ARM 코어 시스템 기반 예외 처리를 위한 벡터 테이블 구성 및 인터럽트 제어

정 준 영*, 정 민 수*, 권오형*
*경남대학교 컴퓨터공학과

Vector Table Composition and Interrupt Control for Exception Handling Based on ARM Core System

Jun-Young Jung*, Min-Soo Jung*, Oh-Hyung Kweon*
*Dept. of Computer Engineering, Kyungnam University

요 약

최근 이동단말기나 PDA, 스마트폰과 같은 정보기기나 디지털 가전기기의 사용이 증대됨에 따라, ARM 코어 시스템을 기반으로 하는 프로세서와 이를 운영하기 위한 소프트웨어 수요도 증가하고 있다. 본 논문은 프로세서를 운영하기 위한 소프트웨어 중에서 예외처리를 위한 일반적인 인터럽트 제어를 다룬다. ARM 시스템 상에서 임의의 주변 장치(타이머/카운터)에 의해 발생하는 인터럽트 처리 과정과 예외처리를 제어하기 위한 벡터 테이블을 구성하는 방법에 대해 분석한다. 그리고 인터럽트를 처리하는 인터럽트 코드부분과 벡터 테이블내의 인터럽트의 상호 연관성에 대해 논의한다.

1. 서 론

ARM 코어를 기반으로 하는 마이크로 프로세서는 이동전화기, 개인휴대단말기(PDA), 스마트폰등 각종 정보기기와 네트워크 장비등에 널리 사용되고 있다. ARM 코어는 CISC 보다는 간단하게 디자인된 RISC(Reduced Instruction Set Computer)로, 내장 응용프로그램에 적합하다. 그러나 명령어가 CISC 보다는 적기 때문에 주어진 작업에 대한 완전한 처리를 위해서는 보다 많은 명령어들을 필요로 한다. 이러한 문제점을 해결하기 위해 32 비트 ARM 명령어를 Thumb 컴파일러를 이용하여 16 비트 명령어로 생성하고 실행시에는 압축을 풀어서 32 비트 ARM 명령어로 동작하도록 설계되었다.[1,2,4,8]

ARM 은 User, FIQ, IRQ, Supervisor, Abort, Undef 와 같은 여섯 가지의 기본 운영 모드를 가지고 있는데, 거의 모든 작업들은 User 모드에서 실행되고, 일반적인 인터럽트가 발생하면 IRQ 모드로 전환된다. 각 모드에서 사용되는 레지스터들도 서로 다르다. User 모드에서는 r0~r12(범용 레지스터), sp(stack pointer), lr(Link register), pc(program counter), cpsr(current processor status register)와 같은 레지스터가 사용되고, IRQ 모드에서는 r0~r12, pc, cpsr 과 같은 레지스터가 사용된다.[8,9]

본 논문은 디지털기기에서 사용될 ARM 기반 프로세서 운영시에 예상되는 예외를 처리 하기 위한 인터럽트의 설치와 인터럽트를 발생시키는 카운터/타이머 컨트롤러를 이용한 일반적인 인터럽트의 실행 과정, 인터럽트의 유형에 기초한 예외 타입의 벡터 테이블 구조, 그리고 인터럽트와 벡터테이블 사이의 관계를 분석한다.

본 논문의 구성은 다음과 같다. 2 장에서는 인터럽트와 카운터/타이머의 상호 작동하는 흐름을 설명한다. 그리고 IRQ 와 Timer2 의 메모리 맵과 레지스터의 기능에 대해 서술하고, 이들 레지스터에서 각 비트의 의미를 살펴보고 3 장에서는 벡터 테이블 내에 위치하게 되는 예외 상황의 종류와 운영 모드를 기술하고, 인터럽트를 제어하기 위해 벡터 테이블을 구성하는 방법과 인터럽트와 타이머 루틴의 구현을 관찰함으로써 상호 연관성을 분석 설명하고, 4 장에서는 결론과 향후 연구방향에 대해 언급한다.

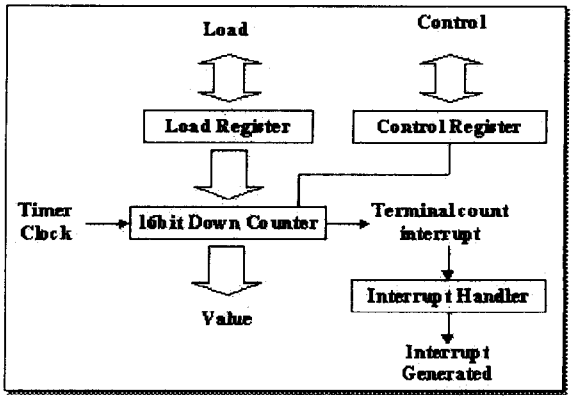
2. ARM 코어 시스템 기반 IRQ(인터럽트)와 타이머/카운터

ARM 코어 시스템의 주변 장치(reference pherals)에는 인터럽트 컨트롤러(15 IRQ 와 1 FIQ), 두개의 16 비트 카운터/타이머(Timer1,Timer2), 그리고 remap 과 pause 컨트롤러가 있다. 본 연구에서는 IRQ 와 Timer2 를 이용해서 분석한다.

