EDDI
Electronic Design
Development Institute

에디로봇아카데미

임베디드 마스터 Lv2 과정

제 1기

2022. 01. 21

손표훈

CONTENTS

EDDI
Electronic Design Development Institute

- RTI(Real Time Interrupt)의 구조
  - HALCOGEN을 이용한 1초 인터럽트 만들기
  - counter레지스터의 shadow register사용?
- RTI 인터럽트 관련 함수
  - rtiEnableNotification 함수
  - _enable_IRQ_interrupt_ 함수
  - rtiStartCounter 함수
  - rtiNotification 함수
    - WEAK symbol?
    - 다중 rti 인터럽트 사용시 rtiNotification 함수
- ARM코어의 인터럽트 처리과정
  - ARM코어 인터럽트 vector table
  - 예 : ARM코어 IRQ 처리과정
  - TI Hercules 시리즈의 인터럽트
  - IRQ vs FIQ

RTI 구조

Figure 17-1. RTI Block Diagram

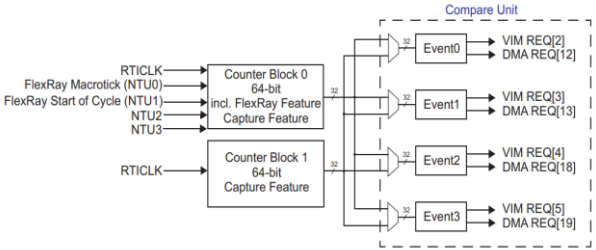
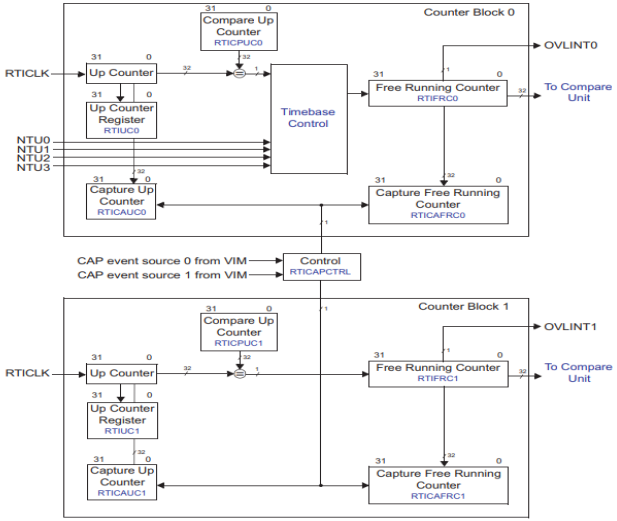


Figure 17-2. Counter Block Diagram



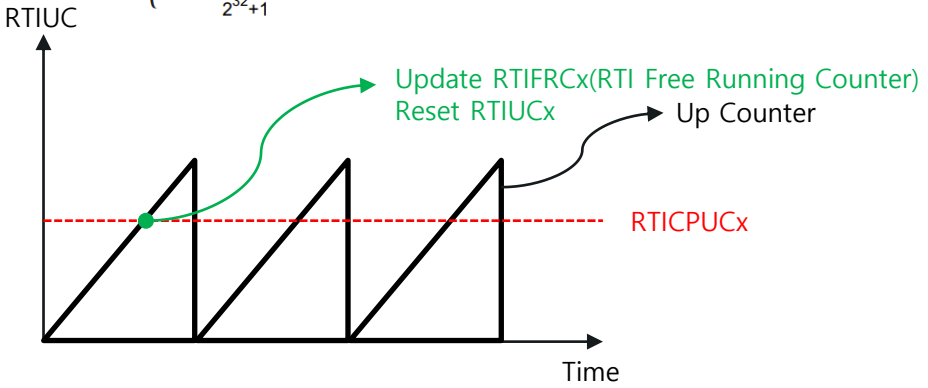
17.1.1 *Features*

The RTI module has the following features:

- Two independent 64 bit counter blocks
- Four configurable compares for generating operating system ticks or DMA requests. Each event can be driven by either counter block 0 or counter block 1.
- One counter block usable for application synchronization to FlexRay network including clock supervision
- Fast enabling/disabling of events
- Two time stamp (capture) functions for system or peripheral interrupts, one for each counter block
- Digital windowed watchdog

