**Real-Time Interrupt (RTI) Module**

\*이장에서는 실시간 인터럽트(RTI=Real Time Interrupt)모듈의 기능에대해 설명

\*RTI는 실시간 운영체제(RTOS=Real Time Operating system)를 지원하는 운영체제 타이머로설계

<Note>

이 장에서는 DMA(Direct Memory Access)와 관계있는기능 및 Timbase 컨트롤과 관련된 기능을 포함하는 RTI 모듈의 superset 구현에 대해 설명

\* 이러한 기능은 장치 별 기능 내용에 따라 다르다 사용중인 장치에 이러한 기능이 적용되는지 확인하려면 장치 별 데이터 시트를 참조

**17.1 Overview 개요**

\*RTI모듈은 운영 체제 및 벤치마킹 코드 용 타이머 기능을 제공

\*RTI 모듈은 운영 체제에서 스케줄링에 필요한 timebases을 정의하는 여러 카운터를 통합 할 수 있다

\*또한 타이머를 사용하면 원하는 코드 범위의 시작과 끝에서 카운터 값을 읽고 값의 차이를 계산하여 특정 코드 영역을 벤치마킹 할 수 있다

**17.1.1 Features 특징**

RTI모듈에는 다음고 같은 기능이 있다

\*두개의 독립적인 64비트 카운터 블록

\* DMA 요청 또는 운영 체제 틱생성을위한 4 개의 구성 가능한 비교

각 이벤트는 카운터 블록 0 또는 카운터 블록 1에 의해 구동 될 수 있다

\* 이벤트의 빠른 활성화 / 비활성화

각 카운터블록에 한 개, 주변 인터럽트 또는 시스템을위한 두개의 타임스탬프(capture)기능이 있다

\* Digital windowed watchdog

Watchdog : 시스템이 기계적인 고장으로 중단 상태가 되거나 프로그램의 오류로 무제한의 반복(loop) 상태로 들어가는 것을 감시하는 장치

**17.1.2 Industry Standard Compliance Statement** 산업 표준 준수 선언문(성명서)

이 모듈은 OSEK (OSEK / 시간 준수 운영 체제는 물론 OSEK (개방형 시스템 및 자동차 전자 장치 용 개방형 시스템 및 해당 인터페이스)의 요구 사항을 충족하도록 특별히 설계되었지만 이는 그것에 국한되지는 않습니다

**17.2 Module Operation 모듈작동**

그림17-1은 RTI 모듈의 상위 레벨 블록 다이어그램을 설명하고 있다

RTI모듈은 다른 타임베이스를생성하는 2개의 독립적인 카운터블록을 가지고있다

* counter block 0 counter block 1

비교단위는 DMA요청을비교경쟁하거나 4개의 독립적인 인터럽트를 생성하고 프로그램가능한카운터를 비교한다

각 비교 레지스터는 카운터 블록 0 또는 카운터 블록 1과 비교되도록 프로그래밍할수 있다

**17.2.1 Counter Operation 카운터 작동**

각 카운터블록은 다음과 같이 구성됩니다(그림17-2참조)

• 하나의 32 비트 프리 스케일 카운터 (RTIUC0 또는 RTIUC1)  
• 하나의 32 비트 프리 런 카운터 (RTIFRC0 또는 RTIFRC1)

\*RTIUC0 / 1은 RTICLK에 의해 구동되며 비교 업 카운터 레지스터 (RTICPUC0 또는 RTICPUC1)의 비교 값에 도달 할 때까지 카운트업합니다

\*비교가 일치하면 RTIFRC0 / 1이 증가하고 RTIUC0 / 1은 0으로 재설정됩니다. RTIFRC0 / 1이 오버플로되면 인터럽트가 VIM에 생성됩니다

\*오버플로 인터럽트는 운영 체제의 타임베이스를 생성하기위한 것이 아닙니다. 타임베이스 생성은 17.2.2 절을 참조하십시오. 업 카운터와 비교 업 카운트 비교는 RTI 클럭을 사전 설정합니다. 프리 런 카운터 (RTIFRC0 / 1)의 주파수에 대한 결과 공식은 다음과 같습니다

참고 : RTICPUCx를 0으로 설정하는 것은 권장되지 않습니다. 이렇게하면 0xFFFFFFFF에서 0으로 오버플로 한 후 2 개의 RTICLK 사이클 동안 업 카운터가 0으로 유지됩니다.

