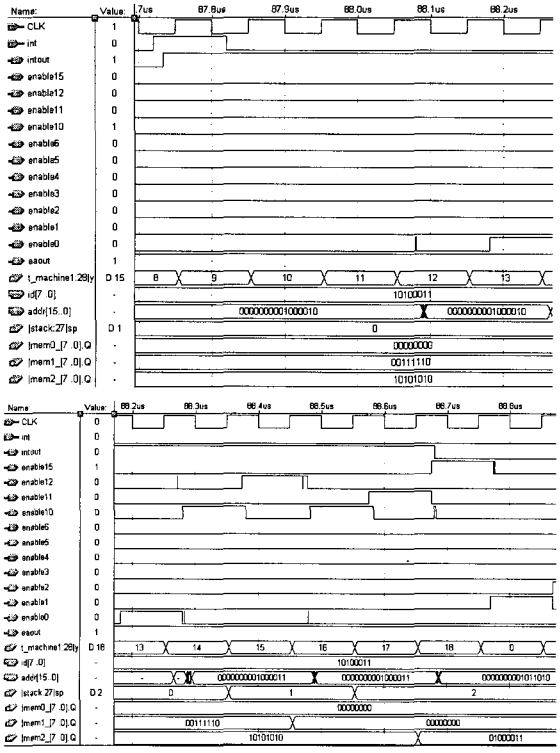
## 3. 시뮬레이션

설계된 블록 각각에 대해 타이밍 시뮬레이션을 통해 제대로 동작하는지를 확인한 후 정의한 64개의 명령어 각각에 대해서 시뮬레이션을 수행해 동작을 검증하였다. 그 다음 전체 블록도에 대해서 동작을 검증하였다. 그 중의 하나인 외부 인터럽트의 경우에 대해 자세히 설명하면 다음과 같다.

표 3은 스테이트 머쉰의 각 상태에서 인에이블되는 신호를 보여준다. 그림 5에서 위 그림과 아래 그림은 서로 시간적으로 연결되는 파형이다. 그림에서 인터럽트(int)가 들어오면 인터럽트 인지 신호(intout)는 상태가 ST18로 되기 전까지 유지됨을 알 수 있다. 이는 ST13에서의 판별뿐만 아니라 그 이후에 인터럽트를 실행하기 위한 현재의 주소를 스택에 저장하는 과정에서 인터럽트 신호를 판별하기 위한 신호이다. 인터럽트 요청이 들어오면 ST13에서 그 신호의 여부를 판별하여, 인터럽트 신호가 들어왔을 경우 ST14~ST18 과정을 추가로 진행하게 된다. 상태 ST14~ST17까지는 현재의 주소를 STACK에 저장하기 위한 것이며 ST18은 인터럽트 요청 시 미리 정해놓은 주소(인터럽트 벡터)로 가도록 해준다. 여기서는 그 주소를 "0000000001011010(005AH)"으로 정해 주었다.

<표 3> 상태에 따라 어서트되는 신호

| ST13 | ST14 | ST15 | ST16 | ST17 | ST18 |
|---|---|---|---|---|---|
| INT 판별 | enable 10 (SP) | enable 12 | enable1 0 (SP) | enable 11 | enable 15 |

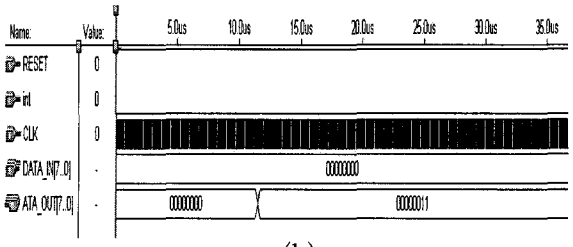


<그림 5> 인터럽트 요청 시 타이밍 다이어그램: 윗 그림 다음에 아래 그림이 시간적으로 연결된다(시간축 값 참조)

