IWDG

2010.06.16

Watchdog?

- Watch + Dog = 쳐다보는 개
- 시스템을 계속 쳐다보고 있는 것
  - 시스템은 무한 루프 같은 상황을 만날 수 있다.
  - 프로그램 상의 오류 및 여러 원인
  - Watchdog은 이러한 상황을 감지해서 시스템을 리셋 시킬 수 있는 기능
- 정상적인 상황에서는 주기적인 동작에 의해서 Watchdog을 계속 초기화 시켜주는 작업을 수행
- 초기화 작업을 할 수 없는 무한 루프와 같은 상황이 발생하면 Watchdog이 초기화 되지 못하고 counter의 값이 0이 되면 Reset이 발생

IWDG, WWDG

- Window watchdog / Independent watchdog

- Independent의 경우는 말 그대로 독립적이라는 의미
- Down counter가 작동하는데 이것이 독립적으로 값이 감소
- IWDG에 인가되는 클럭은 low-speed clock (40 kHz)으로 내부에 기본적으로 들어있는 클럭을 사용 - 항상 active한 상황 유지 가능

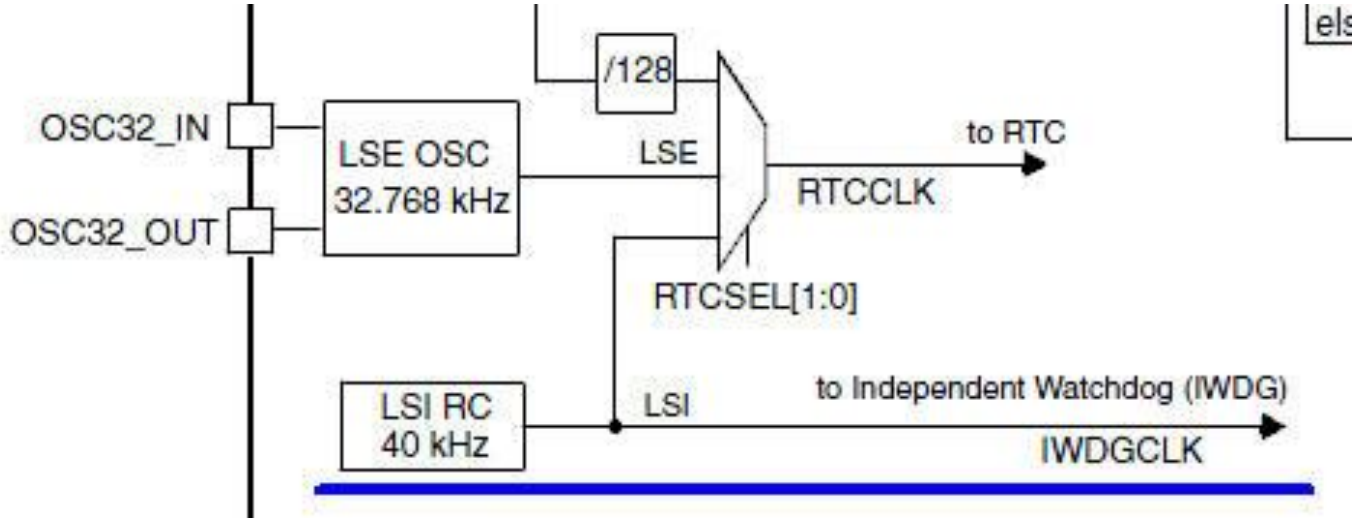
- WWDG는 APB1 clock으로부터 prescale되어서 time-window를 설정
- WWDG는 범위를 주어서 동작하도록 설정하는게 편리하다
- WWDG의 경우는 어떤 주어진 timeout 값에 도달했을 때 인터럽트를 띄워주는 동작을 할 수도 있다.

- IWDG는 main application과는 독립적으로 동작하면서 watchdog을 수행하지만 보다 낮은 timing accuracy를 가지고 수행한다고 할수 있고, WWDG는 보다 정확한 timing window 내에서 watchdog을 수행한다고 볼수 있다.