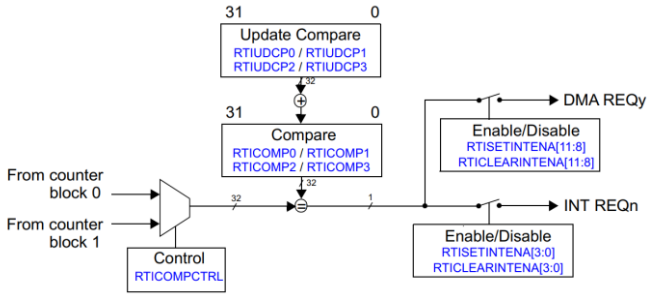
* RTI의 clock source 주기 계산 및 카운트 동작

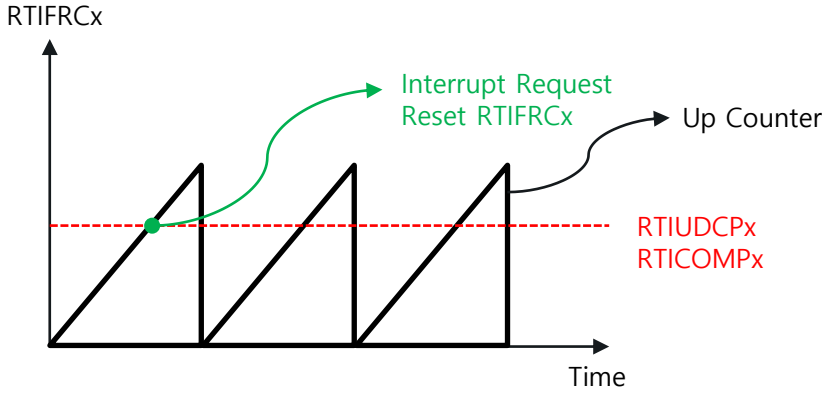
$$f_{RTIFRCx} = \begin{cases} \dfrac{f_{RTICLK}}{RTICPUCx + 1} & \text{when } RTICPUCx \neq 0 \\ \dfrac{f_{RTICLK}}{2^{32}+1} & \text{when } RTICPUCx = 0 \end{cases}$$



RTI 구조

* RTI의 인터럽트 주기 계산 및 카운트 동작





* RTI의 인터럽트 주기 계산

$$t_{COMPx} = t_{RTICLK} \times (RTICPUCy + 1) \times RTIUDCPy$$

if RTICPUCy ≠ 0,

$$t_{COMPx} = t_{RTICLK} \times (2^{32}+1) \times RTIUDCPy$$

if RTIUDCPy = 0,

$$t_{COMPx} = t_{RTICLK} \times (RTICPUCy + 1) \times 2^{32}$$

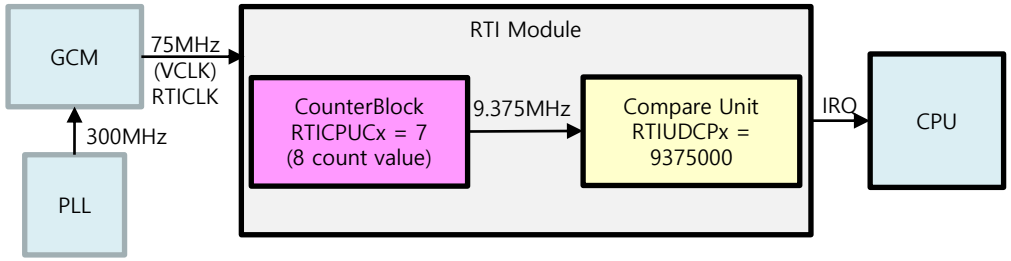
→ RTICOMPCTRL을 이용하여 Counter Block을 선택한다

→ RTISETINTENA/RTICLEARINTENA을 이용하여 interrupt 요청을 CPU로 전달/차단 한다

RTI 구조

EDDI
Electronic Design Development Institute

➢ RTI 1초 인터럽트 만들기 & rtiInit



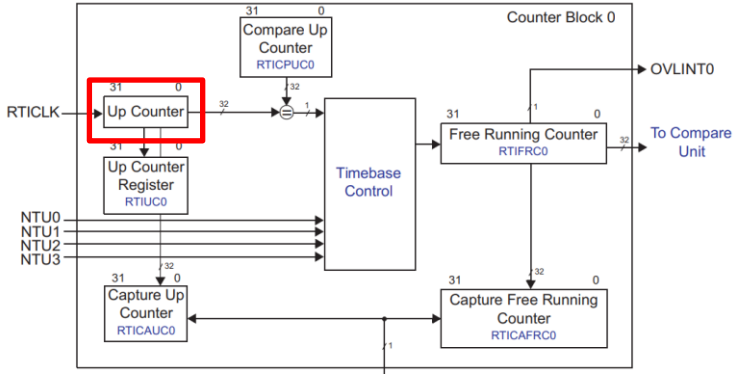
(1) PLL을 통해 300MHz가 GCM으로 입력되고, GCM에서 75MHz로 분주한다.
(2) 분주된 75MHz는 RTI Module로 입력되고, CounterBlock을 통해 RTICPUCx 레지스터 값을 설정하여 75MHz/8 = 9.375MHz로 분주시킨다.
(3) 9.375MHz(=0.10666667us)가 compare unit으로 입력되고 RTIFRCx 값을 업카운트한다.
(4) Compare Unit의 RTIUDCPx값을 9375000으로 설정하면 0.10666667us x 9375000 = 1.038s

```
/** - Setup NTU source, debug options and disable both counter blocks */
rtiREG1->GCTRL = (uint32)((uint32)0x5U << 16U) | 0x00000000U;

/** - Setup timebase for free running counter 0 */
rtiREG1->TBCTRL = 0x00000000U;

/** - Enable/Disable capture event sources for both counter blocks */
rtiREG1->CAPCTRL = 0U | 0U;

/** - Setup input source compare 0-3 */
rtiREG1->COMPCTRL = 0x00001000U | 0x00000100U | 0x00000000U | 0x00000000U;

/** - Reset up counter 0 */
rtiREG1->CNT[0U].UCx = 0x00000000U;

/** - Reset free running counter 0 */
rtiREG1->CNT[0U].FRCx = 0x00000000U;

/** - Setup up counter 0 compare value
*     - 0x00000000: Divide by 2^32
*     - 0x00000001-0xFFFFFFFF: Divide by (CPUC0 + 1)
*/
rtiREG1->CNT[0U].CPUCx = 7U;

/** - Setup compare 0 value. This value is compared with selected free running counter. */
rtiREG1->CMP[0U].COMPx = 9375000U;

/** - Setup update compare 0 value. This value is added to the compare 0 value on each compare match. */
rtiREG1->CMP[0U].UDCPx = 9375000U;
```

RTI 구조

➢ Q. counter레지스터의 shadow register사용?