첫 번째 프로그램은 7에서 4를 감산하는 연산을 수행하는 역할을 하고 프로그램 리스팅은 그림 6(a)에 주어져 있고 시뮬레이션 결과는 그림 6(b)에 제시하였다. 프로그램에서 볼 수 있듯이 감산을 하지 않고 감수(subtrahend)인 4에 대한 2의 보수를 구하고 이 값을 7에 더해주는 가산을 하여 감산을 하게 된다. 타이밍 다이어그램에서 P_OUT을 나타내는 DATA_OUT 신호를 보면 값이 3임을 알 수 있다.

```
ORG 0000H; 0번지부터 저장
MOV A, #00000111B
MOV R1. A
MOV A, #00000100B
CPL A
INC A
ADD A, R1
MOV P_OUT, A
END
```

(a)



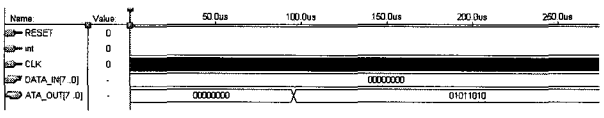
(b)

<그림 6> 첫 번째 프로그램: (a) 리스팅 (b) 타이밍 다이어그램

두 번째 프로그램은 0~18까지의 숫자 중 짝수를 더해주는 프로그램으로서, 리스팅은 그림 7(a)에 제시되어 있고 시뮬레이션 결과는 그림 7(b)에 제시하였다. 프로그램을 보면 처음에 18을 Acc에 넣어주고 2만큼씩 줄여가면서 그 값을 더해나가는 과정으로 연산하였고 반복문을 이용하여 Acc 값이 0이 되면 연산을 끝내고 출력하도록 하였다. 그림 7(b)에 주어진 결과 파형에서 볼 수 있듯이 연산 결과 값이 90(01011010)으로 제대로 출력됨을 볼 수 있다.

```
        ORG 0000H
        MOV A, #00010010B; Acc 값 18
        MOV R0, A; 부분합을 저장
HERE:   DEC A
        DEC A
        MOV R1, A; 이번에 더할 값을 피신
        ADD A, R0; 부분합의 값을 갱신
        MOV R0, A; 갱신된 부분합을 피신
        MOV A, R1
        JNZ HERE
        MOV A, R0; R0 값을 Acc로 복사
        MOV P_OUT A; Acc 값을 출력
        END
```

(a)



(b)

<그림 7> 두 번째 프로그램: (a) 리스팅 (b) 타이밍 다이어그램

세 번째 프로그램은 LED를 제어하는 프로그램

으로서 메인 프로그램과 인터럽트 서비스 루틴으로 구성되어 있다. 설계한 마이크로컨트롤러가 외부의 인터럽트 요청을 제대로 처리해주는지를 확인하기 위한 것이다. 그림 8(a)는 메인 프로그램으로 숫자 255까지 순서대로 카운트하는 프로그램이다. 그림 8(b)는 외부 인터럽트를 인가 시에 실행되는 인터럽트 서비스 루틴으로 한 비트씩 왼쪽으로 쉬프트시키는 프로그램이다. 2번 반복 후에 메인 프로그램으로 복귀한다. 이때의 복귀는 인터럽트가 요청된 시점의 다음 숫자부터 다시 카운트하게 된다.