IWDG 주요 특징

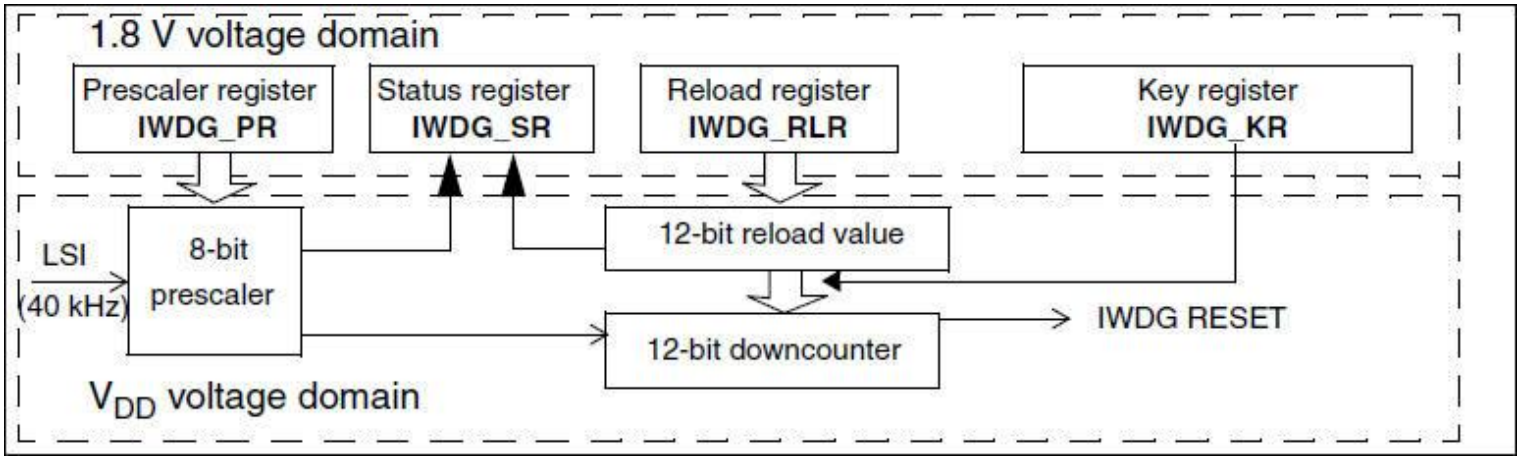
- Down counter가 작동
  - 독립적으로 값이 감소하도록 되어 있다.
  - Independent라는 말의 의미
- 독립적인 내부의 RC oscillator에 의해서 클럭을 공급받는다.
  - 이 클럭은 Standby나 Stop mode에서도 동작될 수 있는 클럭
- Down counter 값이 0에 도달했을 때 시스템을 리셋

IWDG reset



- Standby Low-power mode에 있을 때 깨어날 수 있는 4가지 방법이 있는데 그 중의 하나가 IWDG reset
  - 4가지 방법은 external reset (NRST pin), IWDG reset, rising edge WKUP pin, RTC alarm

IWDG 블록도



- Key register (IWDG_KR) **0xCCCC**
  - independent watchdog 시작
  - counter 리셋 값은 0xFFF – 사용자가 다른 값 설정 가능
  - counter 값이 계속 줄어들면서 0이 되면 reset signal (IWDG reset)
- Key register (IWDG_KR) **0xAAAA**
  - IWDG_RLR value가 재 설정, 그 값부터 다시 counter 값이 줄어든다.
  - IWDG reset이 되기 전에 재 설정 작업을 주기적으로 해주어야 한다.
- Key register (IWDG_KR) **0x5555**
  - IWDG_PR, IWDG_RLR register에 대한 접근은 보호가 되어 있다.
  - 값을 변경하기 위해서는 먼저 IWDG_KR에 0x5555를 적어주어야 한다.

IWDG timeout period

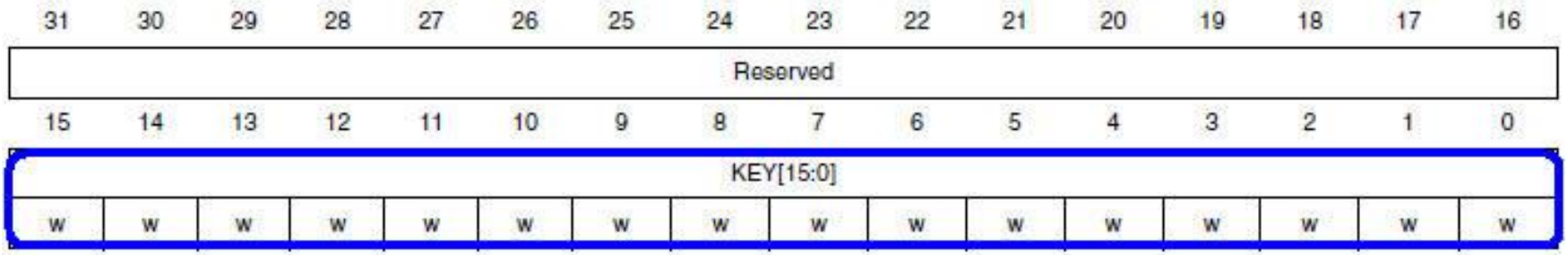
Prescaler divider	PR[2:0] bits	Min timeout (ms) RL[11:0]= 0x000	Max timeout (ms) RL[11:0]= 0xFFF
/4	0	0.1	409.6
/8	1	0.2	819.2
/16	2	0.4	1638.4
/32	3	0.8	3276.8
/64	4	1.6	6553.6
/128	5	3.2	13107.2
/256	6 (or 7)	6.4	26214.4