Figure 17-2. Counter Block Diagram



Reading the Counters

The free running counter (RTIFRCx) must be read first. This priority will ensure that in the cycle when the CPU reads RTIFRCx, the up counter value is stored in its counter register (RTIUCx). The second read has to access the up counter register (RTIUCx), which then holds the value which corresponds to the number of RTICLK cycles that have elapsed at the time reading the free running counter register (RTIFRCx).

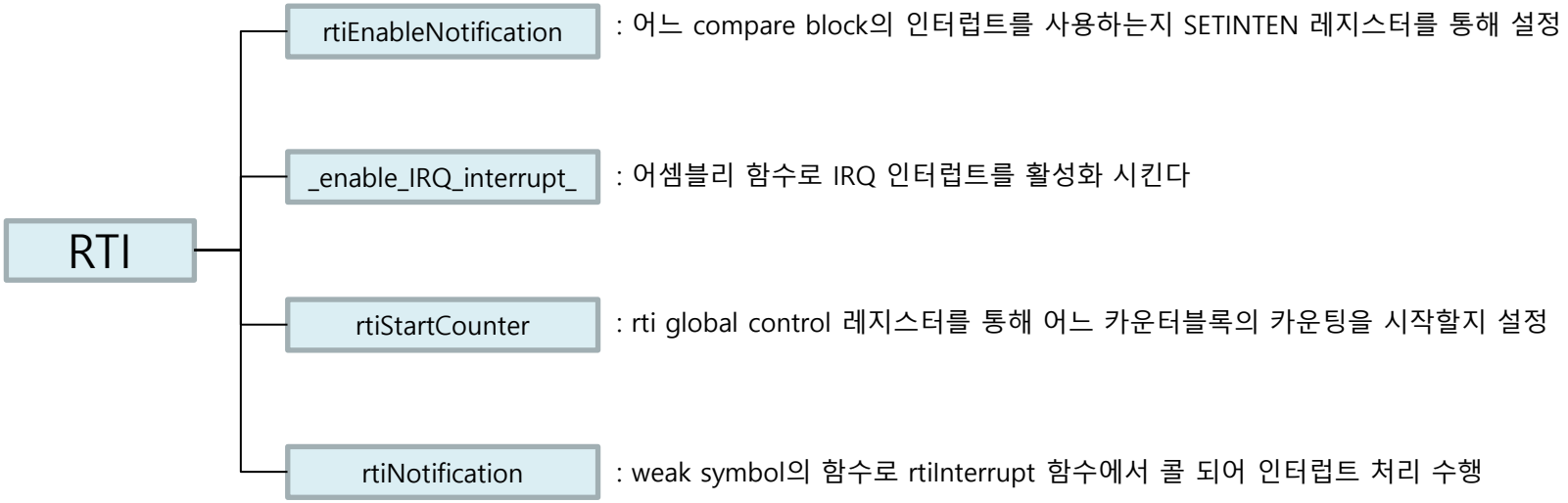
NOTE: The up counters are implemented as shadow registers. Reading RTIUCx without having read RTIFRCx first will return always the same value. RTIUCx will only be updated when RTIFRCx is read.

→ RTI의 counter는 2진, 8진 카운터처럼 클럭이 입력되면 카운트 되는 카운터 H/W로 알고 있는데
비교 레지스터에서 설정한 비교 값과 비교하여 인터럽트 요청 신호를 cpu로 자동으로 넘겨주는데 굳이 shadow register를 처리한 이유를 알고 싶습니다.

→ rti block의 레지스터와 SW상 비교 레지스터의 비교 값 설정 명령어 처리상 동기화 오류가 생겨 사용할 것으로 추측했는데 이미 rtiInit함수에서 비교 값을 고정 시켜놓기 때문에 shadow register의 의미가 없을 거라고 생각됩니다.

RTI 인터럽트 관련 함수

※ rti 인터럽트 관련 함수(simple test 관련)



RTI 인터럽트 관련 함수

EDDI
Electronic Design Development Institute

➢ rtiEnableNotification 함수

```
void rtiEnableNotification(rtiBASE_t *rtiREG, uint32 notification)
{
/* USER CODE BEGIN (38) */
/* USER CODE END */

    rtiREG->INTFLAG = notification;
    rtiREG->SETINTENA    = notification;

    /**   @note The function rtiInit has to be called before this function can be used.\n
     *           This function has to be executed in privileged mode.
     */

/* USER CODE BEGIN (39) */
/* USER CODE END */
}
```

```
//리얼타임 인터럽트를 누가 처리할지 설정
//weak reference : 함수 오버로딩
//RTI compare0에서 발생하는 인터럽트 허용
rtiEnableNotification(rtiREG1, rtiNOTIFICATION_COMPARE0);
```

17.3.27 RTI Interrupt Flag Register (RTIINTFLAG)

The corresponding flags are set at every compare match of the RTIFRCx and RTICOMPx values, whether the interrupt is enabled or not. This register is shown in Figure 17-38 and described in Table 17-28.