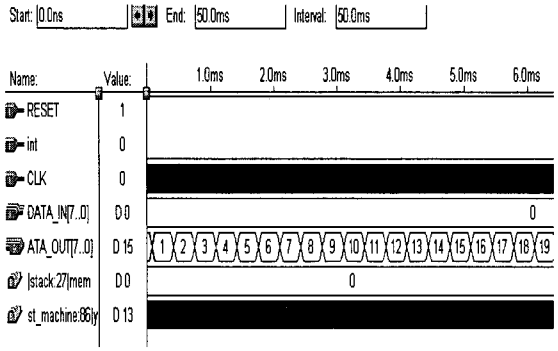
그림 8(c), 8(d), 8(e)는 위의 어셈블리 프로그램을 실행한 결과 파형들이다. 그림 8(c)은 메인 프로그램의 실행 결과로서 1부터 255까지 카운트되는 결과 파형이고 그림 8(d)는 카운트하는 중 외부 인터럽트를 인가하였을 때 왼쪽으로 한 비트씩 이동하는 인터럽트 프로그램의 실행결과를 나타낸 파형이다. 그림 8(e)은 인터럽트 복귀 시에 실행되는 모습을 보여준다. 그림 8(d), 8(e)에서 보는 것처럼 외부 인터럽트가 실행되고 복귀할 때 외부 인터럽트 인가 전에 실행했던 명령어 다음부터 실행시킴을 알 수 있다.

```
        ORG 0000H; 메인 프로그램
        SETB EA; 외부인터럽트 ON시킴
NEXT2:  MOV A, #00000000B
        MOV R2, A
        MOV A, #00000000B
        MOV R1, A
NEXT1:  INC A; Acc 값을 1 증가시킴
        MOV P_OUT, A; Acc 값을 출력
        MOV R1, A
        MOV A, R2; Acc에 0이 들어감
LOOP1:  DEC A; Acc 값을 1 감소시킴
        MOV R3, A
        MOV A, R2; 내부 루프 반복 횟수
LOOP2:  DEC A
        JNZ LOOP2
        MOV A, R3
        JNZ LOOP1
        MOV A, R1
        JNZ NEXT1
        JMP NEXT2; 다시 반복
        END
```

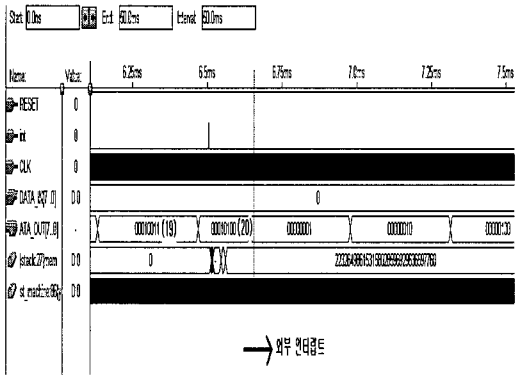
(a)

```
        ORG 005AH; 인터럽트 서비스 루틴
        PUSH A; 현재 Acc 값을 스택에 저장
        PUSH R0; 현재 R0 값을 스택에 저장
        PUSH R1; 현재 R1 값을 스택에 저장
        PUSH R2; 현재 R2 값을 스택에 저장
        PUSH R3; 현재 R3 값을 스택에 저장
        MOV A, #00000000B
        MOV R1, A; R1에 Acc 값을 복사함
        MOV A, #00000001B
        MOV A, R3; Acc에 R3 값을 복사함
NEXT4:  DEC A; Acc의 값을 1 감소시킴
        MOV R3, A; R3에 Acc 값을 복사함
        MOV A, #00000001B
        MOV R0, A; R0에 Acc 값을 복사함
NEXT3:  MOV P_OUT, A; Acc 값을 출력
        ADD A, R0
        MOV R0, A; R0에 Acc 값을 복사함
        MOV A, R1; Acc에 R1 값을 복사함
LOOP3:  DEC A; Acc 값을 1 감소시킴
        MOV R2, A; R3에 Acc 값을 복사함
        MOV A,R1; Acc에 R2 값을 복사함
LOOP4:  DEC A; Acc에 R2 값을 복사함
        JNZ LOOP4
        MOV A, R2
        JNZ LOOP3
        MOV A, R3
        JNZ NEXT4
        MOV A, R1
        JNZ NEXT3
        POP R3; 스택에 저장된 R3 값을 로드
        POP R2
        POP R1
        POP R0
        POP A
        RETI; 인터럽트 처리 후 복귀
        END
```

