FPGA 기반 실용적 마이크로프로세서의 구현

안정일, 박성환, 권성재
대진대학교 통신공학과

FPGA-Based Implementation of a Practical 8-Bit Microprocessor

Jung-Il Ahn, Sung-Hwan Park, and Sung-Jae Kwon
Department of Communication Engineering, Daejin University

요 약

본 논문에서는 마이크로프로세서의 기능을 수행하는 데 필수적이며 사용빈도가 높은 총 64개의 명령어를 정의한 후 이를 처리할 데이터패스를 구성해 스테이트 머쉰으로 제어하는 방식으로 실용적 8비트 마이크로프로세서를 VHDL로 설계를 하고 FPGA로 구현했다. 통상 마이크로프로세서 관련 논문에서는 기능적 시뮬레이션까지만 했거나, 인터럽트 기능이 없든지, 하드웨어로 구현을 하지 않았거나, 또는 개발 관련 내용이 자세히 제시되지 않았었다. 본 논문에서는 데이터 이동, 논리, 가산 연산뿐만 아니라 분기, 점프 연산도 실행할 수 있도록 해 연산 및 제어용도에 적합하도록 하였고, 스택, 외부 인터럽트 기능까지도 지원하도록 해 그 자체로서 완전한 실용적 마이크로프로세서가 되도록 하였다. 또한 프로그램 ROM까지도 칩 안에 넣어 전체 마이크로프로세서를 단일 칩으로 구현하였다. 타이밍 시뮬레이션으로 검증 후 제작 과정을 통해, 설계된 마이크로프로세서가 정상적으로 동작함을 확인하였다. 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.

1. 서 론

마이크로프로세서는 컴퓨터의 중앙처리장치를 단일 칩에 집적시켜 만든 반도체 소자로서, 1971년 Intel이 세계 최초로 4004를 만들었다[1]. 마이크로프로세서의 효시인 4004는 어드레스와 데이터 라인이 멀티플렉스된 4비트 마이크로프로세서로서 750kHz의 클락 주파수까지 동작하였고 2300개의 트랜지스터로 구성되었으며 우선적으로 전자계산기에 사용하고자 개발하였다.

1980년대 초에 마이크로프로세서를 체계적으로 설계하는 방법이 국내에 발표되어 관련 기술의 발전에 많이 이바지하였다[2]. 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 개발이 되고 있다.

19개의 명령어를 가지는 간략형 8비트 프로세서가 Xilinx Foundation, Active-HDL, Xilinx FPGA를 사용하여 합성되었다[3]. 하지만 최대 동작주파수, 게이트 개수 등의 관련 성능 데이터는 논문에 나타나 있지 않다. 필수적인 외부 인터럽트 기능도 없다.

VHDL을 이용하여 Parwan CPU를 모델링하고 시뮬레이션 한 논문이 발표되었다[4,5]. 또한 명령어 셋이 간단한 파이프라인 컴퓨터를 VHDL로 코딩하고 시뮬레이션 하여 동작을 확인하였다[6]. 하지만 시뮬레이션까지만 하였고 구현 결과는 제시하지 않았다.

한편 코드 밀도가 높은 확장명령형 16비트 컴퓨터의 설계 및 구현 논문도 발표된 바 있다[7]. 사용한 툴 및 FPGA의 종류는 명시되지 않았고 12,000개의 게이트가 소요되었고 8MHz까지 동작한다고 되어 있다. 32비트인 경우의 연구 결과도 발표되었다[8].

본 논문에서는 8051 마이크로컨트롤러 명령어 집합 중 필수적이라고 판단되는 명령어들을 실행하는 실용적 8비트 마이크로프로세서를 VHDL과 FPGA를 사용하여 개발하고자 한다. 본 논문의 특징은 모든 설계, 합성, 시뮬레이션을 홈페이지에서 내려받아 무료로 사용할 수 있는 Altera MAX+PLUS II 통합개발환경[9]을 통해 구현하였

고, 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 FPGA 칩으로 완전한 8비트 마이크로프로세서를 만드는 것을 목적으로 하였다.