Bit	Field	Value	Description
0	INT0		Interrupt flag 0. These bits determine if an interrupt due to a Compare 0 match is pending.
		0	*Read:* No interrupt is pending. *Write:* Bit is unchanged.
		1	*Read:* Interrupt is pending. *Write:* Bit is cleared to 0.

17.3.25 RTI Set Interrupt Enable Register (RTISETINTENA)

This register prevents the necessity of a read-modify-write operation if a particular interrupt should be enabled. This register is shown in Figure 17-36 and described in Table 17-26.

0	SETINT0		Set compare interrupt 0.
		0	*Read:* Interrupt is disabled. *Write:* Corresponding bit is unchanged.
		1	*Read or Write:* Interrupt is enabled.

RTI 인터럽트 관련 함수

➢ _enable_IRQ_interrupt_ 함수

→ 해당 함수는 HL_sys_core.asm 파일에 정의되어 있다.

어셈블리어로 정의된 함수를 c에서 호출할 수 있게 설정

```
;---------------------------------------------
; Enable interrupts - CPU IRQ
; SourceId : CORE_SourceId_026
; DesignId : CORE_DesignId_022
; Requirements: HL_CONQ_CORE_SR8

    .def _enable_IRQ_interrupt_
    .asmfunc

_enable_IRQ_interrupt_

        cpsie i
        bx    lr

        .endasmfunc
```

인터럽트 disable 함수

```
; Disable IRQ interrupt
; SourceId : CORE_SourceId_025
; DesignId : CORE_DesignId_021
; Requirements: HL_CONQ_CORE_SR11

        .def _disable_IRQ_interrupt_
        .asmfunc

_disable_IRQ_interrupt_

        cpsid i
        bx    lr

        .endasmfunc
```

RTI 인터럽트 관련 함수

EDDI
Electronic Design Development Institute

➢ _enable_IRQ_interrupt_ 함수

→ _enable_IRQ_interrupt_ 함수의 동작

```
;-----------------------------------------------
; Enable interrupts - CPU IRQ
; SourceId : CORE_SourceId_026
; DesignId : CORE_DesignId_022
; Requirements: HL_CONQ_CORE_SR8

        .def _enable_IRQ_interrupt_
        .asmfunc

_enable_IRQ_interrupt_

         cpsie i
         bx    lr

         .endasmfunc
```

(1) cpsie i : Change Processor State Interrupt Enable로 CPS 레지스터의 7번째 bit인 "I" bit를 1 -> 0으로 설정하여 IRQ interrupt 모드를 활성화 한다

A2.5.6 The interrupt disable bits

A, I, and F are the interrupt disable bits:

A bit Disables imprecise data aborts when it is set. This is available only in ARMv6 and above. In earlier versions, bit[8] of CPSR and SPSRs must be treated as a reserved bit, as described in *Types of PSR bits* on page A2-11.

I bit Disables IRQ interrupts when it is set.

F bit Disables FIQ interrupts when it is set.

CPS

31 30 29 28	27 26 25 24 23 22 21 20	19 18	17	16	15 ... 9	8	7	6	5	4 ... 0
1 1 1 1	0 0 0 1 0 0 0 0	imod	mmod	0	SBZ	A	I	F	0	mode

CPS (Change Processor State) changes one or more of the mode, A, I, and F bits of the CPSR, without changing the other CPSR bits.

(2) bx lr : lr은 레지스터(R14)로 subroutine의 복귀주소를 저장하고있다.
main에서 _enable_IRQ_interrupt_ 함수 호출 다음 주소로 분기명령(bx)를 통해 복귀한다.

Register R14 (also known as the *Link Register* or LR) has two special functions in the architecture:

- In each mode, the mode's own version of R14 is used to hold subroutine return addresses. When a subroutine call is performed by a BL or BLX instruction, R14 is set to the subroutine return address. The subroutine return is performed by copying R14 back to the program counter. This is typically done in one of the two following ways:
  - Execute a BX LR instruction.

RTI 인터럽트 관련 함수

EDDI
Electronic Design Development Institute

➢ rtiStartCounter 함수

```
void rtiStartCounter(rtiBASE_t *rtiREG, uint32 counter)
{
/* USER CODE BEGIN (4) */
/* USER CODE END */

    rtiREG->GCTRL |= ((uint32)1U << (counter & 3U));

    /**   @note The function rtiInit has to be called before this function can be used.\n
     *           This function has to be executed in privileged mode.
     */

/* USER CODE BEGIN (5) */
/* USER CODE END */
}

//리얼타임 인터럽트 카운터 시작 : rtos 스케쥴링시 타이밍 조정하는 기능을 할 때도 rti의 카운터가 필요
//실제로는 RTI counter block0을 시작하여 카운팅을 진행
rtiStartCounter(rtiREG1, rtiCOUNTER_BLOCK0);
```

17.3.1 RTI Global Control Register (RTIGCTRL)

The global control register starts/stops the counters and selects the signal compared with the timebase control circuit. This register is shown in Figure 17-12 and described in Table 17-2.