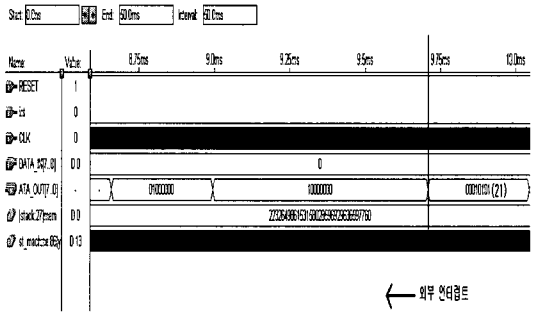
(b)




(c)




(d)




(e)

<그림 8> 세 번째 프로그램: (a) 메인 프로그램 리스팅 (b) 인터럽트 서비스 루틴 리스팅 (c) 메인 프로그램의 타이밍 다이어그램 (d) 인터럽트 서비스 루틴의 타이밍 다이어그램 (e) 외부 인터럽트 복귀시 타이밍 다이어그램

## 4. 실 험

본 마이크로컨트롤러를 테스트하기 위한 회로도를 그림 9에 나타내었다. 입출력 신호들은 다음과 같다. DATA_IN(비트 0~7)은 사용자가 넣고자 하는 데이터를 입력시키기 위한 핀, DATA_OUT (비트 0~7)은 데이터 출력 핀, CLK은 외부 클락 신호를 입력받는 핀, RST는 리셋 핀, INT는 외부 인터럽트 핀을 나타낸다.

즉, DATA_IN은 입력 포트를 의미하며 DATA_OUT은 출력 포트를 구현한 것이다.

FPGA에서 일부 핀을 제외한 나머지 핀은 사용자가 임의로 지정하여 사용할 수 있으나 칩 회사가 권장하는 핀을 사용하였다. 144개의 총 핀 수 중 사용자 입출력 핀 수는 모두 102개인데 이중 19개의 핀을 사용하였다(그림 4, 9). 입력 데이터 핀 8개, 출력 데이터 핀 8개, 클락, 리셋, 인터럽트 핀 각 1개씩해서 모두 19개이다.

상기 회로도를 기반으로 제작한 PCB는 그림 10에 제시하였다. 좌측 중간 밑에 리셋 스위치, 좌측 가장 밑에 인터럽트 버튼, 우측 하단에 표시용 LED가 8개 있는 것을 볼 수 있다. 크리스털 발진기 위의 칩은 74LS163 TTL으로서 분주용이다. 우측 상단의 칩은 직렬 EEPROM이다.

FPGA 칩은 비교적 저렴한 ACEX1K 계열의 칩인 EP1K50TC144-3를 사용하였다. 이 칩은 총 2880개의 로직 엘리먼트(logic element)를 가지고 있다. 게이트 수로는 약 5만 개이다. 뒤의 숫자 3은 스피드 등급을 나타내는데 값이 클수록 칩의 동작 주파수는 낮아진다.

1M비트 용량의 직렬 EEPROM인 AT17LV010A (FPGA configurator memory)에 pof 파일 내용을 프로그램하여 보드에 장착시키고 전원이 인가되면 FPGA 칩이 ROM의 내용을 자동으로 수동 직렬 (passive serial) 방식으로 로딩하도록 하였다. nCONFIG 핀에 파워 온 리셋이 걸린 후 HIGH 레벨로 되면 DCLK, DATA 핀을 통해 직렬 EEPROM이 FPGA에 컨피규레이션 데이터를 공급하여 FPGA의 LUT(look-up table) 및 배선을 설계한 대로 구성해주며 완료가 되면 CONF_DONE 신호가 HIGH로 된다.

