특별연재(6-4)

# TMS320F2812의 리셋과 부팅

백 종 철
(주식회사 싱크웍스 이사)

당초 기획에서는 6-4편에서 C2000 계열을 이용한 고속 ADC에 대해서 다룰 예정이었으나, 많은 분들이 혼란을 겪고 있는 TMS320F2812 DSP를 대상으로 28X의 리셋과 부팅이 더 중요한 듯 해서 주제를 변경했다. 고속 ADC에 대한 내용은 당사 웹사이트(www.tms320.co.kr)에 제법 많은 내용이 올라 있기에 과감하게 주제를 바꾸는 용단을 내렸다. 28X의 리셋 과정부터 자세히 살펴보자.

## 1. 28X의 리셋과 인터럽트 벡터

TMS320C/F28X의 리셋 벡터는 0x3F FFC0에 고정되어 있다. 다시 말해, 28계열의 DSP 칩이 리셋 과정을 거치게 되면 Program Counter(이하 PC)에 0x3F FFC0가 로딩된다. 마이크로 컴퓨터 모드(이하 MC 모드) 로 운용될 경우, 0x3F FFC0는 칩 내부에 탑재된 Boot Rom을 가르킨다. 마이크로 프로세서 모드(이하 MP 모드)로 운용되면 Boot Rom 영역을 포함한 Zone7 영역이 외부 메모리로 맵핑된다. 그림 1은 TMS320C/F2812의 메모리 맵에서 Boot Rom 영역을 포함한 Zone 7 영역을 확대해서 그린 것이다.

그림 1에서 0x3F FFC0 ~ 0x3F FFFF 영역을 주의해서 보자. BROM Vector 라고 쓰여져 있다. 그리고, 32 X 32 되어 있다. 리셋을 포함한 32개의 인터럽트 벡터가 프로그램된 영역으로 각각의 벡터 크기는 32비트, 2word 라는 뜻이다.

그림 1에서 MC모드로 운용할 경우에는, 칩 내부에 탑재된 메모리가 사용된다고 표시하고있다. 그런데, BROM 영역은 개발자가 프로그래밍할 수 있는 영역이 아니다. 아니 세상에, 벡터를 개발자가 마음대로 못 집어 넣는다면, 어떻게 칩을 사용할 수 있단 말인가? 그렇다면, 무조건 MP 모드로 사용해야 한단 말인가? 여기서부터 혼란이 초래되는 듯 하다.

TI는 0x3F F000 ~ 0x3F FFFF 영역에 벡터를 포함해서 부트 로더(Boot Loader) 프로그램, 삼각 함수표 등을 마스킹하여 출하하고 있다. MC 모드로 사용한다면, TI가 마스킹한 벡터를 사용한다는 뜻이 된다. 먼저 Boot Rom의 구성을 보자.

그림 2에서, Math Tables and Functions 영역을 간단하게나마 살펴보자. 아주 유용한 내용인데, 여기저기 돌아다니면서 살펴 본 결과, TI가 마스킹한 내용이 있음에도 불구하고, 동일한 작업을 애써하는 경우도 보았으며, 활용을 제대로 못하는 경우도 더러 보았다. 활용도를 높이기 위해서, 특별히

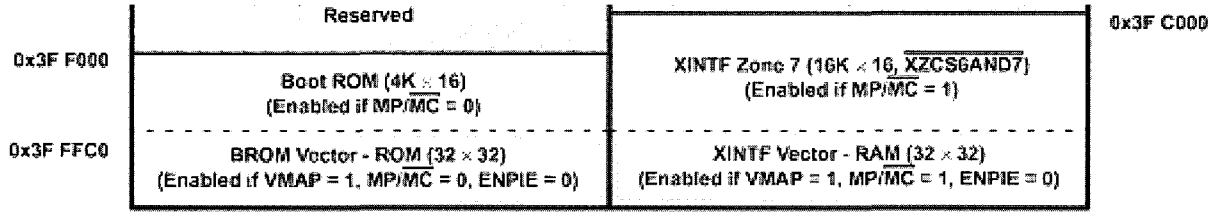


그림 1 TMS320C/F2812의 메모리 맵 - 벡터 영역을 포함한 Zone 7 영역

지면을 활용하여 언급하고자 한다. Math Tables and Functions 영역에는 활용빈도가 매우 높은 수학 함수 값들이 마스킹되어 있다. 그 내용은 그림 3과 같다.

