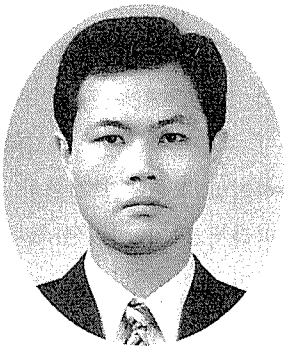
논단 II

DSP(Digital Signal Processor)의 개념 (下)



국제전기(주) 부설연구소
연구원 곽 철 훈

1. TMS320C25 DSP 보드의 설계와 응용

현재, DSP가 내장된 제품이 시장 확대 되고 있으며, 여기서는 제2세대 DSP인 TMS320C25와 고속 대용량 메모리 및 A-D/D-A 변환회로 등을 탑재한 보드의 하드웨어와 소프트웨어, 애플리케이션의 설계 방법를 소개한다.

1.1 DSP 보드의 설계

1) 설계한 보드의 특징과 개요

- 프로그램 메모리 공간 64K워드와 데이터 메모리 공간 64K워드를 실장하고 있다. 또 노웨이트에서의 메모리 액세스가 가능하다.
- DSP를 정지시키지 않고 퍼스널컴퓨터와 데이터를 수수할 수 있으므로 연속적인 계산처리 등이 가능하다.
- I/O로서는 비교적 용도가 많은 A-D 및 D-A 컨버터를 보드상에 탑재하고 있다.
- 유저가 자작하여 I/O 보드에 부착할 수 있도록 외부 확장 커넥터를 설치하고 있다.
- 개발 tool용, 평가용 등의 경우에는 퍼스널컴퓨터의 확장 슬롯에 장착하여 애드온 보드로, 장치에 내장하는 경우에는 스탠드얼론 보드로 사용할 수 있다.

1.2 DSP 보드의 회로 구성

1) 컨트롤 회로

회로는 퍼스널컴퓨터(여기서는 NEC사의 PC-9801)로부터 DSP에 대한 지령, 메모리 뱅크를 전환한다. 먼저 DSP에 대한 지령은 DSP의 기동, 정지 또는 HOLD 요구 및 그 해제 등이다. 퍼스널컴퓨터로부터 I/O 어드레서 0D0H, 또는 02D0H에 OUT 명령을 실행함으로써 DSP는 리셋 상태로 된다. 01D0H또는 03D0H에서 DSP에 HOLD를 요구, 01D2H 또는 03D2H에서 해제로 된다.

메모리 뱅크의 전환은 노웨이트로 64K워드라고 하는 용량을 만족시키기 위해 1개로 32768×3비트의 용량과 25ns의 액세스 타임을 가진 SRAM (MB8287-25)를 8개, 연속적인 처리를 할 수 있는 듀얼 포트 RAM 방식용으로 포트 아어비트레이션(port arbitration)이 부가된 SRAM(MB8422-

90)을 2개 사용한 회로를 나타낸다. 여기서 듀얼 포트 RAM 부분을 DSP가 액세스했을 때에는 RAM의 액세스 타임이 90ns이기 때문에 1웨이트가 삽입된다. 같은 이유로 ROM을 액세스했을 때에도 1웨이트 삽입된다.

듀얼 포트 RAM이 DSP의 메모리 어드레서 공간에서는 0800H에, PC9801의 물리 어드레스 공간에서는 C1000H로 중간적인 어드레스에 존재하는 이유는 TMS320C25에서는 메모리 어드레스 0~1024는 온칩 메모리를 액세스해 버리기 때문에 이것을 피하기 위함이다. 그리고 많은 메모리 용량을 필요로 경우에는 외부 커넥터를 경유하여 증설할 수 있다.

2) A-D, D-A 컨버터 회로

A-D 컨버터 회로는 발전기를 교환함으로써 A-D 변환속도를 3㎲, 5㎲, 10㎲로도 사용할 수 있다. 또 퍼스널컴퓨터의 ±12V 전원을 사용할 수 있다는 등의 이유로 AD7642(아날로그 디바이시스社 제품)을 사용하고 있다.

A-D 컨버터의 변환 요구, HOLD 요구 및 아날로그 스위치의 전환은 모두 소프트웨어 제어로 실시되고 있다. 변환신호나 종료신호는 DSP의 I/O 포트로부터 받아들인다는 점과 BIO 단자에 신호가 반영되고 있다는 등의 이유로 소프트웨어 처리가 편리하다.

