Lecture 10: STM32 Timer/Counters Basics

Seyed-Hosein Attarzadeh-Niaki

Based on slides by ARM, Mazidi

Review

- Using GPIO as External Interrupts
- Interrupt Service Routine
- Timing Analysis
- Program Design with Interrupts

Outline

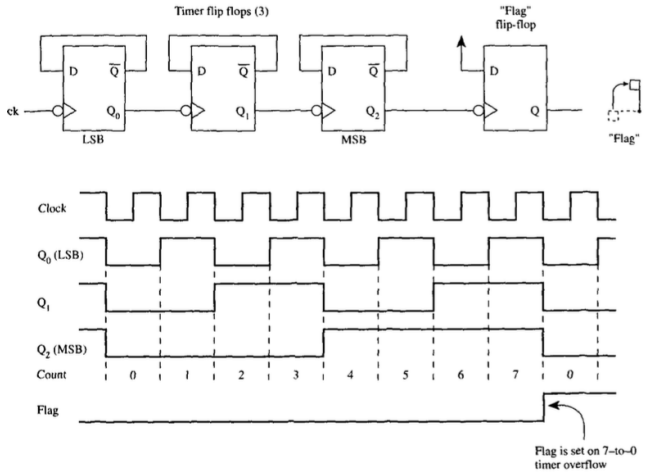
- STM32 timer peripherals
- Timer registers
- Clock source and prescaling
- Upcounting mode and periodic interrupt generation
- Cortex-M4 SysTick timer

Time in Embedded Real-time Systems

- Time is an inherent part of real-time systems
- **Problem**: Instruction-sets do not expose the precise timing of the underlying hardware to the programmer
- **Solutions?**

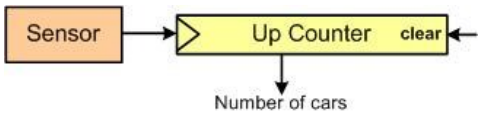
1. Number of executed instructions × cycle time
   - Problems: imprecise timing (especially with modern hardware), inflexible software
2. A dedicated hardware for timing

Simplified Structure of a Timer/Counter

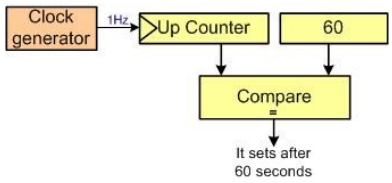


Counter Usages

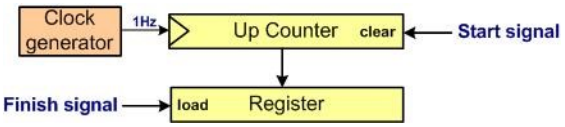
- Counting Events Using a Counter



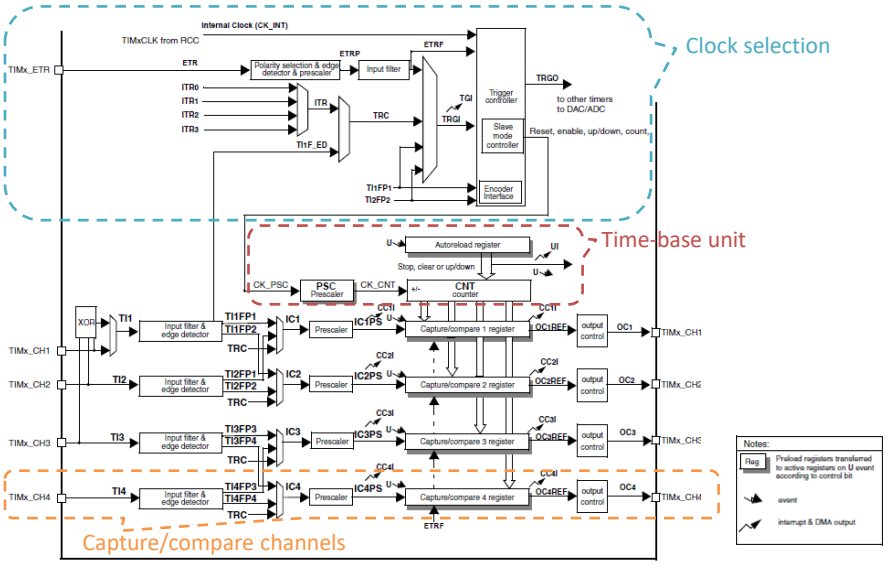
- Using Counter as a Timer



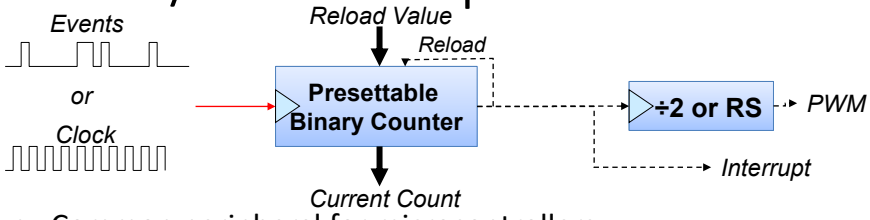
- Capturing



STM32 General Purpose Timers



Timer/Counter Peripheral Introduction



- Common peripheral for microcontrollers
- Based on presettable binary counter, enhanced with configurability
  - Count value can be *read* and *written* by MCU
  - Count direction can often be set to up or down
  - Counter's clock source can be selected
    - **Counter mode**: count pulses which indicate events (e.g. odometer pulses)
    - **Timer mode**: clock source is periodic, so counter value is proportional to elapsed time (e.g. stopwatch)
  - Counter's overflow/underflow action can be selected
    - Generate interrupt
    - Reload counter with special value and continue counting
    - Toggle hardware output signal
    - Stop!