그림 3에서 Sin/Cos 부분만 간단히 설명하고 넘어가도록 하자. 사인 1도부터 90도까지의 값을 0x3F F000영역부터 Q30의 정밀도로 기록되어있다. 정밀도 Q30의 데이터는

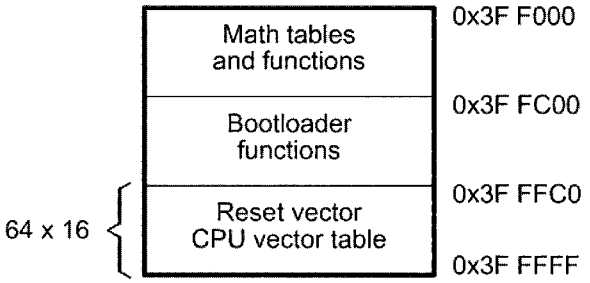


그림 2 Boot ROM의 구성

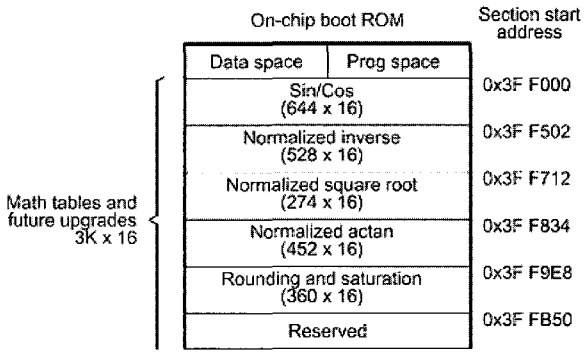


그림 3 TI가 마스킹한 수학 함수표

표 1 TMS320C/F2812에 마스킹된 벡터 코드

| Vector | Location in Boot ROM | Contents (ie points to) | Vector | Location in Boot ROM | Contents (ie points to) |
|---|---|---|---|---|---|
| RESET | 0x3F FFC0 | InitBoot (0x3F FC00) | RTOSINT | 0x3F FFE0 | 0x00 0060 |
| INT1 | 0x3F FFC2 | 0x00 0042 | Reserved | 0x3F FFE2 | 0x00 0062 |
| INT2 | 0x3F FFC4 | 0x00 0044 | NMI | 0x3F FFE4 | 0x00 0064 |
| INT3 | 0x3F FFC6 | 0x00 0046 | ILLEGAL | 0x3F FFE6 | 0x00 0066 |
| INT4 | 0x3F FFC8 | 0x00 0048 | USER1 | 0x3F FFE8 | 0x00 0068 |
| INT5 | 0x3F FFCA | 0x00 004A | USER2 | 0x3F FFEA | 0x00 006A |
| INT6 | 0x3F FFCC | 0x00 004C | USER3 | 0x3F FFEC | 0x00 006C |
| INT7 | 0x3F FFCE | 0x00 004E | USER4 | 0x3F FFEE | 0x00 006E |
| INT8 | 0x3F FFD0 | 0x00 0050 | USER5 | 0x3F FFF0 | 0x00 0070 |
| INT9 | 0x3F FFD2 | 0x00 0052 | USER6 | 0x3F FFF2 | 0x00 0072 |
| INT10 | 0x3F FFD4 | 0x00 0054 | USER7 | 0x3F FFF4 | 0x00 0074 |
| INT11 | 0x3F FFD6 | 0x00 0056 | USER8 | 0x3F FFF6 | 0x00 0076 |
| INT12 | 0x3F FFD8 | 0x00 0058 | USER9 | 0x3F FFF8 | 0x00 0078 |
| INT13 | 0x3F FFDA | 0x00 005A | USER10 | 0x3F FFFA | 0x00 007A |
| INT14 | 0x3F FFDC | 0x00 005C | USER11 | 0x3F FFFC | 0x00 007C |
| DLOGINT | 0x3F FFDE | 0x00 005E | USER12 | 0x3F FFFE | 0x00 007E |