카운터 값은 각각의 카운터 레지스터를 판독하거나 각각의 캡처 레지스터에 카운터 값을 캡처하는 하드웨어 이벤트를 생성함으로써 결정될 수있다. 두 기능은 다음 섹션에서 설명합니다

**17.2.1.1 Counter and Capture Read Consistency 카운터 및 캡쳐 읽기 일관성**

장치 내부 데이터버스의 일부는 32비트 폭이다 응용 프로그램에서 64 비트 카운터 또는 64 비트 캡처 값을 읽으려면 특정 순서의 32 비트 읽기 작업을 수행해야합니다. 이것은 두 카운터에 대한 두 개의 개별 읽기 작업 사이에서 한 카운터가 증가하는 것을 방지하기위한 것입니다.

**Reading the Counters 카운터 읽기**

프리 러닝카운터 (RTIFRCx)를 먼저 읽어야합니다. 이 우선 순위는 CPU가 RTIFRCx를 읽는 사이클에서 업 카운터 값이 카운터 레지스터 (RTIUCx)에 저장되는 것을 보장합니다.  
두 번째 읽기는 업 카운터 레지스터 (RTIUCx)에 액세스해야하며, 그러면 프리러닝카운터 레지스터 (RTIFRCx)를 읽을 때 경과 된 RTICLK 사이클의 수에 해당하는 값을 유지합니다.

참고 : 업 카운터는 섀도우 레지스터로 구현됩니다. RTIFRCx를 먼저 읽지 않고 RTIUCx를 읽으면 항상 동일한 값을 반환합니다. RTIUCx는 RTIFRCx가 읽을 때만 업데이트됩니다

**Reading the Capture Values 캡처 값 읽기**

프리러닝카운터캡쳐레지스터(RTICAFRCx)를 먼저 읽어야합니다. 이 우선 순위는 CPU가 RTICAFRCx를 읽는 주기에서 업 카운터 값이 카운터 레지스터 (RTICAUCx)에 저장되도록 합니다. 두 번째 읽기는 업 카운터 레지스터 (RTICAUCx)에 액세스해야하며, 그러면 캡쳐카운터레지스터 (RTICAFRCx)를 읽을 때 캡쳐 된 값을 유지합니다.

참고 : 캡처 업 카운터 레지스터는 섀도우 레지스터로 구현됩니다. RTICAFRCx를 먼저 읽지 않고 RTICAUCx를 읽으면 항상 동일한 값을 반환합니다. RTICAUCx는 RTICAFRCx를 읽을 때만 업데이트됩니다.

**17.2.1.2 Capture Feature 캡처 기능**

두 카운터 블록은 외부이벤트 캡쳐기능도 제공한다 두개의 캡쳐소스가 캡쳐이벤트를 트리거 할 수 있다 블록을 트리거하는 소스는 구성가능하다 (RTICAPCTRL 캡쳐컨트롤) 소스는 VIM(Vectored Interrupt Manager)에서 발생하며 주변 모듈이 인터럽트를 생성 할 때 캡쳐이벤트 생성을 허용한다 모든 주변 장치 인터럽트는 VIM의 캡쳐이벤트로 선택 될 수 있다 이벤트가 감지되면 RTIUCx 및 RTIFRCx는캡쳐업카운터 (RTICAUCx) 와캡쳐프리러닝카운터 (RTICAFRCx) 레지스터에 저장된다 캡처 된 값의 읽기 순서는 실제 카운터의 읽기 순서와 동일해야한다

**17.2.2 Interrupt/DMA Requests 인터럽트 / DMA 요청**

DMA 컨트롤러에 대한 VIM 또는 DMA 요청에 대한 인터럽트 요청을 생성하기 위해 4 개의 비교 레지스터 (RTICOMPy)가 있다 인터럽트는 운영 체제에 대해 다른 시간기반을 생성하는 데 사용할 수 있다

