2001년도 대한전기학회/대한전자공학회 시스템및제어분야 합동 추계학술대회 논문집 (2001.11.24)

Microprocessor Embedded 2-Axis Motor Control Chip의 설계

노 규 진, 최 성 혁, 원 종 백, 김 종 은, 박 종 식
경북대학교 전자공학과
전화 : 053-940-8839 / 핸드폰 : 017-816-0327

Design of Microprocessor Embedded 2-Axis Motor Control Chip

Kyu Jin Roh, Sung Hyuk Choi, Jong Baek Won, Jong Eun Kim, Jong Sik Park
Department of Electronics Engineering, Kyungpook National University.
E-mail : burnice@hanmail.net

Abstract - In this paper we designed CAMC-SP, the microprocessor embedded 2-axis motor control chip which controls a precise pulse motor by generating the pulse needed to control step motor, DC servo and AC servo motor. This design enables to decrease costs and to minimize a size.

First we designed risc type 8-bit microprocessor compatible with PIC16C84, second we designed pulse motor controller. CAMC-SP is integrated of those two block.

We designed CAMC-SP by VHDL and we testified to the performance of it by performing functional simulation.

1. 서론

실시간 고속 정밀 제어를 위한 펄스 모터 제어 전용칩은 외부의 마이크로프로세서를 이용한 제어가 불가피하다. 펄스 모터 제어 칩과 마이크로프로세서의 코어를 설계하여 one chip으로 집적하여 설계하면 별도의 복잡한 프로세싱 시스템의 구축 없이 stand alone의 특징으로 동작이 가능하다. 이러한 설계는 저가, 소형의 정밀 모터 제어 시스템의 구현을 가능하게 한다. 설계된 칩은 RISC 구조를 갖는 8-bit 마이크로프로세서인 PIC16C84와 호환을 갖는 프로세서 부분과 2축 동작이 가능한 모터 제어 부분으로 구성된다.

본 논문에서 구현한 마이크로프로세서 내장 정밀 모터 제어 칩은 5 Mpps(pulse per second)의 고속 펄스를 출력할 수 있으며, 8가지 출력 펄스를 선택할 수 있다. 외부 센서 신호에 의한 급정지 및 감속 정지가 가능하다. 제어 대상은 step 모터와 서보 모터이며, 모터의 가·감속시 모터에 가해지는 기계적 부담을 감소시킬 수 있다. 또한 펄스 출력 도중 속도 변경 및 이동량 변경이 가능하다.

2. 전체 설계의 개요

본 설계는 Process부와 Motion부가 단일칩 안에 내장된 저소비전력, 고속의 완전 static의 CMOS system이다. 그림 1은 전체 회로의 블록도이다.

마이크로프로세서 내장 모터 제어칩은 크게 Process부와 Motion부로 구성된다. Motion부의 동작을 위한 범용 입출력 신호들을 제외한 주요 data input/output과 address input은 Process부에서 제어하여 동작하며, 사용자는 Process부의 입출력 interface와 내부에 구성된

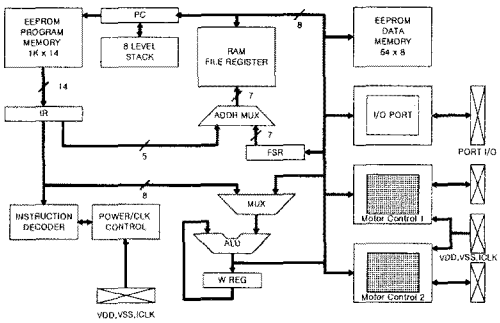

그림 1. 전체 회로의 블록도
Fig. 1. Block diagram of chip

EEPROM program memory를 access함으로써 Motion부를 control 할 수 있다.

Process부는 Von Neumann Architecture 대신 Harvard Architecture를 사용하는데, 이는 데이터와 명령을 위한 분리된 버스와 함께 메모리의 레지스터 파일 개념에 그 기초를 두고 있으며, 8bit-wide data bus와 메모리(RAM), 그리고 14bit-wide program bus와 프로그램 메모리(EPROM)를 구성한다.