Q15씩 나뉘어져 16비트 1 워드를 자리하고 있다. 연산을 하는데 매우 정밀한 삼각함수가 필요하다면 Q30, 즉 2 word를 불러와서 사용하면 된다. 16비트 프로세서에 사용하는 정도의 정밀도로도 충분하다면, 짝수 번지만 불러와서 사용하면 된다. 이런 이유로 Q30 정밀도의 데이터를 2 word로 쪼개 넣었다.

TI가 마스킹한 부트 로더 내용이 구현하고자 하는 어플리케이션과 맞지 않을 경우에는 MP모드로 운영하여 필요한 내용을 구현하면 된다. 하지만, TI의 부트로더 프로그램은 최대한의 범용성을 확보하고 있기에 이를 이용하는 편이 훨씬 편리한 경우가 많다. 그림 2에서는 보는 벡터 영역에 어떤 내용이 기록되어 있는지 자세히 들여다 보자.

표 1의 내용은 28X Core 가 가지고 있는 리셋과 코어 인터럽트와 일치한다. 리셋을 예를 들어 본다면, 28X 칩이 리셋이 되면 PC에 0x3F FFC0가 로딩된다고 했다. 0x3F FFC0에는 어떤 내용이 들어 있을까? 3번째 열을 보면, Contents라고 나온다. 그 내용은 Init Boot라고 나오는데, 이 내용은 TI가 마스킹한 Boot Loader 코드의 시작위치다. 끝 두 열은 TI 자체 테스트용이기에 크게 신경 쓸 필요 없다.

표 1을 보면서 이런 생각을 해보자. 28X의 인터럽트가 표 1에 있는 것 밖에 되지 않을까? 얼마나 많은 주변회로가 달려 있는데, 저것밖에 안된다면... 매우 불편할 것이다. 표 2는 TMS320C/F2812가 가지고 있는 벡터 전체를 표시한 표다. 확실히 많다.

표 2를 보는 방법은 다음과 같다. 표 1에서 본 INT1은 표 2의 좌측 첫 번째 행에 표시된 INT1이다. 즉 코어 인터럽트 1을 나타내다. 코어 인터럽트 1은 총 7개의 개별 인터럽트와 연결되어 있다.

예를 들어, PDPINTA가 발생했다면, 코어 입장에서는 INT1이 발생했다고 알아차린다. 이후 작업으로 어떤 것이 필요하겠는가? 그렇다. INT1을 발생시킨 원인을 분석해야 한다. 이를 위해서는 PIEIFR 레지스터와 PIEIER 레지스터를 분석하는 코드를 작성해야 한다. 코드가 필요하다는 얘기는 수행하는 시간이 걸린다는 얘기가 된다. 인터럽트는 속도가 생명이다. 특히, Hard Real Time System에서는 최고의 덕목이기까지 한데 인터럽트가 분석하는데 시간이 필요하다면 난감하지 않을 수 없다. 여기서 TMS320C/F2812의 메모리 맵 특정영역을 살펴보도록 하자.