각 비교 레지스터는 RTIFRC0(프리러닝카운터) 또는 RTIFRC1과 비교하도록 구성 할 수 있다 이 값은 업데이트 비교레지스터(RTIUDCPy)에 저장되며 비교가 일치하면 추가된다 생성된 인터러브/DMA요청의 기간(시간)은 다음을 사용하여 계산 할 수 있다

RTIFRCx 카운터가 오버플로되는 경우 생성 될 수있는 또 다른 인터럽트는 오버플로 인터럽트 (OVLINTx)이다 인터럽트 / DMA 요청은 RTISETINTENA 레지스터에서 활성화하고 RTICLEARINTENA 레지스터에서 비활성화 할 수 있습니다. RTIINTFLAG 레지스터는 보류중인 인터럽트를 표시합니다.

**17.2.3 RTI Clocking RTI 클럭킹**

카운터블록은 RTICLK로 클럭된다(정의는 2.4.2 절 참조 누르면 클럭도메인으로이동) 운영체제 틱 누락을 방지하기위해 NTUx클럭킹개요를이행하는 클럭감독이 구현된다

**17.2.4 Synchronizing Timer Events to Network Time (NTU)**

**네트워크시간(NTU)을 타이머이벤트시간에 동기화**

시간 트리거 통신 버스에 참여하는 응용 프로그램의 경우 응용 프로그램이나 네트워크 시간의 운영 체제를 동기화하는 것에 종종 도움이 됩니다. RTI는 통신 모듈에 의해 제공되는 주기적 클록에 의해 Free Running Counter 0 (RTIFRC0)을 증가시키는 특징이 있다. 이 경우 두 개의 다른 클럭을 선택할 수 있습니다

이 응용프로그램은 RTIFRC0 클럭에 어떤 클록 (RTICLK, NTU0, NTU1)을 사용해야하는지 제어해준다. NTUx가 사용되면 클럭 감시 회로가 이 클럭을 모니터링 할 수있게하고 클록이 작동하지 않는 경우 대체 솔루션을 제공합니다. 너무 빠르게 실행되는 NTUx는 감지 할 수 없습니다

RTIUC0는 NTUx 신호를 모니터하는데 이용된다. 유효한 NTU 클록 펄스가 발생해야하는 검출 창을 프로그래밍 할 수 있습니다. 펄스가 감지되지 않으면 RTI는 자동으로 다시 작동하여 RTIUC0으로 Free Running Counter 0을 클록합니다. 동작 틱에서 큰 지터를 피하기 위해, RTIUC0로 돌아가는 스위치가 발생하는 경우, RTICPUC0는 클록 주파수 RTIUC0 출력이 NTUx 주파수와 거의 같도록 값으로 설정되어야한다.

**17.2.4.1 Detecting Clock Edges 클럭 엣지 감지**

NTUx 신호에서 클록 에지를 검출하기 위해, 타임베이스 로우 비교는 RTICPUC0 레지스터에 저장된 값보다 낮거나 같게 설정되어야하고 타임베이스 하이 비교는 0보다 높게 설정되어야하고 타임베이스 로우 비교 값보다 낮게 설정되어야합니다.  
이렇게하면 NTUx 신호의 엣지가 예상되는 창을 효과적으로 열 수 있습니다 (그림 17-5 참조).  
이 창 밖에서는 엣지가 감지되지 않습니다. 감지 창 내부에서 엣지가 발생하지 않으면 멀티플렉서는 내부 타임베이스로 전환됩니다.  
응용 프로그램은 타임베이스 인터럽트 (TBINT)를 생성하도록 선택할 수 있으며 INC 비트가 설정된 경우 자동으로 RTIFRC0을 1 씩 증가시켜 NTUx의 누락 된 클록주기를 보완합니다.  
창 내부에서 엣지가 발생하면 RTIUC0가 재설정되어 두 개의 시간축을 동기화합니다.  
엣지 감지가 제대로 작동하게하려면 RTICPU0의 값이 RTIU0이 NTUx와 비슷한주기를 갖도록 조정해야합니다.