- 40 kHz input clock 사용 – Watchdog timeout period

예제 코드 시나리오

- SysTick_Handler에서 IWDG counter를 reload하는 작업 수행
- SysTick은 **250 msec의 간격**으로 호출되도록 설정
  - SysTick_Handler에서 IWDG counter를 reload하게 되고 IWDG reset은 발생하지 않게 된다. SysTick_Handler에서 Yellow LED를 계속 toggle하도록 설정해서 문제가 없는 것을 인지할 수 있도록 한다.
- Prescaler divider의 값을 32로 나누는 것을 선택
- timeout으로 선택할 값은 349를 이용해서 값을 설정
  - 하나당 0.8 msec이기 때문에 349를 곱하면 약 **280 msec**
- 문제를 일으키는 방법
  - KEY1을 설정할때 priority를 SysTick 인터럽트보다 높게 설정
  - EXTI 인터럽트 발생 시 pending bit clear 작업을 하지 않는다
  - EXTI 인터럽트가 계속 호출되고 SysTick 인터럽트는 수행 되지 못한다.
- 결국 timeout 값이 다 끝나는 (IWDG counter가 0이 됨) 약 280 msec가 지나고 나면 IWDG reset이 발생
- IWDG reset 발생 이후 리셋이 IWDG reset으로 인해서 발생한 것인지 아니면 최초 Power on으로부터 발생한 것인지를 구분해서 빨간색 LED로 이것을 구분하도록 설정

Key register (IWDG_KR)



- Key register (IWDG_KR)에 저장할 수 있는 값은 3가지 중의 하나
- 이 레지스터는 Write Only. 읽으면 늘 0 이다.

- AAAAh - counter의 값을 재 loading하게 만든다.
- 5555h - IWDG_PR과 IWDG_RLR register에 대한 접근 가능
- CCCCh - watchdog을 시작한다.

EXTI1_IRQHandler

- 정상적인 코드라면 반드시 있어야 하는 것이 바로 아래의 문장

EXTI_ClearITPendingBit(GPIO_EXTI_Line_KEY2);

- 이것은 Pending Register의 해당 비트를 clear해줌으로써 인터럽트가 정상적으로 처리되었다는 것을 알려주는 것
- 만약 이처럼 clear 시켜주지 않게 되면 시스템은 이 인터럽트가 처리되었는지 아닌지를 알지 못하게 되고 마치 또 다른 인터럽트가 발생한 것으로 인지하게 되고 다시 EXTI1_IRQHandler를 호출하게 된다.
- 일부러 이런 상황을 만들기 위해서 위 문장을 수행하지 않은 것

SysTick Clock 설정

```
/* SysTick Re-Configuration */
/* SysTick interrupt each 250ms with counter clock equal to 9MHz */
if (SysTick_Config((SystemFrequency/8) / 4))
{
    /* Capture error */
    while (1);
}
/* Select HCLK/8 as SysTick clock source */
SysTick_CLKSourceConfig(SysTick_CLKSource_HCLK_Div8);
```

- SysTick interrupt 발생하는 간격 250 msec 되도록 한다.
- SysTick_CLKSourceConfig() 설정: HCLK 값을 8로 나눈 값
- SystemFrequency는 72,000,000이고 HCLK 역시 72,000,000
- 이 값을 8로 나눈 값은 9,000,000 입니다. 이것을 4로 나눈 값을 SysTick_Config() 설정하기 때문에 결국 1초에 4번 발생하게 되는 것
- 원하는 250 msec 마다 한번씩 발생하는 상황을 만들 수 있다.

Interrupt Priority 설정

```
/* 1 bits for pre-emption priority and 3 bits for subpriority */
NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);
/* Set Button EXTI Interrupt priority to 0 (highest) */
NVIC_SetPriority(EXTI1_IRQn,
                 NVIC_EncodePriority(NVIC_GetPriorityGrouping(), 0 ,0));
/* Set SysTick interrupt vector Preemption Priority to 1 */
NVIC_SetPriority(SysTick_IRQn,
                 NVIC_EncodePriority(NVIC_GetPriorityGrouping(), 1 ,0));
```

- NVIC_EncodePriority()의 두 번째 파라미터: pre-emption priority
- EXTI1_IRQn는 0으로 SysTick_IRQn은 1로 설정
- 결국 EXTI1_IRQn이 SysTick_IRQn 보다 더 높은 우선순위를 갖도록 한 것이다.