31	20	19	16
Reserved		NTUSEL	
R-0		R/WP-0	

15	14	2	1	0
COS	Reserved		CNT1EN	CNT0EN
R/WP-0	R-0		R/WP-0	R/WP-0

0	CNT0EN		Counter 0 enable. This bit starts and stops counter block 0 (RTIUC0 and RTIFRC0).
		0	Counter block 0 is stopped
		1	Counter block 0 is running

RTI 인터럽트 관련 함수

EDDI
Electronic Design Development Institute

- rtiNotification 함수
  - → rtiNotification 함수는 HL_notification.c에서 WEAK 심볼로 처리 되어있다.
  - → HL_rti.c에서 rtiInterrupt 함수에 의해 콜 된다.
  - → HL_sys_main.c에서 함수 기능이 정의 하거나 사용자정의 파일에서 정의 할 수 있다.

```
/* USER CODE BEGIN (11) */
/* USER CODE END */
#pragma WEAK(rtiNotification)
void rtiNotification(rtiBASE_t *rtiREG, uint32 notification)
{
/*  enter user code between the USER CODE BEGIN and USER CODE END. */
/* USER CODE BEGIN (12) */
/* USER CODE END */
}
```

HL_notification.c

```
#pragma CODE_STATE(rtiCompare0Interrupt, 32)
#pragma INTERRUPT(rtiCompare0Interrupt, IRQ)

/* SourceId : RTI_SourceId_022 */
/* DesignId : RTI_DesignId_022 */
/* Requirements : HL_CONQ_RTI_SR12 */
void rtiCompare0Interrupt(void)
{
/* USER CODE BEGIN (43) */
/* USER CODE END */

    rtiREG1->INTFLAG = 1U;
    rtiNotification(rtiREG1,rtiNOTIFICATION_COMPARE0);

/* USER CODE BEGIN (44) */
/* USER CODE END */
}
```

HL_rti.c

```
/* USER CODE BEGIN (4) */
//이 부분은 RTI Interrupt Handler로 카운터가 차면 구동됨
void rtiNotification(rtiBASE_t *rtiREG, uint32_t notification)
{
    switch(notification)
    {
    case rtiNOTIFICATION_COMPARE0:
        gioToggleBit(gioPORTA, 4);
        break;
    case rtiNOTIFICATION_COMPARE1:
        gioToggleBit(gioPORTA, 5);
        break;
    }
#if 0
    //GPIO PORTA의 4번을 토글
    gioToggleBit(gioPORTA, 4);
#endif
}
```

HL_sys_main.c

RTI 인터럽트 관련 함수

EDDI
Electronic Design Development Institute

- rtiNotification 함수
  - WEAK symbol?

```
/* USER CODE BEGIN (11) */
/* USER CODE END */
#pragma WEAK(rtiNotification)
void rtiNotification(rtiBASE_t *rtiREG, uint32 notification)
{
/*  enter user code between the USER CODE BEGIN and USER CODE END. */
/* USER CODE BEGIN (12) */
/* USER CODE END */
}
```

HL_notification.c

```
/* USER CODE BEGIN (4) */
//이 부분은 RTI Interrupt Handler로 카운터가 차면 구동됨
void rtiNotification(rtiBASE_t *rtiREG, uint32_t notification)
{
    switch(notification)
    {
    case rtiNOTIFICATION_COMPARE0:
        gioToggleBit(gioPORTA, 4);
        break;
    case rtiNOTIFICATION_COMPARE1:
        gioToggleBit(gioPORTA, 5);
        break;
    }
#if 0
    //GPIO PORTA의 4번을 토글
    gioToggleBit(gioPORTA, 4);
#endif
}
```

HL_sys_main.c

(1) WEAK symbol? WEAK symbol은 ELF(Executable Linkable Format)파일을 linking하는 과정에서 특별히 사용되는 symbol이다. 별도의 WEAK symbo을 사용하지 않으면 STRONG symbol임을 의미한다.
(2) WEAK vs STRONG : STRONG으로 표기된 symbo이 WEAK 표기된 symbol을 Override한다.
(3) WEAK 표기된 symbol은 별도의 definition없이 사용이 가능하나, STRONG은 반드시 definition이 있어야 컴파일 에러가 발생하지 않는다.
(4) 왜 WEAK를 사용하는지? WEAK 처리된 함수가 정의된 파일에서 상황에 맞게 함수를 꺼내 쓸 수 있게 함수의 default implementation을 제공할 때 사용된다.
HL_notification.c에는 다양한 주변장치들의 notificatio함수가 있다.
이 함수들은 모두 WEAK처리 되어있으며, 사용시 HL_notification.c를 참조하여 사용자가 원하는 파일(예 : main.c)에서 함수의 상세 기능을 정의하여 사용할 수 있게 한다.
(5) 소스파일의 유지보수 측면에서 유리?