참고 : NTUx 신호가 제대로 감지되도록하려면 NTUx 기간은 RTICLK 기간의 두 배 이상이어야합니다.

**17.2.4.2 Switching from Internal Source to External Source**

**내부 소스에서 외부 소스로 전환**

애플리케이션이 내부 소스에서 외부 소스로 전환하는 경우 두 신호를 동기화해야합니다 (그림 17-6 참조). 동기화는 TBEXT 비트가 설정 될 때 발생합니다. RTIUC0는 리셋되고 엣지 검출 회로는 하나의 (RTICPUC0 + RTITBHCOMP) 기간 동안 또는 엣지가 검출 될 때까지 활성화 될 것이다.  
이 기간 동안 펄스가 없으면 소스는 외부 클럭 소스에서 내부 클럭 소스로 재설정됩니다.엣지가 감지되면 윈도우 엣지 감지 동작이 수행됩니다. TBEXT 비트를 설정하면 프리 런 카운터 0이 증가하지 않습니다.

참고 : 외부 타임베이스를 사용하는 경우 소프트웨어는 TBEXT를 외부 소스로 전환하기 전에 타임베이스 로우 비교 및 타임베이스 하이 비교가 유효한 상태로 프로그래밍되도록 보장해야합니다. 이 상태는 타임베이스 제어 회로를 올바르게 작동시키는 데 필요합니다. 다음 조건을 충족해야합니다.  
• RTITBHCOMP < RTITBLCOMP £ RTICPUC0

RTICPUC0에 도달하면 RTIUC0가 재설정되므로 RTITBHCOMP는 RTICPUC0보다 낮아야합니다. RTITBHCOMP는 NTU 엣지가 검출되지 않을 때 회로가 내부 타임베이스로 전환 할 때까지 RTICPUC0에서 RTICLK 사이클 수를 나타낼 것이다.  
외부 타임베이스가 사용되는 경우 타임베이스 제어 회로가 들어오는 신호를 무기한 대기하지 않도록 RTIGCTRL [0]을 1 (RTIUC0 사용)로 설정해야합니다.

**17.2.4.3 Switching from External Source to Internal Source**

**외부 소스에서 내부 소스로 전환**

엣지 감지가 활성화되어 있고 (TBEXT = 1), 프로그래밍 된 감지 창 내에서 NTUx의 클럭 엣지가 감지되지 않으면 RTI는 자동으로 타임베이스를 RTIUC0로 전환합니다.  
그림 17-7은 누락 된 NTU 신호의 타이밍 예를 보여줍니다. 이 경우에는 INC 비트가 설정되고 RTIFRC0은 누락 된 NTU 펄스를 보완하기 위해 자동으로 1씩 증가합니다.  
TBEXT = 0으로 설정하면 RTIFRC0에 대한 클럭 소스가 RTIUC0으로 전환됩니다.

**17.2.5 Digital Watchdog (DWD) 디지털 워치독**

디지털워치독은 런어웨이CPU를 감지하고 리셋 또는 NMI(non-maskable interrupt)를 생성하는 선택적 안전진단이다

프로그램가능기간후에 리셋또는 NMI를생성하거나 올바르지않은 키 시퀀스가 RTIWDKEY레지스터에 기록된다 그림 17-8은 DWD를 보여줍니다.

**17.2.5.1 Digital Watchdog (DWD)**

DWD는 초기에는 비활성화되어있다 사용해야하는 경우 모듈의 고정 배선 코드의 반전 된 값인 32 비트 값을 RTIDWDCTRL 레지스터에 기록하여 활성화해야합니다