앞으로 DSP를 사용한 디지털 신호처리 스스템의 개발이 점점 활발하게 진행될 것으로 생각한다. 이 어셈블러가 그 개발에 조금이나마 도움이 되었으면 한다.

1.3 TMS320C25 보드의 응용 예

이 TMS320C25 보드의 응용 예로서 2차의 IIR 밴드패스 필터를 소개한다. 밴드패스 필터의 계수는 중심주파수 f[Hz], 필터 특성의 Q, z변환의 z=1/q로 하여 다음 식으로 표현된다.

Fs : 샘플링 주파수

$w = 2\pi f/Fs$ $\quad a = \{1-(w/2q)^2$

$b = (1+a)\cos w$ $\quad b0 = (1-a)/^2$

$b1 = 0$ $\quad b2 = -b0$

$a1 = b$ $\quad a2 = -a$

라 하면,

$$y(q) \quad b0 + b_1 \cdot q + b_2 \cdot q^2$$
$$x(q) \quad 1 - a_1 \cdot q + a_2 \cdot q^2 \qquad (1.1)$$

이다. 상기의 전달함수는,

y[n]=현재의 출력값
x[n] = 현재의 입력값
y[n-1] = 하나 직전의 출력값
x[n-1] = 하나 직전의 입력값
y[n-2]=둘 직전의 출력값
x[n-2] = 둘 직전의 입력값

으로 하고 다음과 같은 적화식으로 표현할 수 있다.

$$y[n] = ary[n-1] + ary[n-2] + b0x[n] + b1x[n-1] + b2x[n-2] \qquad (1.2)$$

이 관계를 구하는 프로그램을 [리스트 1]에 나타낸다.

샘플링 주파수를 20㎑, 중심주파수를 1㎑, Q를 1000으로 했을 때의 계수는 다음과 같이 된다.

b_0=0.00015707
b_1 = 0.0
b_2=-0.00015707
a_1 = 1.90181427
a_2 =-0.99968587

〔리스트 1〕 필터의 계수를 구하는 프로그램

```
/x
        밴드패스 필터의 계수를 구한다
x/

#include        〈stdio.h〉
@Inciudc        〈math.h〉

/x
#define PI=3.14159265358979
x/

main()
{
double Fs, f, q, w, a, b, c, d, PI;
double a1, a2
double b0, b1, b2;
long wk;
   PI=3.14159265358979;
      printf("샘플링 주파수 ? ");
      scanf("xD", &wk);
      Fs=wk;
      printf(" 중심주파수  ? ");
      scanf("xD", &wk);
      f=wk;
      printf(" 필터 특성  Q  ? ");
      scanf("xD", &wk);
      q=wk;

      w=2*(double)PI*f/Fs;
      a=(1-w/(2*q);
      ax=a;
      b=(1+a)/*cos(w);
      b0=(1-a)/2;
      b1=0;
      b2--b0;
      a1=b;
      a2=-a;

      printf("\n")
      printf("b0 %12.8f\n", b0);
      printf("b1 %12.8f\n", b1);
      printf("b2 %12.8f\n", b2);
      printf("a1 %12.8f\n", b1);
      printf("a2 %12.8f\n", b2);
```

2. HD64180과 TMS320C25을 사용한 음진동 해석 시스템의 설계

DSP 시스템의 하드웨어와 소프트웨어를 소개한다. 이것은 모터 소리(진동)를 FFT 해석함으로써 설비의 이상을 검출하는 것이 목적이다. 시스템 전체의 제어는 호스트 CPU인 HD641480이 담당하고,FFT 연산은 DSP(TMS320C25)가 담당하고 있다. 또 A-D 변환부의 데이터 수납에는 호스트 CPU의 DMS 기능을 이용하고 있다.

2.1 시스템의 목적과 개요

음향 센서를 이용하여 모터나 터빈 등의 이상

유무를, 기기를 정지시키지 않고 감시 해석하기 위한 방법으로는 다음과 같다.