각각의 명령어들은 스테이트 머쉰에 의해 제어되어 원하는 결과를 얻도록 동작하게 된다. 각 명령어는 기본적으로 ST0~ST13까지의 총14개의 상태를 가지며 각 상태마다 스테이트 머쉰(state machine)에서 인에이블 신호를 발생시켜 동작하게 되고 프로그램 카운터를 동작시키는 인에이블 신호에 의해 명령코드 및 데이터를 갖고 있는 ROM의 번지를 증가시켜 주고 그 번지에 들어있는 명령어 또는 데이터를 출력시킨다. 출력된 명령코드는 명령어 레지스터/디코더에 의해 그 명령코드를 각각의 블록으로 보내어 스테이트 머쉰의 제어 하에 동작하게 한다.

2. 명령어 정의 및 하드웨어 설계

본 논문에서 설계한 마이크로프로세서의 어드레스 버스의 폭은 16비트이고 데이터 버스의 폭은 8비트이다. 물론 명령어의 길이는 기본적으로 8비트이지만 즉치 데이터(immediate data) 또는 번지를 필요로 하는 명령어의 길이는 각각 총2바이트 및 총3바이트이다. 기본적으로 마이크로프로세서는 저장된 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 1은 본 논문에서 구현할 마이크로프로세서의 명령어 집합으로서 AT89S51 명령어와 유사하다[10]. 2바이트 즉치(immediate) 데이터 명령어인 MOV A, DATA, 3바이트 명령인 JMP ADDR, CALL ADDR, JNZ ADDR를 제외한 모든 명령어는 1바이트 길이의 명령어이다. 표 1의 명령어들을 실행하기 위한 하드웨어의 블록도는 그림 1과 같이 주어진다. 기능으로 보면 마이크로프로세서는 ALU와 제어 유니트의 두 부분으로 구성된다. 마이크로프로세서의 설계는 데이터패스를 먼저 설계한 다음, 이를 제어하는 컨트롤 로직을 만드는 순서로 설계한다.

본 논문에서는 스테이트 머쉰을 사용하여 설계를 진행한다. 각 블록도는 스테이트 머쉰에 의해 발생하는 인에이블 신호에 의해 동작하며 마이크로프로세서 설계 시 가장 중심이 되는 블록은 스테이트 머쉰이다. 일단 프로그램 카운터가 다음에 실행할 명령어를 가리키도록 번지를 증가시키고 ROM에서는 각 번지에 저장된 명령어 코드, 데이터, 또는 번지를 내보내게 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 동작한다. 각각의 레지스터는 명령어에 따라 그 블록의 인에이블 신호가 발생 시 동작하게 된다.

본 마이크로프로세서의 설계에서는 레지스터를 R0~R7의 8개로 하였으며 스택 구조를 갖는 레지스터를 설계함으로써 마이크로프로세서의 중요한 부분이라 할 수 있는 인터럽트 기능을 추가하였다. 마이크로프로세서의 설계는 먼저 명령어를 작성하고 그것에 맞게 전체 블록도를 작성하고 각 명령어와 스테이트 머쉰을 정한 후에 각각의 블록도를 설계하고 합성을 하는 순서로 진행한다.

마이크로프로세서를 구성하는 블록들은 그림 1에서처럼 주어져 있다. 우선 스테이트 머쉰은 8bit 마이크로프로세서를 설계하는 데 가장 중요한 블록이다. 왜냐하면 명령어에 따라 마이크로프로세서의 데이터패스를 활성화시켜주는 인에이블(enable) 신호를 만들어주기 때문이다. 클락 신호가 들어왔을 때 발생하지만 인에이블 신호를 결정하는 것은 각 명령어의 명령코드이다. 인에이블 신호가 클락 신호의 상승 에지(rising edge)에서 어서트(assert)되는 경우 원하는 동작이 실행되도록 설계를 하였다.