참고 : 일단 DWD가 활성화되면 시스템 재설정 또는 재부팅을 제외하고는 비활성화 할 수 없습니다

올바른 키 시퀀스가 RTIWDKEY 레지스터 (0xE51A 다음에 0xA35C)에 기록되면 25 비트 DWD 다운 카운터는 RTIDWDPRLD에 저장된 왼쪽 정렬 된 12 비트 사전로드 값으로 다시로드됩니다.  
잘못된 값이 기록되면 워치 독 재설정 또는 NMI가 즉시 발생합니다. DWD 다운 카운터가 0으로 감소 될 때도 리셋 또는 NMI가 생성됩니다.  
장치가 일시 중지 모드 (디버그 모드 중단)에있는 동안 DWD 다운 카운터는 일시 중지 모드로 들어갈 때의 값을 유지합니다.  
DWD 다운 카운터는 RTICLK 주파수로 감소합니다

DWD 다운 카운터의 만료 시간은 다음 방정식으로 결정할 수 있습니다

texp = (DWDPRLD + 1) × 213/RTICLK

where

DWDPRLD = 0...4095

참고 : RTI가 워치 독 레지스터에 대한 CPU 쓰기가 이루어져 쓴것이 RTI에 전파 될 수 있도록주의해야합니다.

**17.2.5.2 Digital Windowed Watchdog (DWWD)**

17.2.5.1 절에서 논의 된 디지털 감시 장치를 통해 구성 할 수있는 제한 시간 경계 외에도 강화 된 안전 메트릭스의 경우 단일 시간 임계 값을 사용하는 대신 시간 창 내에서 감시 "pet"을 확인하는 것이 바람직합니다.  
이 기능은 DWWD (Digital Windowed Watchdog) 기능에 의해 활성화됩니다.  
• 기능적 행동

DWWD는 워치 독이 서비스되어야하는 구성 가능한 타임윈도우를 연다 이 시간 창 외부에서 워치 독을 서비스하려고 시도하거나이 타임윈도우에서 워치 독을 서비스하지 못하면 워치 독에서 재설정 또는 NMI를 CPU로 생성합니다. 이것은 RTIWWDRXNCTRL 레지스터를 구성하여 제어됩니다. DWD와 마찬가지로 DWWD는 재부팅 후에 비활성화됩니다 DWWD가 윈도우 위반시 non-maskable interrupt (마스크 불가능 인터럽트)를 생성하도록 구성된 경우 워치 독 카운터는 계속 카운트 다운된다

NMI 처리기는 워치 독 위반 상태 플래그를 지우고 정확한 시퀀스를 워치 독 키 레지스터에 기록하여 워치 독에 서비스를 제공해야합니다.  
이 서비스를 통해 워치 독 카운터가 사전로드 값에서 다시로드되고 카운트 다운이 시작됩니다.  
NMI 처리기가 워치 독을 제 시간에 서비스하지 않으면 0으로 줄이고 랩 어라운드 할 수 있습니다.  
NMI 처리기가 시간에 워치 독을 서비스하지 않으면 카운터가 '0'으로 계산 될 때마다 NMI가 연속적으로 생성됩니다.

DWWD는 DWD (Digital Watchdog) 사전로드 레지스터 (RTIDWDPRLD) 설정을 사용하여 창의 종료 시간을 정의합니다.  창의 시작 시간은 창 크기 구성 레지스터 (RTIWWDSIZECTRL)에 의해 정의됩니다.  
  
기본 창 크기는 100 %로 설정되며 타임 아웃 전용 워치 독의 DWD 기능에 해당합니다.  
 그림 17-10에 표시된 것처럼 창 크기는 100 %, 50 %, 25 %, 12.5 %, 6.25 % 및 3.125 % 중에서 선택할 수 있습니다 (RTIWWDSIZECTRL 레지스터를 통해). 해당 크기의 창은 DWD 만료가 끝나기 전에 열립니다. 사용자는 창에서 워치 독을 제공해야합니다. 그렇지 않으면 재설정 또는 NMI가 생성됩니다.  
그림 17-11은 DWWD 작동 예 (25 % 창)를 보여줍니다.  
 • DWWD 구성  
  
DWWD 프리로드 값 (DWD 프리로드와 동일)은 DWWD 카운터가 비활성화 된 경우에만 구성 할 수 있습니다. 워치 독이 활성화 된 후에도 위반에 대한 창 크기 및 워치 독 반응을 구성 할 수 있습니다.  
창 크기 및 워치 독 반응 구성에 대한 변경 사항은 DWWD의 다음 서비스 이후에만 적용됩니다.  
이 기능을 사용하면 작업 실행 시간을 기반으로 다양한 크기의 창을 동적으로 설정하고 오류 범위를 개선 할 수있는 프로그램 시퀀스 요소를 진단에 추가 할 수 있습니다.