모터나 터빈 등의 기기는 정상적인 동작을 하고 있는 때와 이상이 있는 경우에는 발생하는 음에 대한 차이가 있다. 또 이상이 있는 경우에도 그 정도에 따라 발생하는 음도 미묘하게 나타내는 것을 경험적으로 알고 있다. 이 소자를 센서로 받아들여 신호의 특징을 추출하여 기기의 이상 유무와 그 정도를 판단하고 표시하는 것이 본 장치의 목적이다. 또한, 본 장치 내부에 있는 DSP는 신호의 특징을 추출할 때에 A-D컨버터로 받아들인 신호를 주파수 성분으로 변호나하는 FFT 처리를 위해 사용하고 있다.

2.2 하드웨어의 구성

본 장치는 CPU, DSP, I/O 등을 탑재한 메인 보드와 A-D 컨버터 보드 2채널분으로 합계 3장의 보드로 구성되어 있다.

1) CPU부

CPU는 8비트 타입의 HD64180을 사용하고 있으며, 본 장치에서 대부분의 제어를 담당하고 있다.

2) CPU용 메모리부

CPU용 메모리로 ROM 32K바이트, RAM 192K 바이트를 내장하고 있으며, ROM의 부트 프로그램에 의해 IC 카드에서 각종 프로그램을 RAM 상에 로딩하여 사용하고 있다.

3) DSP 인터페이스부

CPU와 DSP르 접속하기 위한 인터페이스부이며, CPU로부터는 DSP의 데이터 메모리가 CPU 메로리 탭의 일부로 보이는 것처럼 되어 있다.

4) DSP부

DSP는 TI사 제품인 TMS320C25를 사용하고 있으며 사이클 시간은 400ns로 원래의 성능에서 상당히 여유를 갖는 동작을 시키고 있다.(그래도 1024점의 FFT 처리는 수백 ms가 걸린다.)

DSP에서는 이번 프로그램의 노하우가 적었기 때문에 A-D 컨버터로 받아들인 데이터의 윈도우 처리, 데이터의 재배치, FFT 및 파워 스팩트럼 처리만 하기로 함으로써 프로그램 개발의 부담을 경감시키고 있다.

5) DSP 메모리부

DSP 메모리는 프로그램용과 데이터용으로 각각 32K 워드를 내장하고 있다.

6) A-D 컨트롤러부

CPU로부터의 기동신호에 의하여 A-D 보드에 대해 입력신호의 선택, 샘플 홀드, A-D 변환 개시 타이밍 등을 발생한다. 또 CPU에 대해서는 A-D 변환의 타이밍에 맞추어 DMA 요구선호를 발생하고, CPU의 DMA 기능에 의해 A-D 컨버터로부터의 데이터를 메모리에 수납한다.
이부분은 가운터와 ROMDP 의한 시퀀스 제어로 실현하고 있다.

7) A-D 컨버터부

A-D 컨버터는 12비트의 AD7572와 샘플 홀드용으로 AD585를 사용하고 있다. A-D 컨버터부는 메인 보드와 별도로 2채널분이 있으며, 메인 앰프의 유니폴러 출력을 그대로 단한 바이폴러 출력을 수납하는(FFT용 신호라 부른다) 경우가 있

으며, 양자를 아날로그 스위치로 전환하여 사용하고 있다.

먼저, 입력신호는 직류성분을 차단하고 교류신호로 한다. 이 시점에서의 입력신호는 ±의 양극성을 가지기 때문에 가산기에 의해 1/2 풀스케일 분(2.5V)을 가산하고, 1/2 풀스케일에서 ±로 움직이도록 한다. 이상으로 1/2 풀스케일을 중심으로 한 바이폴러 출력이 얻어지도록 하고 있다.

2.3 D-A 컨버터, A-D 컨버터와 DSP의 접속법

DSP는 디지털 신호처리의 전문이다. 신호가 디지털이라면 어떠한 처리라도 해치워 버린다. 그러나 우리들이 처리하고 싶은 정보는 처음부터 디지털 신호인가? 우리들이 살고 있는 세계는 기온, 기압, 소리 등 모두가 아날로그 값이며 DSP로 처리하는 데에는 디지털 값으로 변환해야 한다. 반대로 DSP가 처리한 결과를 우리들이 이해하는 데에는 오감으로 포착되는 신호로 변환할 필요가 있다. 이때 디지털 - 아날로그 변환기 (D-A 컨버터), 아날로그 - 디지털 변환기 (A-D 컨버터)이다.