RTI 인터럽트 관련 함수

EDDI
Electronic Design Development Institute

➢ rtiNotification 함수

❖ 다중 rti 인터럽트 사용시 rtiNotification 함수 활용

```
#pragma CODE_STATE(rtiCompare0Interrupt, 32)
#pragma INTERRUPT(rtiCompare0Interrupt, IRQ)

/* SourceId : RTI_SourceId_022 */
/* DesignId : RTI_DesignId_022 */
/* Requirements : HL_CONQ_RTI_SR12 */
void rtiCompare0Interrupt(void)
{
/* USER CODE BEGIN (43) */
/* USER CODE END */

    rtiREG1->INTFLAG = 1U;
    rtiNotification(rtiREG1,rtiNOTIFICATION_COMPARE0);

/* USER CODE BEGIN (44) */
/* USER CODE END */
}
```

HL_rti.c

```
//이 부분은 RTI Interrupt Handler로 카운터가 차면 구동됨
void rtiNotification(rtiBASE_t *rtiREG, uint32_t notification)
{
    switch(notification)
    {
    case rtiNOTIFICATION_COMPARE0:
        gioToggleBit(gioPORTA, 4);
        break;
    case rtiNOTIFICATION_COMPARE1:
        gioToggleBit(gioPORTA, 5);
        break;
    }
#if 0
    //GPIO PORTA의 4번을 토글
    gioToggleBit(gioPORTA, 4);
#endif
}
```

HL_sys_main.c

(1) rtiNotification은 rti.c에서 rtiCompare0Interrupt함수에서 콜 된다.
(2) rtiCompare0Interrupt를 보면 rtiNotification함수의 파라메터로 rti레지스터와 rtiNotification_compare0(= 0)이 입력된다.
(3) main.c에서 정의한 rtiNotification 함수에서 위에 입력 받은 compare0/compare1 파라메터를 식별하여 해당 인터럽트에 맞는 인터럽트 처리를 하면 된다.

ARM코어의 인터럽트 처리과정

EDDI
Electronic Design Development Institute

➢ ARM코어 인터럽트 vector table

예외 상황	프로세서 모드	벡터 주소
리셋	SVC(supervisor)	0x00000000
Undefined instruction	undefined	0x00000004
Software interrupt	SVC(supervisor)	0x00000008
Prefetch abort	Abort	0x0000000C
Data abort	Abort	0x00000010
IRQ(interrupt)	IRQ	0x00000018
FIQ(fast interrupt)	FIQ	0x0000001C

→ ARM의 예외상황에는 7가지가 있다.

ARM코어의 인터럽트 처리과정

EDDI
Electronic Design Development Institute

➢ 예 : ARM코어 IRQ 처리과정

```
R14_irq   = address of next instruction to be executed + 4
SPSR_irq  = CPSR
CPSR[4:0] = 0b10010            /* Enter IRQ mode */
CPSR[5]   = 0                  /* Execute in ARM state */
                               /* CPSR[6] is unchanged */
CPSR[7]   = 1                  /* Disable normal interrupts */
CPSR[8]   = 1                  /* Disable Imprecise Data Aborts (v6 only) */
CPSR[9]   = CP15_reg1_EEbit    /* Endianness on exception entry */
if VE==0 then
  if high vectors configured then
      PC    = 0xFFFF0018
  else
      PC    = 0x00000018
else
  PC = IMPLEMENTATION DEFINED    /* see page A2-26 */
```

To return after servicing the interrupt, use:

```
SUBS PC,R14,#4
```

(1) 외부 장치로부터 인터럽트 요청 발생
(2) R14(lr)레지스터에 IRQ 다음 명령어의 주소를 저장
(3) SPSR(Save Program Status Register)레지스터에 CPSR(Current Program Status Register)의 값을 저장
(4) CPSR의 모드에 해당하는 bit field를 IRQ로 설정
(5) CPSR의 7번째 bit인 I bit를 1로 설정하여 다른 interrupt 요청(nested interrupt)을 비활성화 시킨다
(6) CPSR의 8번째 bit인 A bit를 1로 설정하여 부정확한 데이터 차단
(7) PC(Program Counter)값을 ISR의 주소로 설정하여 ISR을 실행한다

CPS

31 30 29 28	27 26 25 24 23 22 21 20	19 18	17	16	15 ... 9	8	7	6	5	4 ... 0
1 1 1 1	0 0 0 1 0 0 0 0	imod	mmod	0	SBZ	A	I	F	0	mode

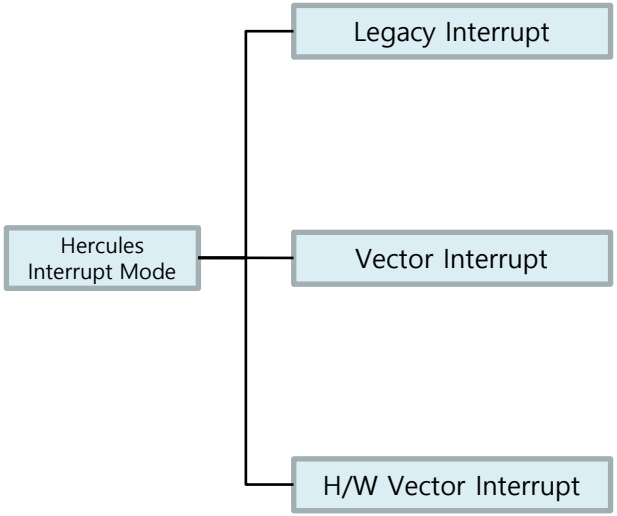
CPS (Change Processor State) changes one or more of the mode, A, I, and F bits of the CPSR, without changing the other CPSR bits.