**17.2.6 Low Power Modes 저전력 모드**

저전력 모드는 저전력 대 기능 및 빠른 웨이크 업 응답 동안 사용 된 전류를 교환 할 수 있습니다. 모든 저전력 모드에는 다음과 같은 특징이 있습니다.  
• CPU 및 시스템 클록이 비활성화됩니다.  
• 플래시 뱅크 및 펌프가 절전 모드입니다.  
• 모든 주변 장치 모듈이 저전력 모드이고 클록이 비활성화되어 있습니다 (예외가 발생할 수 있으며 특정 장치 데이터 시트에 문서화되어 있음).  
  
클록을 활성화 및 비활성화 할 수있는 유연성으로 다양한 저전력 모드를 구현할 수있다 (2.4.3 절 참조).  
RTI Module의 작동은 Run, Doze 및 Snooze 모드에서 보장됩니다. 절전 모드에서는 모든 클럭이 꺼지고 RTI는 작동하지 않습니다.

Doze 및 스누즈 모드에서 RTI는 활성화되어 있으며 비교, 타임베이스 및 오버플로 인터럽트로 장치를 웨이크 업할 수 있습니다.비교 인터럽트는 장치를 주기적으로 깨우는 데 사용할 수 있습니다. 오버플로 인터럽트는 카운터 오버플로가 발생했음을 운영 체제에 알리는 데 사용할 수 있습니다.  
이러한 저전력 모드 모두에서 주변 모듈이 캡처 이벤트를 트리거 할 수있는 인터럽트를 생성 할 수 있기 때문에 Vectored Interrupt Module (VIM)에 의해 생성 된 이벤트를 캡처하는 것도 가능합니다. 절전 모드에서 이벤트를 캡처하는 것은 RTI에 대한 시계가 활성화되어 있지 않으므로 지원되지 않습니다.  
  
디바이스가 저전력 모드로 들어갈 때, 외부 클록 NTU를 생성하는 주변 장치는 더 이상 활성화되지 않으며, 타임베이스 제어 회로는 NTU에서 누락 된 클록을 검출 할 때 내부 클로킹 구조로 전환해야한다. 타임베이스 인터럽트는 장치를 깨우고 응용 프로그램 소프트웨어는 정기적 인 인터럽트 생성을 내부 클록 소스에 적응시켜야합니다.  
DMA 전송은 비활성화되고 DMA 컨트롤러의 전원이 꺼지기 때문에 장치 깨우기 후에 DMA 요청이 생성되지 않습니다.

참고 : 도스 모드의 RTICLK  
PLL이 꺼진 도즈 모드의 특수한 경우에는 RTICLK가 오실레이터 출력에서 파생되므로 RTICLK는 PLL이 활성화 된 것과 다른 기간을 가질 수 있습니다. VCLK와 RTICLK의 비율은 3 : 1 이상이어야합니다

**17.2.7 Halting Debug Mode Behaviour 디버그 모드 동작 정지**

시스템이 정지 디버그 모드에 들어가면 RTI의 동작은 COS (중단시 계속) 비트에 따라 다릅니다. 비트가 지워지고 디버깅 모드가 중지되면 모든 카운터가 작동을 멈 춥니 다. 비트가 1로 설정되면 모든 카운터가 정상적으로 클럭되며 RTI는 일반 모드와 같이 작동합니다. 그러나 외부 타임베이스 (NTU)를 사용하고 시스템이 디버그 모드를 중지하면 일시 중단 된 통신 컨트롤러의 누락 된 NTU 신호를 감지하면 타임베이스 제어 회로가 내부 타임베이스로 전환합니다. 장치가 디버그 모드를 종료 한 후 소프트웨어를 다시 동기화 할 수 있도록 TBINT 인터럽트로 신호를 보냅니다.