2.4 DSP와 A-D / D-A 컨버터의 접속

DSP는 텍사스 인스트루먼트社 (이하 TI社)의 TMS320C5000 시리즈 (이하 C5000)를 든다. C5000트, 버퍼드 시리얼 포트, 시분할 다중 시리얼 포트, 16비트 패러렐 I/O 포트로 다양한 선택이 가능하다. TI社의 각종 DSP 평가 기판(DK)DMS 아날로그 인터페이스 서킷(AIC)이라는 A-D / D-A 컨버터를 탑재하고 있다. 그리고 DSK를 사용한 제작 기사에는 반드시라고 할 정도로 AIC에 관한 기재가 있다. 그러나 일반의 A-D/D-A 컨버터와 DSP와의 인터페이스는 AIC의 그것과는 다르다. 가장 큰 차이는 AIC는 자체의 '시계'를 지니고 있다는 점이다. AIC는 자체의 타이밍으로 DSP에 데이터의 송수신을 요구한다. 그에 비해 일반의 A-D / D-A 컨버터는 '시각'에 관하여 아무 것도 관여하고 있지 않다. 때문에 DSP와 AIC를 접속하는 경우에는 하드웨어나 소프트웨어의 설계가 달라진다.

3. TMS320C32 C 언어

3.1 특 징

- 표준의 ANSI C 규격을 따르므로 호환성 및 이식성이 우수하다.
- 효율적이고 컴팩트한 어셈블리 코드를 생성하는 최적화 기능을 가지고 있다.
- ANSI C 라이브러리 표준의 라이브러리 파일(rts30.lib, rts40.lib, rts.src)을 제공한다.
- C 소스 코드를 번역하여 어셈블리 코드로 변환한다.
- 컴파일, 어셈블, 링크를 일련적으로 수행하는 쉘 프로그램(CL30.EXE)을 함께 제공한다.
- COFF 형식의 오브젝트 파일은 디버거를 사용하여 소스 레벨의 디버깅을 가능하게 한다.
- 매우 우수한 인라인 어셈블 기능을 가진다.
- ROM을 위한 코드를 생성하는 유틸리티(HEX30.EXE)나 C 소스나 어셈블리 코드를 연결 시켜주는 유틸리티(CLIST.EXT)를 제공한다.
- 32비트(char, short, int, long, float, double) 및 40비트(long double) 길이의 데이터를 사용한다.

■ C 컴파일의 수행 및 링크의 과정을 일련적으로 수행

- 쉘 프로그램인 CL30.EXE를 사용하면 컴파일 - 배치파일로 작성(C.BAT)

C:\DSP3X\TMS-32>C TEST[Enter]

```
    echo off
if "%1" =="" goto error
if not "%2" =="" goto command

echo on
CL30 -g -as -v32 %1, C -z %1, CMD
echo off
if exist %1, OBJ DEL %1, OBJ
goto end

:command
echo on
CL30 -g -as -v32 %1, C -z %2
echo off
if exist %1, OBJ DEL %1, OBJ
goto end

:error
echo          ********************************************
echo          *          Syntzx  :      C sourecfile[.c]              *
  echo      *                  C sourecfile[.c] commandfile.cmd      *
echo          ********************************************
:end
```

3.2 C 컴파일러에서의 섹션 사용

○ 어셈블리 프로그램에서는 사용자가 .text, .data, .bss 및 기타 섹션의 정의함. 데이터를 .data 색션에 저장

○ C 언어 프로그램에서는 섹션들을 컴파일러가 자동적으로 생성 .data 섹션은 기본적으로 사용하지 않음. 데이터의 성격에 따라 .cinit, .const, .bss, .sysmem 섹션 등의 사용

〈표 3-1〉 C 컴파일러에 의하여 자동적으로 생성되는 섹션

섹 션	형 식	내 용
.text	initialized	실행 코드 또는 부동소숫점 상수를 포함하는 초기화 섹션이다.
.cinit	initialized	초기화된 글로별 변수 또는 static variable을 포함하는 초 화 섹션이다.
.const	initialized	부동소숫점 상수 또는 switch table을 포함하는 초기화 섹션이다.
.bss	uninitialized	글로벌 변수 또는 static variable을 포함하는 섹션으로서, 프로그램의 startup루틴이 실행될 때 데이터를 .cinit 섹션으로부터 .bss 섹션으로 복사한다.
.symem	uninitialized	malloc, caloc, realloc 등의 메모리 함수에 의하여 사용되는 데이터를 포함하는 섹션이다.
.data	initialized	기본적으로 사용하지 않는 섹션이다.
.stack	uninitialized	시스템 스택으로 사용되는 섹션이다.