2.1 타이머/카운터와 인터럽트 수행

타이머/카운터의 값을 초기화 하면 16 비트 다운 카운터는 감소하면서 0 이 되면 인터럽트를 발생시킨다. 운영 모드는 User 모드에서 IRQ 모드로 전환되고 인터럽트 제어기는 발생된 인터럽트를 입력으로 받고 벡터 테이블에서 해당되는 인터럽트 타입을 찾아서 핸들러 처

리루틴으로 분기하여 실행된다. [5,6]



[그림 1] 인터럽트와 타이머/카운터의 실행 구조

2.2 IRQ(인터럽트)

인터럽트 메모리와 레지스터의 기능은 다음과 같다.[2,5,6,8]

[표 2] 인터럽트 컨트롤러 레지스터에 대한 주소

Address	Read Location	Write Location
0x0A00000	IRQStatus	Reserved
0x0A00004	IRQRawStatus	Reserved
0x0A00008	IRQEnable	IRQEnableSet
0x0A0000C	Reserved	IRQEnableClear
0x0A00010	Reserved	Programmed IRQ Interrupt

- **Enable Register**
  어떤 비트 패턴의 특정한 비트 위치정보를 변경 하거나 삭제하기 위하여 검사하는데 사용된다. 이 레지스터는 Enable Set 에 의해 값이 변경된다.
- **Enable Set**
  인터럽트 Enable 레지스터의 비트를 설정하는데 사용된다.
- **Enable Clear**
  인터럽트 Enable 레지스터의 비트를 제거하는데 사용된다.
- **Source Status**
  인터럽트 컨트롤러에게 매스킹(masking)하기 전의 인터럽트 소스의 상태를 제공한다.
- **Interrupt Request**
  인터럽트 컨트롤러에게 매스킹(masking)한 후의 인터럽트 소스의 상태를 제공한다.
- **Programmed IRQ Interrupt**
  이 레지스터는 Programmed Interrupt 를 설정하거나 제거한다. 비트를 1 로 설정하면 programmed interrupt 를 생성하고, 비트를 0 으로 설정하면 programmed interrupt 를 제거한다. 레지스터의 값은 Source Status Register 의 비트 1 을 읽어 들임으로써 결정 될 수 있다.

2.3 카운터/타이머

타이머가 사용하는 주소 값과 레지스터의 기능은 다음과 같다.[2,5,6,8]

[표 2] 타이머 레지스터에 대한 주소

Address	Read Loaction	Writion Location
0x0A800000	Timer1Load	Timer1Load
0x0A800004	Timer1Value	Reserved
0x0A800008	Timer1Control	Timer1Control
0x0A80000C	Reserved	Timer1Clear
0x0A800020	Timer2Load	Timer2Load
0x0A800024	Timer2Value	Reserved
0x0A800028	Timer2Control	Timer2Control
0x0A80002C	Reserved	Timer2Clear

- **Load Register**
  타이머의 초기값을 가지고 있으며, Periodic timer 모드에서 값을 읽어오는데 사용된다.
- **Value**
  타이머의 현재 값을 가지고 있다.
- **Clear**
  카운터 타이머에 의해 생성된 인터럽트를 제거한다.
- **Control Register**
  타이머의 Enable/Disable, 모드, Prescale 설정등을 제공한다.

2.3 벡터 테이블

(1) 벡터 테이블의 구조와 예외타입(Exception Type)

벡터 테이블은 프로세서 예외 처리를 제어하는데 사용되는 32 바이트의 예약된 영역이다. 일반적으로 벡터 테이블은 메모리 구조에서 하위에 위치하는데, 각 예외 타입(exception type)에 1 워드 공간을 할당하고, 1 워드는 예약된 공간이다. 하지만 1 워드는 핸들러에 대한 전체 코드를 포함하기에는 충분하지 못하다. 그래서 각 예외 타입에 대한 벡터 엔트리는 적절한 핸들러 실행을 위해 예외에 대한 Branch 나 Load PC 을 포함하고 있다.[2,5,6,8]

[표 3] 예외타입에 대한 벡터 주소와 예외모드

Vector Address	Exception Type	Exception Mode
0x00	Reset	Supervisor(SVC)
0x04	Undefined Instruction	Undef
0x08	Software Interrupt(SWI)	Supervisor(SVC)
0x0C	Prefech Abort	Abort
0x10	Data Abort	Abort
0x14	Reserved	Not Application
0x18	Interrupt(IRQ)	Interrupt(irq)
0x1C	Fast Interrupt(FIQ)	Fast interrupt(fiq)

(2) 벡터 테이블내에 예외 핸들러 설치

벡터 테이블내에 예외 핸들러를 설치하는 두가지 방법은 Branch Method 와 Load PC Method 이다.[5,6,8]

- **Branch Method**
  ① 예외 핸들러의 주소를 얻는다.
  ② 예외 핸드러의 주소에서 IRQ 의 벡터 주소(0x18)를 감산한다.
  ③ Prefetching 이 가능하도록 0x8 을 감산한다.
  ④ 오른쪽으로 2 만큼 쉬프트 연산을 수행한다.