이 아키텍처로 고속으로 명령의 fetch와 실행 사이클이 중첩되는 상태로 고속으로 비트, 바이트, 레지스터 연산을 할 수 있는 강력한 명령 세트들을 간단한 형태로 설계될 수 있다 즉, 명령이 실행되고, 다음 명령은 동시에 프로그램 메모리로부터 읽혀지는 것과 같은 중첩 처리가 가능한 것이다.

8비트의 ALU는 하나의 Working 레지스터(W 레지스터)를 가지고 있다. ALU는 W 레지스터에 있는, 데이터와 다른 어떤 파일 레지스터간의 산술연산 또는 논리연산을 수행한다. 또한 단일 오퍼랜드 연산을 W레지스터 또는 다른 어떠한 file register와도 수행할 수 있다.

Process부는 Motion부와의 data 입출력을 위하여 Motion부 내의 두 motor control blcok에 대칭되는 레지스터를 두 개씩 할당하여 address와 data를 각각 설정하는데 read enable신호와 write enable신호에 따라 data 전송이 가능하게 된다.

입력된 Clock은 Motion부로는 그대로 전달되고, Process부로는 Q1, Q2, Q3, Q4로 이름 붙여진 4개의 중첩되지 않은 clock으로 나뉜다. 내부에서 Program counter는 Q1시작점에서 increment되고, instruction은 Q4 이내에 프로그램 메모리로부터 fetch되고, instruction register에 latch된다. 즉, instruction은 Q1에서 Q4 사이에 순차적으로 번역되고 실행된다.

3. Process부의 구현

Process부의 구성은 ALU block, register block, control block으로 구분된다. 그 블록도는 그림 2와 같다.

Control block은 크게 명령어 디코딩과 프로그램 카운터의 두 가지 기능을 수행한다. 내장된 ROM에서 인출된 명령어는 명령어 디코딩 블록에서 해독된다. 모든 명령 형식에서 data operand의 주소 지정방식은 직접 addressing mode로 규정되어 있다. 그러나 INDF register(address = 00h)를 참조하는 경우 FSR register 간접 addressing을 수행한다. 실제로 INDF register는 존재하지 않는다. 다음의 두 명령을 비교해보면, 형식상 두 명령 모두 data는 직접 addressing이다. 그러나 00h번지를 참조하는 경우 fsr(4:0)번지의 register가 clear된다.

Program counter 과정을 살펴보면 다음과 같다. Microprocessor의 동작은 계속적인 명령 읽기와 decoding의 연속이라 볼 수 있다. 명령을 읽어 IR에 저장하는 과정이 fetch 동작이다. 일단 fetch에 의해서 명령이 IR에 읽혀지면 명령 decoding과 decode-execute는 clock에 의해서 진행된 뿐이다. RISC형태의 processor에서 명령 fetch를 위한 program counter의 동작은 매우 단순하다. 모든 명령이 동일한 크기의 1개 word로 이루어져 있기 때문에 매번 명령이 읽혀질 때마다 pc를 1씩 증가시키면 된다. 실행중인 명령이 몇 개의 word로 구성되어 있는지 확인하기 위해서 decoding의 결과를 기다릴 필요가 없으므로 명령 실행 cycle은 2단계(fetch와 decode-execute)로 끝난다.
결국 decode-execute하는 동안 미리 명령을 읽어 오도록 함으로서 명령 수행 cycle을 단축할 수 있게 된다.