각 스테이트에 따른 발생되는 인에이블 신호가 그림 2에 표시되어 있다. 본 논문에서 사용하는 스테이트 머쉰은 ST0부터 ST18까지 총19개의 상태로 구성되는 유한 스테이트 머쉰(finite state machine; FSM)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. 프로그램 카운터는 프로그램의 수행을 제어하는 카운터로서 명령어를 페치할 번지를 가리키고 특별한 경우가 아닌 경우 자동으로 명령어의 길이만큼 증가되어 다음 번지를 가리킬 준비

를 한다.

ROM은 이름 그대로 명령어와 데이터를 읽어 올 수 있도록 각 주소에 명령코드(OP 코드)와 피연산자(operand)를 저장하고 있다. 입력 주소는 16bit이고 출력 데이터는 8비트이다. ROM은 FPGA 외부에 연결할 수도 있으나 ROM은 기본적으로 민텀의 합(sum of minterms)으로 표현되는 조합논리이므로 본 논문에서는 FPGA 내부에서 구현하였다.

명령어 레지스터/디코더는 ROM의 출력을 받아서 이를 저장하고 해독하여 각 명령어 수행 시 각 블록이 그 명령어에 맞게 수행되도록 내보내는 역할을 한다.

Acc는 누산기이며 R0~R7은 범용 레지스터이다. P_IN은 외부에서 입력하는 값을 저장하는 레지스터이고 P_OUT은 Acc의 값을 받아서 출력으로 내보낸다.

산술 논리 연산장치(ALU)는 명령어 레지스터/디코더로부터 명령어를 받아 각 명령어에 맞도록 Acc, R0~R7으로부터 받은 값을 산술 또는 논리 연산을 하는 역할과 인터럽트 시 스택에 저장된 값을 받아 내보내는 역할을 한다.

스택은 인터럽트 발생 시 현재의 Acc 값과 R0~R7 값, 주소를 저장하는 레지스터이다. 인터럽트가 끝난 후 다시 인터럽트 발생 전의 프로그램으로 돌아갈 때 스택에서 복귀주소를 불러오며 인터럽트 실행 전 프로그램의 Acc 값과 R0~R7 값도 불러와 도중에 중단된 프로그램을 계속 실행해주는 역할을 한다.

표 1. 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

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.

마이크로프로세서를 구성하는 각 블록은 VHDL로 작성했고 전체 블록도는 그래픽 디자인 포맷을 사용하여 스키매틱 형태로 구성하였다. 그림 3은 전체 블록도를 보여준다. DATA_IN 단자는 P_IN 포트의 입력 단자이고 DATA_OUT 단자는 P_OUT 포트의 출력 단자이다. RESET 핀은 마이크로프로세서를 리셋해주고 CLK은 마이크로프로세서를 동기적으로 구동하는 클락 신호 입력이며 INTERRUPT 핀은 외부 인터럽트 요청 입력이다.