(8) ISR 종료 후 SUBS명령을 통해 PC를 R14에 저장된 값으로 설정하고 CPSR레지스터에 SPSR레지스터의 값으로 변경하여 현재 프로그램 모드를 예외처리 이전의 모드로 복귀시킨다.
(9) 위 과정을 통해 ARM의 예외처리가 진행된다

ARM코어의 인터럽트 처리과정

EDDI
Electronic Design Development Institute

➢ TI Hercules 시리즈의 인터럽트



: 일반적인 Arm7의 인터럽트와 처리과정이 유사하다
인터럽트 디스패칭(프로그램 상태 전이)이 SW로 처리된다.

: VIM(Vector Interrupt Manager)모듈을 통해 Legacy 모드와 다르게
인터럽트 디스패칭(프로그램 상태 전이)이 H/W로 처리되고,
ISR으로 분기는 SW로 처리된다.

: Vector Interrupt모드와 다르게 ISR분기가 H/W를 통해 처리되며, CPU의
VIC포트를 통해 인터럽트 요청 시 VIM이 우선순위 처리를 한 후 해당 벡터의
주소를 전달 받은 후 ISR로 분기하여 CPU의 클럭 사이클과 인터럽트 지연을
최소화 한다.

ARM코어의 인터럽트 처리과정

EDDI
Electronic Design Development Institute

➢ TI Hercules 시리즈의 인터럽트

※ Legacy Mode의 인터럽트 처리과정

4.2 *Legacy ARM7 Interrupts*

The flow is visualized in Figure 1, the connection scheme of the peripherals, VIM and R4(F) inside the MCU in Figure 2.

Figure 1. Legacy ARM7 Interrupt Flow

1. Request FIQ/IRQ.
2. Fetch from 0x18/0x1C.
3. Branch to ISR dispatcher.
4. Load Interrupt offset.
5. Decide which ISR to execute.
6. Branch to ISR.

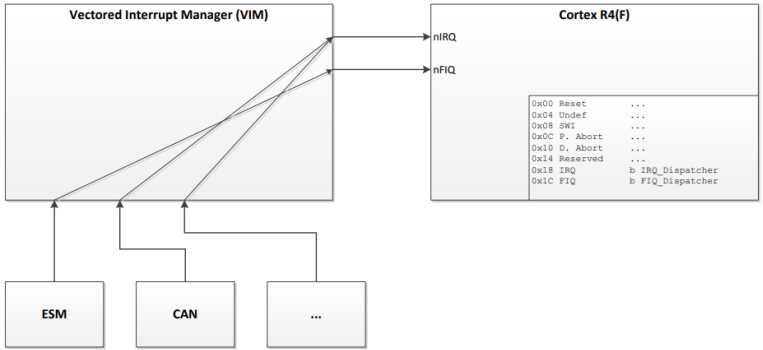


Figure 2. Legacy ARM7 Interrupt Connection Scheme

* Interrupt and Exception Handling on Hercules 참고(https://www.ti.com/lit/an/spna218/spna218.pdf?ts=1642600248661)

ARM코어의 인터럽트 처리과정

EDDI
Electronic Design Development Institute

➢ TI Hercules 시리즈의 인터럽트

※ Vector Interrupt Mode의 인터럽트 처리과정

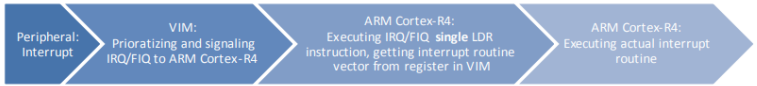


Figure 3. Vectored Interrupt Flow

1. Request FIQ/IRQ.
2. Fetch from 0x18/0x1C
3. Branch to ISR
   (LDR PC, [PC, #-0x1B0]),
   Address for highest active interrupt request derived from IRQVECREG/FIQVECREG.

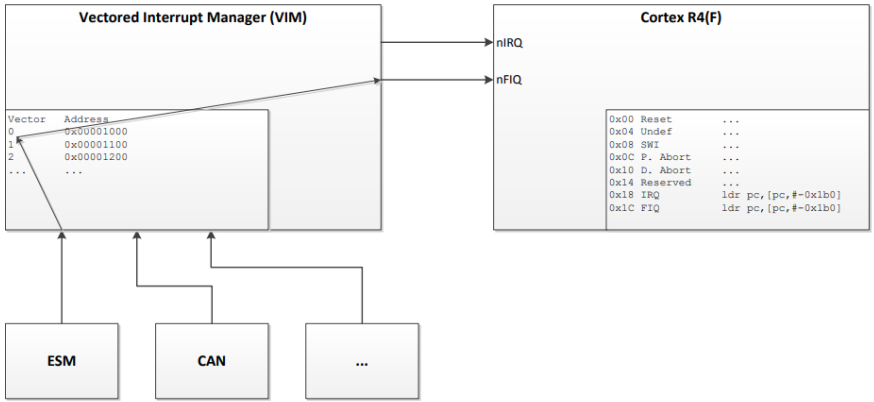


Figure 4. Vectored Interrupt Connection Scheme

* Interrupt and Exception Handling on Hercules 참고(https://www.ti.com/lit/an/spna218/spna218.pdf?ts=1642600248661)

ARM코어의 인터럽트 처리과정

EDDI
Electronic Design Development Institute

- TI Hercules 시리즈의 인터럽트

  ※ H/W Vector Interrupt Mode의 인터럽트 처리과정

4.4 Hardware Vectored Interrupts (only IRQ)

The flow is visualized in Figure 5, the connection scheme of the peripherals, VIM and R4(F) inside the MCU in Figure 6.