3.3 ROM 모델과 RAM 모델

○ C 컴파일러는 컴파일러 할 때 전역 변수(global variable)들을 자동으로 초기화하기 위한 데이터 테이블을 생성

○ 이를 오브젝트 파일이 .cnint 섹션에 저장하여 두었다가 나중에 초기화될 때 .bss 섹션으로 복사

1) ROM 모델

- fldzjdptj -c 옵션(default)으로 지정
- 전역 변수들을 프로그램 실생시에(at runtime) 초기화
- 로더가 실행 프로그램을 메모리에 로드할 때 .cinit 섹션의 초기화 데이터 테이블을 만들어 두며, 프로그램이 실행될 때 루틴(boot routint) boot..obj에 의하여 이를 .bss 섹션으로 복사
- 메모리에 .cinit 섹션과 .bss 섹션이 모두 존재하므로 메모리 용량을 많이 차지하는 단점
- 실행파일내에 포함되어 있는 부트 루틴이 데이터를 초기화하여 주므로 특별한 틀이 필요

없음.

2) RAM 모델
- 링커에서 -cr 옵션으로 지정
- 전역 변수들을 프로그램 로드시에(at load time) 초기화
- 로더가 실행 프로그램을 메모리에 로드할 때 데이터의 초기화를 .bss 섹션에 직접 수행 오브젝트 파일에는 .cinit 섹션이 있으나 메모리에는 이로부터 초기화가 완료된 .bss 섹션만 남을
- 메모리 용량을 적게 사용하는 장점
- 로더가 이러한 초기화 기능을 가지고 있어야만 정상적인 처리가 수행

3.4 TMS320C32의 프로그래밍 테크닉

1) 정수 데이터의 제로 확장 및 부호확장
- 명령에서 하용되는 16비트의 즉치 데이터가 내부 레지스터에 저장될 때는 32비트로 확장됨

2) 제로 확장
- 놀리연산 명령이나 단일 반복명령 RPTS에서 사용
- 모두 양수로 간주되므로 상위 16비트가 0으로 채워짐(zero extended)
- 0000H~FFFFH(0~65535) 범위의 정수를 자유롭게 사용

예 LDI 00FFH,R0 : R0 ← 000000FFH (부호 확장)
AND 900FH,R0 : R0 ← R0, AND. 0000900FH (제로 확장)

```
        R0 =  0000 0000 0000 0000 0000 0000 1111 1111B
AND)    0000 0000 0000 0000 1001 0000 0000 1111B
-------------------------------------------------------
      0000 0000 0000 0000 0000 0000 0000 1111B
```

※ 논리연산에서 상위 16비트를 변경시키지 않고자 할 때 주의할 것

- 모든 논리연산에서 사용하는 즉시 데이터는 제로 확장되므로 상위 16비트가 0으로 됨
- 따라서 AND masking 기법을 사용하면 상위 16비트가 클리어되어 소실됨

예 AND900FH,R0 ; R0 ← R0 . AND. 0000900FH (제로 확장)

```
        R0 = XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXXB
AND)       0000 0000 0000 0000 1001 0000 0000 1111B
-------------------------------------------------------
      0000 0000 0000 0000 X00X 0000 0000 XXXXB
```

- 해결책은 AND 명령 대신에 ANDN 명령을 사용함

예 ANDN 6FF0H,R0 : R0 ← R0 . AND. 00006FF0H (제로 확장)

```
R0 = XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXXB
     0000 0000 0000 0000 0110 1111 1111 0000B
AND) 1111 1111 1111 1111 1001 0000 0000 1111B
-------------------------------------------------
     XXXX XXXX XXXX XXXX X00X 0000 0000 XXXXB
```

3) 부호 확장

- 정수 로드 명령이나 정수 산술연산 명령에서 사용
- 항상 2의 보수로 간주되어 최상위 비트의 부호를 고려하여 상위 16비트가 채워짐 (sign extended)
- 0000H~7FFFH(0~32767) 범위는 양수, 8000H~FFFFH(-32768~-1) 범위는 음수로 취급됨
- 32768 이상의 양수 범위에서 입력하고 나서 필요한만큼 비트 시프트시킴