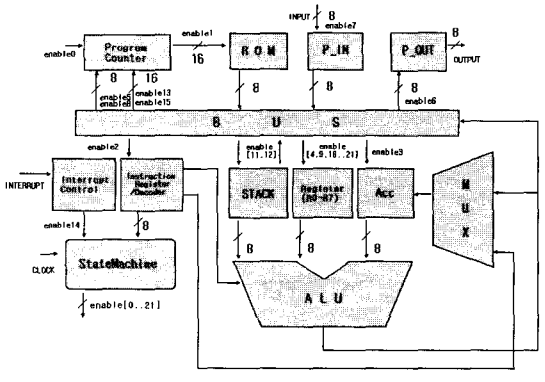


그림 1. 설계한 마이크로프로세서의 구조

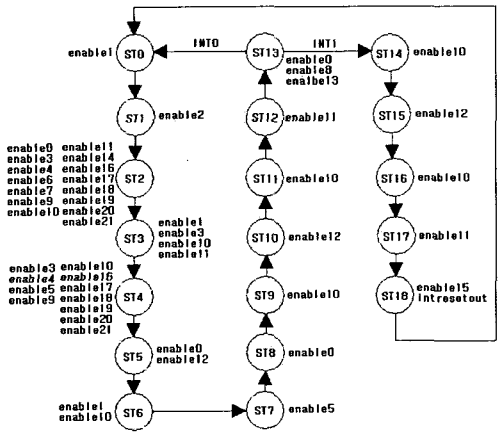


그림 2. 스테이트 머쉰의 상태도

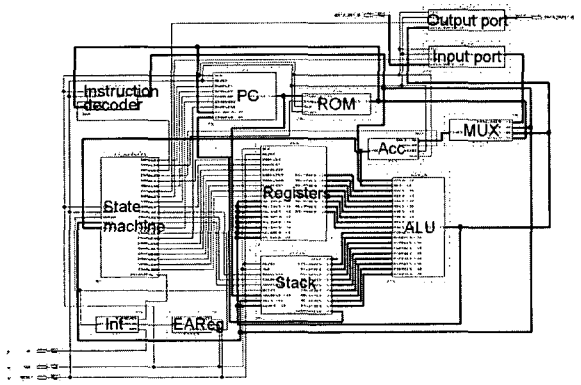


그림 3. 설계한 마이크로프로세서의 전체 블록도

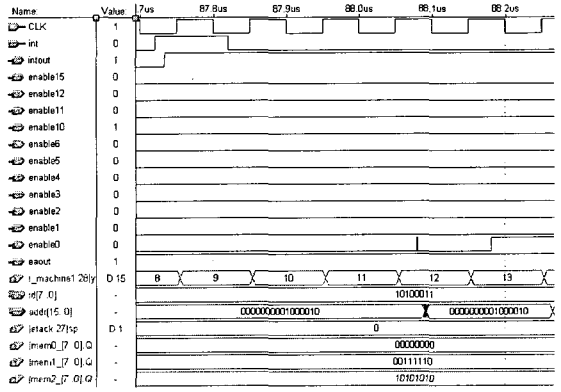
3. 시뮬레이션

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.

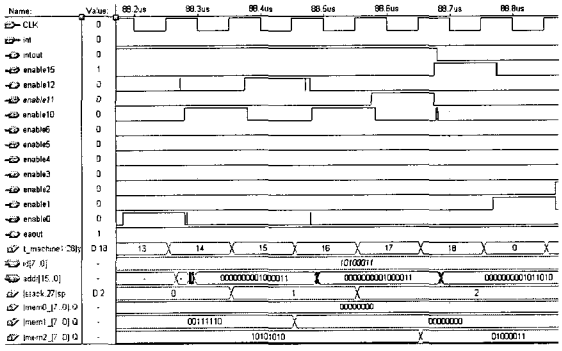
표 2는 스테이트 머쉰의 각 상태에서 인에이블되는 신호를 보여준다. 그림 4(a)와 (b)의 두 파형은 시간적으로 연결되는 파형이다. 그림에서 인터럽트(int)가 들어오면 인터럽트 인지 신호(intout)는 상태가 ST18로 되기 전까지 유지됨을 알 수 있다. 이는 ST13에서의 판별뿐만 아니라 그 이후에 인터럽트를 실행하기 위한 현재의 주소를 스택에 저장하는 과정에서 인터럽트 신호를 판별하기 위한 신호이다. 인터럽트가 들어오면 ST13에서 그 신호의 여부를 판별하여 인터럽트 신호가 들어왔을 경우, ST14~ST18의 상태가 진행된다. 상태 ST14~ST17은 현재의 주소를 스택에 저장하기 위한 것이며 ST18은 인터럽트 요청 시 미리 정해놓은 주소(인터럽트 벡터)로 가도록 해준다.

표 2. 상태에 따라 어서트되는 신호

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



(a)



(b)

그림 4. 인터럽트 요청 시 타이밍 다이어그램: (a) 다음에 (b)가 시간적으로 연결된다

명령어 각각에 대해 제대로 동작함을 확인한 후, 다양한 어셈블리 언어 프로그램을 작성하여 동작을 확인하였다. 이때 클락 주파수는 1MHz로 하여 시뮬레이션을 수행하였다.