표 2 TMS320C/F2812의 인터럽트 벡터

| | INTx.8 | INTx.7 | INTx.6 | INTx.5 | INTx.4 | INTx.3 | INTx.2 | INTx.1 |
|---|---|---|---|---|---|---|---|---|
| INT1 | WAKEINT | TINT0 | ADCINT | XINT2 | XINT1 | | PDPINTB | PDPINTA |
| INT2 | | T1OFINT | T1UFINT | T1CINT | T1PINT | CMP3INT | CMP2INT | CMP1INT |
| INT3 | | CAPINT3 | CAPINT2 | CAPINT1 | T2OFINT | T2UFINT | T2CINT | T2PINT |
| INT4 | | T3OFINT | T3UFINT | T3CINT | T3PINT | CMP6INT | CMP5INT | CMP4INT |
| INT5 | | CAPINT6 | CAPINT5 | CAPINT4 | T4OFINT | T4UFINT | T4CINT | T4PINT |
| INT6 | | | MXINT | MRINT | | | SPITXINTA | SPIRXINTA |
| INT7 | | | | | | | | |
| INT8 | | | | | | | | |
| INT9 | | | ECAN1INT | ECAN0INT | SCITXINTB | SCIRXINTB | SCITXINTA | SCIRXINTA |
| INT10 | | | | | | | | |
| INT11 | | | | | | | | |
| INT12 | | | | | | | | |

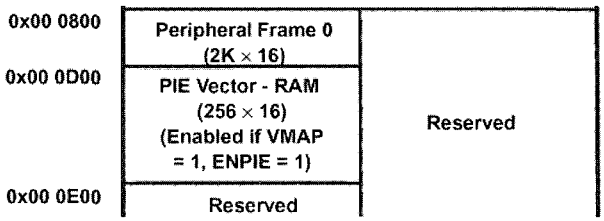


그림 4 TMS320C/F2812의 PIE 벡터 영역

그림 4에서 PIE 벡터영역을 자세히 보자. 그리고, 그림 1에서 본 BROM 영역과 비교해보시기 바란다. 어떤 부분에서 차이가 나는지.

PIE는 Peripherals Interrupt Enable 의 약자다. ENPIE 비트를 1로 두면, BROM 벡터에 맵핑된 Core 인터럽트가 각각의 개별 인터럽트로 분해되어, PIE Vector 영역에 개별 맵핑된다. 예를 들어, ENPIE 비트가 1인 상태에서, PDPINTA가 발생한다면 0x00 0D40 번지의 벡터가 실행된다. BROM 영역을 사용할 때에 비해서 벡터를 분석하는 작업이 없기에 인터럽트 지연(Latency)이 확연히 줄어든다. 그럼 처음부터 ENPIE를 1로 두고 사용하도록 고정시키면 될 것을 왜 이렇게 복잡하게 만들었나 싶으실 것이다. C2000 계열에 속한 다른 DSP와의 호환을 위해서 마련한 장치다. 28X를 28X 스럽게 사용할려면 ENPIE 비트를 반드시 1로 두고 사용하셔야 본래의 성능을 고스란히 사용할 수 있다.

## 2. TMS320C/F2812 부트 로더의 이해

MC모드에서 리셋이 일어났다고 하자. 그럼 리셋벡터가 실행된다. 표 1에서 살펴보았듯이, 리셋 벡터의 내용은 Init Boot로 점프하는 것이다. Init Boot는 TI가 마스킹한 부트 로더의 시작점이다. 부트 로더는 주로 칩을 시동하는 단계에서 프로그램등을 실행가능한 메모리로 옮겨는 프로그램을 말한다. 일종의 복사 프로그램이다. 주로 ROM과 같은 비휘발성/저속 메모리에서 RAM과 같은 고속 메모리로 프로그램을 옮겨서 고속 수행이 가능하도록 한다. TI가 마스킹한 부트 로더는 그림 2에서 살펴보았듯이, 0x3F FC00 ~ 0x3F FFC0 구역에 위치 한다.