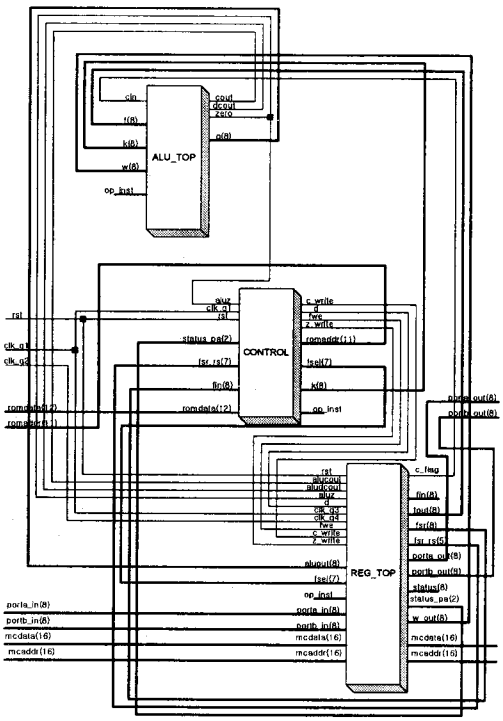

그림 2. Process부의 블록도
Fig. 2. Block diagram of process part

명령의 수행 및 연산의 결과는 Register에 저장된다. Process부에는 ALU연산을 위한 W register와 데이터 메모리로 사용하는 Register file이 있다. Register file는 크게 Special register와 General register 두 부분으로 분류할 수 있다. General register는 사용자의 프로그래밍에 따라 저장장소로 쓸 수 있는 일반적인 저장장소이며 special register는 프로세서의 상태를 표시하는 status register를 비롯하여 io-port 입출력을 위한 porta와 portb register, 그리고 이 양방향 버스 제어를 위한 tri-state buffer 제어 register인 trisa와 trisb register등 총 32개의 특수 용도의 register를 포함한다. 이 Special register중 6개는 motion control을 위해 할당된다.

ALU block은 기본적인 산술연산이나 boolean연산을 수행하기 위해 register file과 W register의 data와 관여한다. 즉 ALU는 8-bit data와 working register로 구성된다. 따라서 ALU block은 피연산 dat를 제공하는 두 개의 mux부분과 opcode를 생성하는 function encoding 부분, 연산을 수행하는 operation부분으로 나뉜다.

ALU는 두 입력의 mux와 논리연산의 function block으로 설계한다. 명령에 의한 연산 수행은 W register와 data memory의 직접 addressing에 의하여 이루어진다. 이를 위하여 fin/fout, win/wout, k 등 3개의 8bit 내부 버스를 둔다.

4. Motion부의 구현

Motion부는 크게 두 개의 동일한 motion control block으로 구분할 수 있다. 이 두 block은 서로 독립적으로 동작하며 이는 모두 process부에 의해 제어된다. 두 motion block에서의 출력 신호는 2차원 평면 운동을 위한 2축 모터 제어를 위하여 x축, y축에 각각 연결된 모터에 신호를 인가함으로써 사용되거나 두 개의 독립적인 모터 제어 시스템을 각각 제어하는 용도로 사용될 수 있다.
하나의 motion block은 bus control, registers, pulse generate module, drive stop control 등으로 구성되어 있다.
다음은 motion block의 블록도이다.

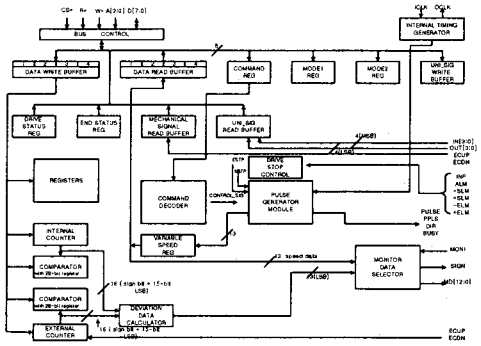

그림 3. Motion block의 블록도
Fig. 3. Block diagram of motion block

4-1 펄스 생성 알고리즘

모터의 제어에 필요한 펄스를 프로그램된 속도로 출력하기 위해서 펄스 발생 알고리즘을 개발하였다. 일정한 속도를 시간에 대하여 적분하면 그 값은 발생된 펄스의 수와 같다.

$$n(0:t) = \int_t^0 v(t')dt'$$

여기서 n(0:t)는 0에서 t시간까지에 발생된 펄스의 합이 되며, v(t')는 시간 t'에서 발생된 펄스의 속도를 나타낸다. v를 목표속도와 주파수 단위의 곱으로 나타내면 다음과 같다.

$$v = OBJ - Vp * Funit$$

Motion부는 출력주파수 설정 단위 파라미터로써 range data(0001h-1FFFh)가 있고, 출력주파수 파라미터로써 출발/정지 스피드 data (0001h- 1FFFh), 목표 스피드 data (0001h-1FFFh) 로 스피드 설정 기능이 있다. 입력 기준 clock은 최대 주파수 20Hz이고 출력 주파수의 단위는 pps(pulse per second)이다.