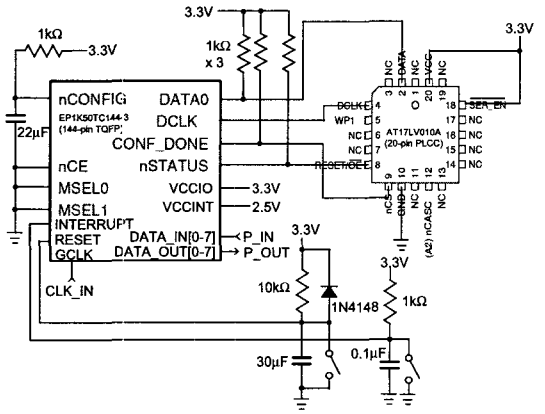
크리스털 발진기의 클락 주파수는 16MHz이나 바로 위에 있는 74LS163 TTL IC를 사용하여 1MHz로 만든 후 Altera 칩의 GCLK(global clock) 단자로 입력해주었다.

사용한 전원 전압을 보면 TTL 및 크리스털 발진기는 5V이고 AT17LV010A[15]는 3.3V, FPGA 칩에서는 코어(VCCINT)는 2.5V, 입출력(VCCIO)

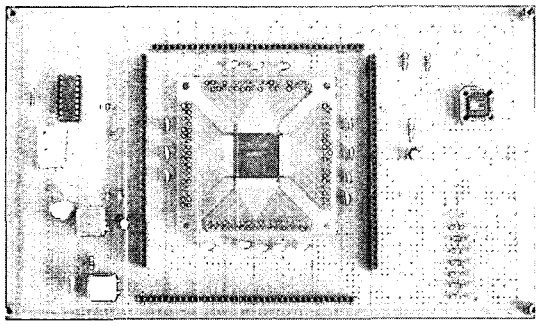
은 3.3V를 공급해주었다.

3절에 제시된 프로그램들을 실행시켜 결과가 모두 제대로 나옴을 확인할 수 있었다.

FPGA 칩의 로직 엘리먼트의 사용률은 97%로 나왔고 최대 동작주파수는 약 10MHz까지 가능하였다.




<그림 9> 설계한 8비트 마이크로컨트롤러의 테스트 보드의 전체 회로도



<그림 10> FPGA로 구현한 8비트 마이크로컨트롤러를 실장한 테스트용 PCB

## 5. 결 론

본 논문에서는 기본적인 명령어 세트를 구현하는 8비트 마이크로컨트롤러를 VHDL을 이용하여 설계하고 FPGA 칩으로 구현하여 제대로 동작함을 확인하였다. 명령어 니모닉은 8051과 유사하게 했으며 총 64개의 명령어를 구현하였다. 기존에는 기능적 시뮬레이션까지만 하든지, 타이밍 시뮬레이션을 했더라도 하드웨어로 구현하지 않은 경우가 많았다. 따라서 본 논문에서는 간단하지만 핵심적인 명령어를 갖는 실용적인 마이크로컨트롤러를 처음부터 끝까지 설계 및 구현하는 방법을 체계적으로 제시하였다.

명령어 하나하나 시뮬레이션을 통해 동작을 확인한 후 세 가지 종류의 어셈블리 언어 프로그램의 실행을 통해 마이크로컨트롤러가 제대로 동작함을 확인하였다. 또한 FPGA 보드를 제작하는 실험을 통해 본 논문에서 설계한 마이크로컨트롤러가 실제 상황에서 제대로 동작함을 확인할 수 있었다.

추후 스테이트 머쉰을 구성하는 상태 수를 줄일 필요가 있으며 회로 구조 등에서도 최적화를 더 시킬 필요가 있다고 사료된다. 물론 상용 8051과 비교할 때 명령어 개수, 주변장치, 기능적인 측면에서 본 논문에서 설계 및 구현한 마이크로컨트롤러를 보완해야 할 필요가 있다. 그런데 엔지니어링 경험에 의하면 우리가 상용 8051을 사용할 때 모든 기능을 다 사용하는 것이 아니라 통상 일부 기능만을 사용하는 경우가 많으므로 본 논문에서 제안한 마이크로컨트롤러를 원하는 용도에 맞게 필요에 따라 최적화를 시켜주면 이점이 있을 것으로 예상된다. 또한 마이크로컨트롤러를 필요로 하는 다른 시스템의 구현에서도 IP로 넣어 활용할 수도 있을 것으로 전망된다.