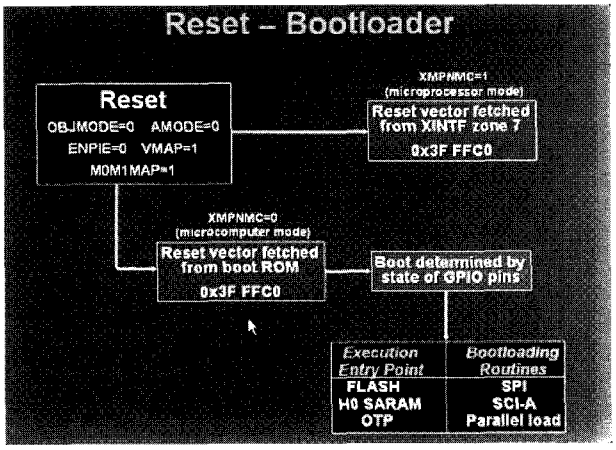


그림 5 28X의 부트 과정

| GPIO pins | | | | |
|---|---|---|---|---|
| F4 | F12 | F3 | F2 | |
| 1 | x | x | x | jump to *FLASH* address 0x3F 7FF6 * |
| 0 | 0 | 1 | 0 | jump to *H0 SARAM* address 0x3F 8000 * |
| 0 | 0 | 0 | 1 | jump to *OTP* address 0x3D 7800 * |
| 0 | 1 | x | x | bootload external EEPROM to on-chip memory via *SPI* port |
| 0 | 0 | 1 | 1 | bootload code to on-chip memory via *SCI-A* port |
| 0 | 0 | 0 | 0 | bootload code to on-chip memory via *GPIO port B (parallel)* |

그림 6 28X의 부트 모드 설정

28X는 아주 다양한 부트모드를 지원한다. 그림 5를 보도록 하자.

그림 5를 보면, MC 모드에서 리셋이 일어나면, 부트로더가 실행되어(이 내용은 그림에 표시되어 있지 못하다.) 특정 핀의 상태를 점검한다. 그 핀의 상태에 따라서 부트 모드를 결정한다. 가능한 부트 모드가 우측 하단 표에 나타나 있다. 그림 6는 부트 로드 옵션이 GPIO 핀 상태에 의해 결정되는 내용을 보여 준다.

내용은 간단하다. TI가 마스킹한 부트로더에 핀 상태를 감지하는 내용이 있다. 감지된 핀 상태에 따라서, 코드 시작점을 어디로 할 것인가가 정해진다. 혹시 eZdsp2812 보드를 공장 출하 상태로 사용하시는 분들이라면, MC 모드로 되어 있고, 코드 시작점이 H0 영역이라는 것을 잘 아실 것이다. 이런 이유로 링커 커맨더 파일에서 H0 영역에 코드를 배치하도록 기술하는 것이다. 당연히 부트모드가 달라지면 링커 커맨더 파일도 달라져야 하고, 부트 모드에 따라 적절한 필요 조치를 해주어야 한다.

각별한 주의가 요구되는 부분은 코드 개발을 모두 끝마치고, 코드를 플래시 메모리에 구워서 동작시키고자 한다면, 그림 6에서 보듯이 플래시 부트 모드로 GPIO 핀상태를 바꿔주어야 한다. 그리고, 플래시 메모리는 램보다 속도가 느리다. 28X에 탑재된 플래시 메모리는 대략 80~100MHz 까지 동작이 가능하다. 따라서 150MHz로 속도를 맞출려면 wait가 필요하다. 이런 내용이 플래시 메모리 사용시 주의점이 되는데, 지면 관계상 이 내용은 TI에서 제공한 응용보고서를 소개함으로써 대신하고자 한다.

Texas Instruments, "Running an Application from Internal Flash Memory on the TMS320F281x DSP", Literature Number : SPRA958A, Nov. 2003.

## 〈저 자 소 개〉

**백종철(白宗哲)**

1992년 KAIST 전기 및 전자공학과 졸업. 1994년 동 대학원 전기 및 전자공학과 졸업(석사). 1999년 현대오토넷(구 현대전자 전장사업본부) 근무. 2002년 Texas Instruments Korea 근무. 2002년 11월~현재 주식회사 싱크웍스 이사.