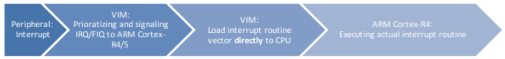


Figure 5. Hardware Vectored Interrupt Flow

1. Request IRQ only.
2. CPU reads IRQ vector address instead of 0x18.
3. VIM provides address of highest pending request directly to the processors VIC port.
4. CPU branches directly to ISR.

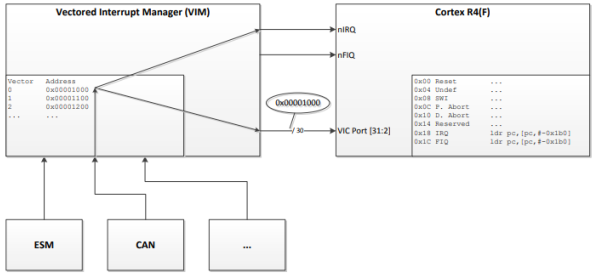


Figure 6. Hardware Vectored Interrupt Connection Scheme

* Interrupt and Exception Handling on Hercules 참고(https://www.ti.com/lit/an/spna218/spna218.pdf?ts=1642600248661)

ARM코어의 인터럽트 처리과정

EDDI
Electronic Design Development Institute

➢ IRQ vs FIQ

Table 2. Processor Registers Organization

		Modes						
		Unprivileged Mode	Privileged Modes					
				Exception Modes				
		USR	SYS	SVC	ABT	UNDEF	IRQ	FIQ
ARM Mode (32 Bit)	Thumb Mode (16 Bit)	R0, A1	R0	R0	R0	R0	R0	R0
		R1, A2	R1	R1	R1	R1	R1	R1
		R2, A3	R2	R2	R2	R2	R2	R2
		R3, A4	R3	R3	R3	R3	R3	R3
		R4, V1	R4	R4	R4	R4	R4	R4
		R5, V2	R5	R5	R5	R5	R5	R5
		R6, V3	R6	R6	R6	R6	R6	R6
		R7, V4, AP	R7	R7	R7	R7	R7	R7
	Can only be accessed via special MOV instructions in Thumb Mode	R8, V5	R8	R8	R8	R8	R8	R8_fiq
		R9, V6	R9	R9	R9	R9	R9	R9_fiq
		R10, V7	R10	R10	R10	R10	R10	R10_fiq
		R11, V8	R11	R11	R11	R11	R11	R11_fiq
		R12, V9, 1P	R12	R12	R12	R12	R12	R12_fiq
	Thumb Mode	R13, SP	R13	R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
		R14, LR	R14	R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
		R15, PC	PC	PC	PC	PC	PC	PC
		CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
				SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq
			Banked Registers					

Table 3. Exception Vector Table

Vector Table Offset	Exception	Mode on Entry	Bits in CPSR		
			A	F	I
0x00	Reset	SuperVisor	Set	Set	Set
0x04	UNDEF	Undefined (Mode)	Unchanged	Unchanged	Set
0x08	SVC	SuperVisor	Unchanged	Unchanged	Set
0x0C	PABT	Abort	Set	Unchanged	Set
0x10	DABT	Abort	Set	Unchanged	Set
0x14	Reserved				
0x18	IRQ	IRQ	Set	Unchanged	Set
0x1C	FIQ	FIQ	Set	Set	Set

(1) IRQ : Normal Interrupt라고 한다.
(2) FIQ : Fast Interrupt로 IRQ보다 인터럽트 처리시간이 빠른 모드이다
(3) 왜 빠른가?

➔ 첫 번째는 위 그림은 FIQ와 IRQ 모드의 사용 레지스터를 보여준다.
IRQ모드에서 사용 할 수 있는 범용 레지스터는 R13,R14 두 개만 존재하고, 따라서 기존 R13, R14의 데이터를 스택에 PUSH하고 ISR이 끝나면
다시 스택에서 POP하는 과정이 필요하나, FIQ는 FIQ모드에서만 사용할 수 있는 범용 레지스터가 7개로 스택에 PUSH, POP하는 별도의 과정이 필요하지 않다

➔ 두 번째 벡터의 주소를 보면 IRQ모드에서 FIQ사이의 공간은 4byte 공간 밖에 안된다. 따라서 IRQ 루틴을 실행하기 위한 프로그램이 저장된 메모리 번지로 분기 명령을 통해 루틴이 실행되지만 FIQ는 벡터 테이블 주소의 마지막으로 0x1C에서 부터 루틴을 실행하기 위한 프로그램이 저장될 수 있기 때문에 별도의 분기 명령에 해당되는 CPU cycle을 절약 할 수 있다.