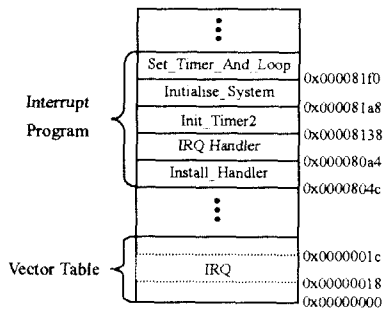
⑤ 결과가 24 비트 길이만을 가지도록 상위 8 비트를 삭제한다.(Branch offset 를 제한)
⑥ 벡터에 위치될 값을 만들기 위해 0xea000000 (Branch 명령에 대한 opcode)를 사용해서 OR 연산을 한다.

• **Load PC Method**

① 예외 핸들러의 주소를 얻는다.
② IRQ 의 벡터 주소(0x18)를 감산한다.
③ 파이프라인이 가능하도록 0x8 을 감산한다.
④ 벡터에 위치될 값을 만들기 위해 0xe59ff000 를 사용해서 OR 연산을 한다.

3. 인터럽트 프로그램의 세부 모듈

인터럽트 프로그램이 실행시에 모듈이 기억공간에 위치 되는 구조는 다음과 같다.[5,6]



[그림 2] 모듈별 기억공간 구조

(1) Install_Handler

인터럽트 핸들러 설치는 Branch 방법을 사용한다. 인터럽트는 벡터 테이블에 의해 제어되는데, 새로운 벡터 테이블 값은 인터럽트 핸들러의 엔트리 주소에서 인터럽트 벡터 주소(0x18)와 파이프 오셋(0x08)을 감산한 뒤, 오른쪽으로 2 만큼 쉬프트 시킨다. 계산된 벡터 테이블 값은 0xea000000 와 새로운 벡터 값을 OR 연산 하고 인터럽트 벡터 공간에 저장한다.[5,6,8]

```
unsigned Install_Handler
(unsigned routine, unsigned *vector)
{
  unsigned new_vector_value = 0;
  old_vector_value = 0;
  new_vector_value = ((routine - (unsigned)vector
                  - PIPE_OFFSET) >> WORD_OFFSET);
  if (new_vector_value & CHECK_24_BIT)
     {
      printf("Installation of Handler failedWn");
      return 0;
     }
 new_vector_value = BRANCH_OP_CODE |
                    new_vector_value;
 old_vector_value = *vector;
 *vector =new_vector_value;
 return (old_vector_value);
}
```

(2) Init_Timer2

카운터/타이머는 Disable 되고, 어떠한 타이머 인터럽트도 발생하지 않도록 설정한다.[5,6,9]

```
int Init_Timer2(void)
{
  *IRQEnableClear = ~0;
  if (((*IRQEnable & 0xffff) != 0) )
     {
      printf("error, interrupts still enabled);
     }
  *Timer2Control = 0;
  *Timer2Clear = 0 ;
   return 0;
}
```

(3) Set_Timer_And_Loop

타이머에 대한 카운터 값, Enable, Periodic, Prescale 등을 설정한다. 인터럽트를 모두 제거하고, 인터럽트가 Timer2 에서 발생 가능하도록 설정한다. 그리고 인터럽트의 관찰이 가능하도록 무한 루프를 실행한다.[2,5,6]

```
int Set_Timer_And_Loop(void)
{
  printf("Setting periodic timer interrupts...Wn");
  *Timer2Load    = 0xffff;
  printf("C2Load = 0x%xWn",*Timer2Load & 0xFFFF);
  *Timer2Control = (TimerEnable |
                   TimerPeriodic | TimerPrescale8 );
  *IRQEnableClear = ~0;
  *IRQEnableSet = IRQTimer2 ;
  while (INFINITE_LOOP) {};
  return 0;
}
```

4. 결론

ARM 시스템에서는 여섯 가지의 모드를 가지고, 인터럽트 핸들링은 벡터 테이블 내에 명시된 예외처리의 타입에 따라 Branch 방법에 의해 인터럽트 핸들링의 처리 루틴으로 제어가 전달된다. 인터럽트 컨트롤러와 주변 장치는 고유의 주소 레지스터를 사용해서 이들간의 상호 연관성을 가지면서 작동한다.

향후 디지털 가전 기기나 정보가전 기기내에서 발생할 수 있는 예외처리에 적용할 수 있는 제어 프로그램을 설계하는데 기초자료로서 활용할 수 있다

참 고 문 헌

[1] Steve Furver, "ARM System Architecture" Addison-Wesely
[2] http://www.arm.com
[3] ARM Limited, "ARM datasheets"
[4] van Someren and Atack, "The ARM RISC Chip"
[5] ARM DUI 0061A, "Programmer's Model of the ARM Development Board"
[6] ARM DDI 0062D, "Reference Peripherals Specificaation"
[7] ARM DUI 0040D Manual, " ARM Software Development Toolkit"
[8] ARM DDI 0100B, "ARM Architecture Reference Manual"