## 참 고 문 헌


[1] The Intel 4004, http://www.intel4004.com.

[2] 김명환, "디지탈 시스템과 마이크로프로세서 설계(I)," 대한전기학회지, 31권, 7호, pp. 501-504, 1982.

[3] 김명환, "디지탈 시스템과 마이크로프로세서 설계(II)," 대한전기학회지, 31권, 8호, pp. 578-585, 1982.

[4] 김명환, "디지탈 시스템과 마이크로프로세서 설계(III)," 대한전기학회지, 31권, 9호, pp. 652-659, 1982.

[5] 김명환, "디지탈 시스템과 마이크로프로세서 설계(IV)," 대한전기학회지, 31권, 10호, pp. 710-718, 1982.

[6] 송호정, 송기용, "HDL을 이용한 간략형 8-Bit

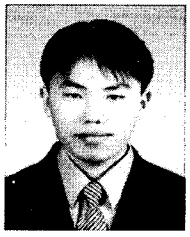
프로세서의 설계," 한국신호처리시스템학회 추계학술대회 논문집, pp. 241-244, 2000.
[7] Z. Navabi, VHDL: Analysis and Design of Digital Systems, McGraw-Hill, 1998.
[8] 박두열, "VHDL을 이용한 Parwan CPU의 Modeling과 Design," 한국컴퓨터정보학회 논문지, 7권, 2호, pp. 19-33, 2002.
[9] 박두열, "VHDL을 이용한 파이프라인 SIC의 시뮬레이션," 한국컴퓨터정보학회 논문지, 8권, 2호, pp. 24-32, 2001.
[10] 조경연, "16 비트 EISC 마이크로 프로세서에 관한 연구," 멀티미디어학회 논문지, 3권, 2호, pp. 192-200, 2000.
[11] 조경연. "확장 명령어 32 비트 마이크로 프로세서에 관한 연구," 대한전자공학회 논문지, 36권, D편, 5호, pp. 391-400, 1999.
[12] 최병윤, 장종욱, "TCP/IP 프로토콜 스택을 위한 리스크 기반 송신 래퍼 프로세서 IP 설계," 한국해양정보통신학회 논문지, 8권, 6호, pp. 1166-1174, 2004.
[13] 정진우, 김성철, "내장형 네트워크 프로세서의 설계 및 구현," 한국해양정보통신학회 논문지, 9권, 6호, pp. 1211-1217, 2005.
[14] 김재우, 김영훈, 오민석, 남기훈, 이광엽, "저전력 기법을 적용한 ARM7 마이크로프로세서의 FPGA 구현 및 측정," 대한전자공학회 하계종합학술대회 논문집, 27권, 1호, pp. 423-426, 2004.
[15] Intel Corporation, http://www.intel.com
[16] Atmel Corporation, http://www.atmel.com.
[17] Microchip Technology Inc., http://www.microchip.com.
[18] Altera Corporation, http://www.altera.com.

**권 성 재** (Sung Jae Kwon)

- 1984년 경북대학교 전자공학과 공학사
- 1986년 한국과학기술원 전기 및 전자공학과 공학석사
- 1990년 한국과학기술원 전기 및 전자공학과 공학박사
- 1990년 ~ 1997년 LG전자 멀티미디어연구소 책임연구원
- 1997년 ~ 현재 대진대학교 통신공학과 부교수
- 관심분야 : 영상 및 통신 시스템)

**안 정 일** (Jung Il Ahn)

- 2006년 대진대학교 통신공학과 공학사
- 관심분야: 영상 시스템, 디지털 시스템



**박 성 환** (Sung Hwan Park)

- 2006년 대진대학교 통신공학과 공학사
- 관심분야: 마이크로프로세서, 디지털 시스템