각 파라미터와 출력주파수와의 관계는 다음과 같다.

$$Funit = \frac{Fclk}{range\ data * 32,768}$$

$Fstsp = Funit * start/stop\ speed\ data$:

$Fobj = Funit * object\ speed\ data$: 목표 주파수

$Fout = Funit * speed\ data$: 동작 주파수

이상 식에서 Funit은 출력주파수 설정 단위이다. Range data 값을 변경하여 출력 주파수의 속도를 변경할 수 있다. Funit에 speed data를 곱하면 현재 동작 주파수 값을 계산할 수 있다. Fstsp는 Funit에 출발/정지 speed data를 곱한 출발/정지 주파수이고, Fobj는 Funit에 목표 speed data를 곱한 목표 주파수 값이다. Funit, Fstsp, Fobj, Fout의 단위는 모두 pps이다.

가감속 시간을 설정하는 파라미터인 rate-1 data(0001h-1FFFh), rate-2 data (0001h-1FFFh), rate-3 data(0001h-1FFFh)가 있어서 speed 가감속 시간을 설정하는 기능이 있다. 3개의 rate data 값을 각각 따로 설정하여 S자 drive 시 3개의 구간을 서로 다른 가감속율을 가진 drive를 실행할 수 있다. 입력 기준 clock은 최대 20MHz이다. Tunit은 가감속 시간 설정단위이며 단위는 초(sec)이다. Tud는 가감속 시간이며 단위는 초(sec)이다. Object speed data는 목표 speed data이며 start/stop speed data는 출발/정지 speed data이다.

이들 data와 가감속 시간과의 관계는 다음과 같다.

$$Tunit = \frac{rate - n\ data * 8}{Fclk}$$

$$Tud = Tunit * (ABS\ [obj - std])$$

Preset pulse drive 시에 감속 개시 포인트 검출 방식을 선택할 수 있다. Model register의 7 bit 값이 '0'이면 자동 검출 방식, '1'이면 나머지 펄스 수 지정 방식으로 감속을 개시한다.

자동 검출 방식은 감속 후 start/stop(출발/정지) speed data로 slow down/rear data 만큼의 펄스가 출력되고 drive를 정지한다.

5. 시뮬레이션 결과

Motion 동작의 주요 알고리즘은 VHDL coding 이전에 C program으로 알고리즘 구현을 하여 검증을 하였다. 결과 data를 추출하여 도표화하여 각 mode별로 동작을 확인하였다.

Rate-1, rate-2, rate-3 값을 각각 다르게 주어 가감속 시에 보다 기계적인 무리를 줄일 수 있는데 이러한 Quasi-S type의 대칭적 동작을 테스트로 검증하였다. 다음은 그 검증 결과이다.

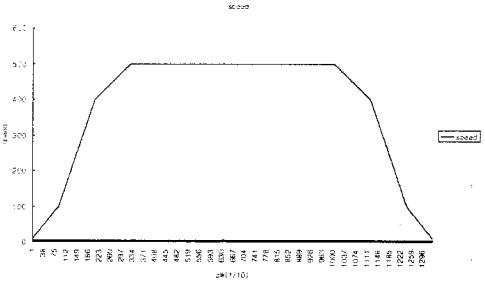

그림 4 Quasi-S형의 시뮬레이션 결과
Fig. 4. Result of Quasi-S simulation

VHDL설계 후 function simulation을 하였다.

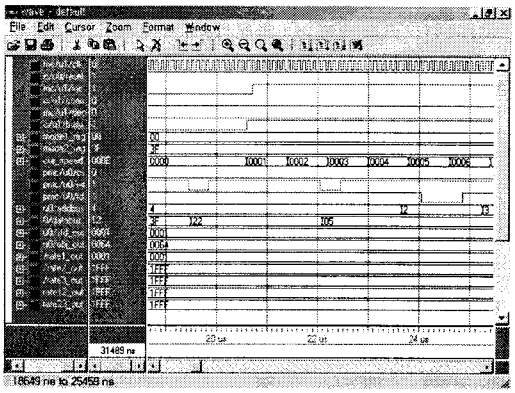

그림 5. Motion 동작 시점의 결과
Fig. 6-5 Result of motion drive start

이상 결과를 살펴보면 process부에서 motion부에 start명령(22h)을 내린 시점 다음 동기에서 busy_out이 '1'로 선언되면서 motion부는 속도 증가 동작을 시작하게 된다. Speed 값은 process부에서 sampling하여 읽어들여 범용 port로 출력하도록 프로그래밍 하였다. 목표 속도는 64h(100)으로 두었고, 시작/종료 속도는 1로 하였다.