4. 실 험

사용한 FPGA 칩은 ACEX1K 계열의 칩인 EP1K50TC144-3이다. 이 칩에는 총2880개의 로직 엘리먼트(logic element)가 있다. 게이트 수로 환산하면 약 5만 개에 해당한다.

그림 5는 실험을 위해 제작한 프로토타입보드를 보여준다. 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. 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. 입력 데이터 핀 8개, 출력 데이터 핀 8개, 클락, 리셋, 인터럽트 핀 각 1개씩해서 모두 19개이다.

크리스털 발진기로 발생시킨 16MHz를 분주하여 1MHz로 만든 후 FPGA 칩의 GCLK(global clock) 단자로 입력해주었다.

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

어셈블리 언어 프로그램을 메인 프로그램과 인터럽트 서비스 루틴으로 구성해 LED 제어를 해보았다. 설계한 마이크로프로세서가 외부의 인터럽트 요청을 제대로 처리해주는지를 확인하기 위한 것이다. 메인 프로그램은 숫자 255까지 순서대로 카운트하도록 되어있고, 외부 인터럽트 요청이 있는 경우 실행되는 인터럽트 서비스 루틴은 한 비트씩 왼쪽으로 쉬프트하도록 만들었다. 2번 반복 후에 메인 프로그램으로 복귀하도록 했다. 복귀를 하면 인터럽트가 요청된 시점의 다음 숫자부터 다시 카운트하게 된다. 실험 결과 제대로 동작함을 확인할 수 있었다.

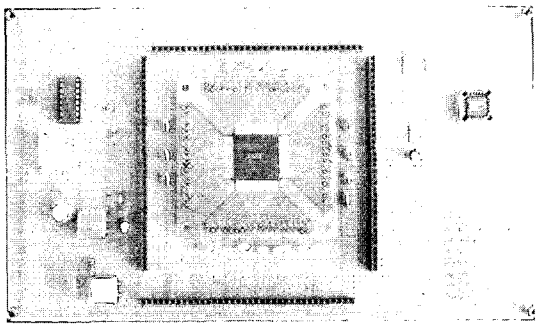


그림 5. 단일 FPGA로 구현한 마이크로프로세서

5. 결 론

본 논문에서는 기본적인 명령어 세트를 구현하는 실용적 8비트 마이크로프로세서를 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 있었다. 따라서 본 논문에서는 제어에 필요한 명령어를 갖는 실용적 마이크로프로세서를 처음부터 끝까지 설계와 구현하는 방법 및 시뮬레이션과 제작 결과를 체계적으로 제시하였다.

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

마이크로프로세서를 필요로 하는 다른 시스템의 설계에서도 IP로 넣어 활용할 수도 있을 것으로 전망된다. 추후 스테이트 머쉰을 구성하는 상태 수를 줄일 필요가 있으며 회로 구조 등에서도 최적화를 더 시킬 필요가 있다.

참 고 문 헌

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

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

[3] 송호정, 송기용, "HDL을 이용한 간략형 8-Bit 프로세서의 설계," 한국신호처리시스템학회 추계학술대회, 2000, pp. 241-244.

[4] 박두열, "VHDL을 이용한 Parwan CPU의 Modeling과 Design," 한국컴퓨터정보학회 논문지, 7권, 2호, pp. 19-33, 2002년 6월.

[5] Z. Navabi, VHDL: Analysis and Design of Digital Systems, McGraw-Hill, 1998.

[6] 박두열, "VHDL을 이용한 파이프라인 SIC의 시뮬레이션," 한국컴퓨터정보학회 논문지, 8권, 2호, pp. 24-32, 2001년 6월.

[7] 조경연, "16 비트 EISC 마이크로 프로세서에 관한 연구," 멀티미디어학회 논문지, 3권, 2호, pp. 192-200, 2000년 4월.

[8] 조경연. "확장 명령어 32 비트 마이크로 프로세서에 관한 연구," 대한전자공학회 논문지, 36권, D편, 5호, pp. 391-400, 1999년 5월.

[9] Altera Corp., http://www.altera.com.

[10] Atmel Corp., http://www.atmel.com.

1D. 네트워크 프로토콜