예 LDI 0080H, R0 : R0 ← 00000080H (부호 확장)
LSH 8, R0 : R0 ← R0 FFFF9000H (부호 확장)

- 해결책은 16비트 즉시 데이터를 양수 범위에서 입력하고 나서 필요한만큼 비트 시프트시킴

예 LDI 0080H, R0 : R0 ← 00000080H (부호 확장)
LSH 8, R0 : R0 ← 00008000H
LDI 0090H, R1 : R1 ← 00000090H (부호 확장)
LSH 8, R1 : R1 ← 00009000H
ADD R1, R0 : R0 ← R0 + R1

※ 보조 레지스터에 I/O 번지를 지정할 때 주의할 것

- I/O 번지는 24비트이므로 이를 즉시 어드레싱으로 보조 레지스터에 로드할 수 없음.
- 일반적인 방법

예 .text

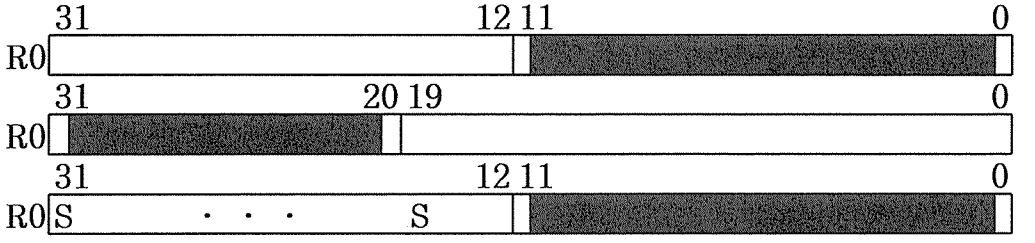
```
        LDI     @_8255_CS,AR0    : AR0 ← 810000H (Direct Addressing Mode)
        LDI     10000010B, R0    : R0 ← 00000082H
        STI     R0, *+AR0(3)     : (AR0 + 3) ← R0 (Indirect Addressing Mode)
        .data
_8255_CS.int 810000H             : 8255A PPI address
```

※ 2의 보수로 표현되는 12비트의 A/D 컨버터를 읽을 때 주의할 것

- A/D 컨버터에서는 최상위의 12번째 비트인 b11이 부호
- 읽혀져서 DSP 내부 레지스터에 저장되면 b31이 부호로 되어야 함

예 .text

```
        LDI     @_ADC_CS,AR0 : AR0 ← 812000H
        LDI     *AR0, R0     : R0 ← (AR0)(A/D 컨버터를 읽음)
        AND     03FFH,R0     : R0 ← R0 .AND. 03FFH (하위 12비트만을 취함)
        LSH     20, R0       : 왼쪽으로 12비트 논리적 이동 (부호비트 조정)
        ASH     -20, R0      : 다시 오른쪽으로 12비트 산술적 이동
        .data
ADC CS  .int 812000H         : A/D converter address
```



4) 시간지연 루틴의 작성

- 단일 반복명령 RPTS 또는 블록 반복명령 RPTB을 사용
- RPTS를 사용할 경우 지연시간동안 인터럽트가 허용되지 않는 문제 발생

```
                .text
D5
OUS:  RPTS 1245              : 50us delay at 500㎒ system clock(4+1245+1 cycle)
                NOP
                RETS

D5OUS1:         LDI 1245, R0     : 50us delay

      RPTS R0
                NOP
                RETS

D10MS:          RPTS @_Counter : 10ms delay at 50㎒ system clock(4+249995+1 cycle)
                NOP
                RETS
                  .
                  .
                  .
                .data
 Counter        .int 249995
```

〈그림 3-1〉 RPTS 명령을 사용한 반복처리 프로그램의 형식

```
D1US:           PUSH            R7
                LDI             1, R7           : delay 1 us
                BR              TIME1US
D5OUS:          PUSH            R7
                LDI             50, R7   : delay 50 us
                BR              TIME1US
D1MS:           PUSH            R7
                LDI             1000, R7: delay 1 ms
                BR              TIME1US
D10MS:          PUSH            R7
                LDI             10000, R7       : delay 10 ms
                BR              TIME1US
D100MS:         PUSH            R7
                LDI             10000, R7       : delay 100 ms
                BR              TIME1US
D1SEC:          PUSH            R7
                LDI             10000, R7       : delay 1 sec
                MPYI            100, R7
                BR              T

IME1US
                  .
                  .
                  .
TIME1US         RPTS            15              : (4
                NOP                             : + 16
                SUB1            1, R7           : + 1
                BNZ             TIME1US         : +4 = 25 cycle = 1 us at 50 ㎒)
                POP             R7
                RETS
```