그림 6은 목표속도에 도달 후 등속운동을 시작하는 시점의 파형이다. 주어진 목표 속도 64h에 도달하게 되면 정지 명령이 입력될 때까지 최대 속도인 64h로 등속운동을 시작하게 된다. 이상에서 보면 cur_speed가 지속적으로 증가하다가 목표속도에 이르는 순간 inc 신호는 1에서 0으로, cons 신호는 0에서 1로 변이함을 알 수 있고, 더 이상의 cur_speed의 변화는 일어나지 않음을 알 수 있다.

5. 결론

본 논문에서는 고속 정밀 메카트로닉스 장비 등에 요구되는 정밀 모터 제어 칩의 Stand alone 동작을 위하여 Microprocessor를 내장시켜 단일칩으로 설계하였다. 이는 기존 방식의 외부 CPU 제어시 발생하는 고가 및 대형화의 단점을 보완하고 사용자의 프로그래밍 과정의 편의를 도왔다.

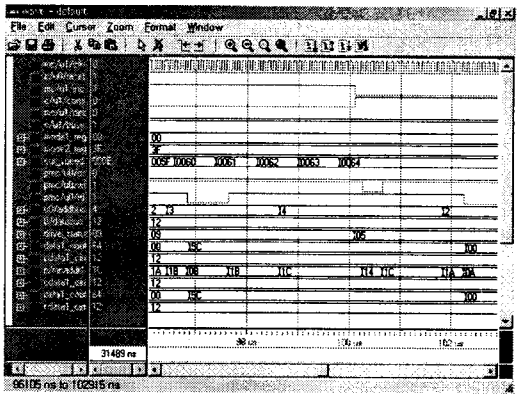

그림 6-6 등속운동 시작 시점의 파형
Fig. 6-6 Waveform of constant speed

본 설계의 전체 회로는 VHDL로 설계하였다. 설계 시 시스템은 PC 환경의 Modelsim tool을 이용하여 function을 검증하였고, Xilinx FPGA를 이용하여 Hardware Test를 하였다.

본 논문에서는 두 개의 개별적인 모터 제어 출력 신호를 갖는 구조로 프로그래밍하여 검증을 하였다. 이는 독립적인 두 개의 모터를 각각 제어할 수 있으며 x, y축에 각각 연결하여 평면 제어가 가능한 모터 시스템을 구현할 수도 있다.

본 논문에서 설계된 칩을 정밀 모터 제어 시스템에 사용하게 되면 사용자는 간단한 프로그래밍만으로 원하는 동작을 자유롭게 조절할 수 있으며, 별도의 복잡한 장비 없이 저가, 소형의 회로로 기존의 시스템의 기능을 모두 얻을 수 있다. 또한 자체적으로 범용 입출력 포트를 가지고 있으므로 다양한 동작을 요구하는 특수한 상황에서 복잡한 시스템 구조를 모두 제어할 수 있다는 특징을 가진다.

[참 고 문 헌]

[1] K. C. Chang, Digital Systems Design with VHDL and Synthesis, COMPUTER SOCIETY, 1999

[2] Ben Kohen, VHDL Coding styles and Methodologies, Second Edition, KLUWER ACADEMIC PUBLISHERS, 1999

[3] (V)DHL Compiler Reference Manual Version 3.4a, Synopsis, 1996

[4] VHDL for ASIC Synthesizer User Guide, COMPASS Design Automation, 1994

[5] Design Compiler Tutorial Version 3.4, Synopsis, 1996

[6] Neil H. E. Weste AND Kamran Eshrghian, PRINCIPLES OF CMOS VLSI DESIGN A Systems Perspective Second Edition, ADDISON-WESLEY PUBLISHING COMPANY, 1993

[8] Wayne Wolf, Modern VLSI Design A SYSTEMS APPROACH, PTR Prentice Hall, 1994