〈그림 3-2〉 모든 문제가 해결된 시간지연 루틴

```
void c_int01(void)                    : INTO service routine
{           .
            .
            .
}
void c_int02(void)                    : INTO service routine
{           .
            .
            .
}
void c_int09(void)                    : TINTO service routine
{           .
            .
            .
}
void c_int10(void)                    : RINT1 service routine
{           .
            .
            .
}

asm("       .sect  \" .vector\" ")    : define section
asm("       .space 1            ")    : keep first vector empty
asm("       .word _c_int01      ")    : INT0 interrupt vector
asm("       .word _c_int02      ")    : INT1 interrupt vector
asm("       .space 5            ")    : keep unused interrupt vectors empty
asm("       .word _c_int09      ")    : TINT0 interrupt vector
asm("       .word _c_int10      ")    : TINT1 interrupt vector
                  .
                  .
void main(void)
{           .
            .
            .
}
```

〈그림 3-3〉 C 언어 프로그램에서 인터럽트 벡터 테이블 및 인터럽트 서비스 루틴을 작성하는 양식

5) ROM 프로그래밍

- HEX30.EXE를 사용하여 COFF 형식의 실행파일(XXX.OUT)을 Intel HEX 파일(XXX.HEX)로 변환

  ※ COFF(Common Object File Format)

  ※ 링커 옵션에서 -v0, -v1, -v2로 각각 COFF0, COFF1, COFF2 형식의 실행파일을 생성

- 커맨드 파일로 메모리 구성 형식이나 버스제어 레지스터값 등을 지정

  filename은 변화하고자 하는 COFF 형식의 실행파일(xxx.out) 또는 커맨드 파일(xxx.층)

HEX30 [-options] filename

```
/*****************************************************************/
/*      TMS-32 V1,0 : HEX30 Command File for Boot ROM Conversion  */
/*****************************************************************/
        test.out
        -o test.hex
        -i
        -datawidth 32           /* 프로그램에서의 논리적 데이터 길이 */
        -memwidth 8             /* ROM의 물리적인 데이터 길이 */
        -romwidth 8             /* ROM 소자 1개의 데이터 길이 */
        -boot
        -bootorg 001000h                /* 부팅 메모리의 시작번지 */
        -iostrb 000000c8h               /* 버스제어 레지스터 ISTRB의 값 */
        -strb0                          /* 버스제어 레지스터 STRB0의 값 */
        -strb1                          /* 버스제어 레지스터 STRB1의 값 */
```

4. 결론 DA/AD CONVERTER

A-D 컨버터, D-A 컨버터의 변화속도를 충분히 사용한 신호처리가 필요한 경우도 있는 것이다.

DSP는 그러한 요구에 부응하는 기능도 내장하고 있다. 예를 들면 버퍼드 시리얼 포트(BSP)나 다이렉트 메모리 액세스(DMA)등이다. 그것들을 사용하면 DSP의 퍼포먼스를 희생하는 일 없이 데이터의 송수신이 가능케 된다. 그러나 이를 위해서는 어느 정도의DSP의 지식, 기능이 필요하다.

먼저 DSP나 A-D / D-A 컨버터 등의 최신 정보를 입수하여 참조하는 것이 정확하게 움직이는 설계의 우선이다.

참고 문헌

1. Digital Signal Processing and the Microcontroller Dale Grover & John R. Deller
   Prentice Hall 1999
2. 마이크로프로세스 응용실습 (1988 자유아카데미)
3. C언어로 쉽게 쓰는 TMS320C31
   박귀태, 이상락
   고려대학교 전기공학과 자동제어연구실
4. 전자기술 98. 9호
5. DSP 마스터 시리즈 ①, ② TMS320C31, TMS320C32 마스터 TMS31, 32 키트 (1988, ohm사)
6. 초보자를 위한 DSP (TMS320C50)
   차영배
   다다미디어 1998
7. 디지탈 신호처리의 기초와 DSP 응용실무
   박선호
   동역메카트로닉스연구소
8. Digital Signal Processing with C and the TMS320C30
   Rulph Chassaing
   JOHN WILEY & SONS