STM32 Timer Peripherals

Advanced Control Timer

- TIM1 and TIM8
- Input capture, output compare, PWM, one pulse mode
- 16-bit auto-reload register
- Additional control for driving motor or other devices

General Purpose Timer

- TIM2 to TIM5
- Input capture, output compare, PWM, one pulse mode
- 16-bit (3,4) or 32-bit (2,5) auto-reload register

General Purpose Timer

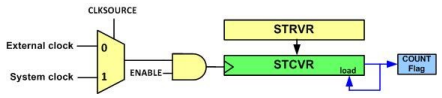
- TIM9 to TIM11 (if available)
- Input capture, output compare, PWM, one pulse mode
- Only 16-bit auto-reload register

Basic Timer (Simple timer)

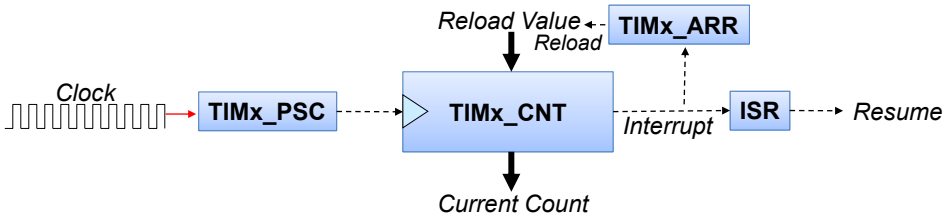
- TIM6 and TIM7 (not available in F401)
- Can be generic counter and internally connected to DAC
- 16-bit auto reload register

Also a 24 bit system timer (SysTick)

Timer type	Timer	Counter resolution	Counter type	Prescaler factor	DMA request generation	Capture/ compare channels	Complementary output	Max. interface clock (MHz)	Max. timer clock (MHz)
Advanced-control	TIM1	16-bit	Up, Down, Up/down	Any integer between 1 and 65536	Yes	4	Yes	84	84
General purpose	TIM2, TIM5	32-bit	Up, Down, Up/down	Any integer between 1 and 65536	Yes	4	No	42	84
	TIM3, TIM4	16-bit	Up, Down, Up/down	Any integer between 1 and 65536	Yes	4	No	42	84
	TIM9	16-bit	Up	Any integer between 1 and 65536	No	2	No	84	84
	TIM10, TIM11	16-bit	Up	Any integer between 1 and 65536	No	1	No	84	84



STM32 General Purpose Timer



- Timer can count down or up or up/down (up by reset)
- A prescaler can divide the counter clock
- 4 independent channels for the timer
  - Input capture
  - Output compare
  - PWM generation
  - One-pulse mode output
- Overflow or underflow will cause an update event (UEV), thus the *interrupt* and possibly the *reload*

General Purpose Timer Control Registers

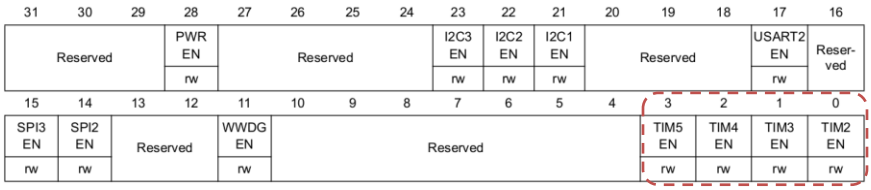
- Highly Configurable

```
typedef struct
{
  __IO uint32_t CR1;         /*!< TIM control register 1,              Address offset: 0x00 */
  __IO uint32_t CR2;         /*!< TIM control register 2,              Address offset: 0x04 */
  __IO uint32_t SMCR;        /*!< TIM slave mode control register,     Address offset: 0x08 */
  __IO uint32_t DIER;        /*!< TIM DMA/interrupt enable register,   Address offset: 0x0C */
  __IO uint32_t SR;          /*!< TIM status register,                 Address offset: 0x10 */
  __IO uint32_t EGR;         /*!< TIM event generation register,       Address offset: 0x14 */
  __IO uint32_t CCMR1;       /*!< TIM capture/compare mode register 1, Address offset: 0x18 */
  __IO uint32_t CCMR2;       /*!< TIM capture/compare mode register 2, Address offset: 0x1C */
  __IO uint32_t CCER;        /*!< TIM capture/compare enable register, Address offset: 0x20 */
  __IO uint32_t CNT;         /*!< TIM counter register,                Address offset: 0x24 */
  __IO uint32_t PSC;         /*!< TIM prescaler,                       Address offset: 0x28 */
  __IO uint32_t ARR;         /*!< TIM auto-reload register,            Address offset: 0x2C */
  __IO uint32_t RCR;         /*!< TIM repetition counter register,     Address offset: 0x30 */
  __IO uint32_t CCR1;        /*!< TIM capture/compare register 1,      Address offset: 0x34 */
  __IO uint32_t CCR2;        /*!< TIM capture/compare register 2,      Address offset: 0x38 */
  __IO uint32_t CCR3;        /*!< TIM capture/compare register 3,      Address offset: 0x3C */
  __IO uint32_t CCR4;        /*!< TIM capture/compare register 4,      Address offset: 0x40 */
  __IO uint32_t BDTR;        /*!< TIM break and dead-time register,    Address offset: 0x44 */
  __IO uint32_t DCR;         /*!< TIM DMA control register,            Address offset: 0x48 */
  __IO uint32_t DMAR;        /*!< TIM DMA address for full transfer,   Address offset: 0x4C */
  __IO uint32_t OR;          /*!< TIM option register,                 Address offset: 0x50 */
} TIM_TypeDef;
```

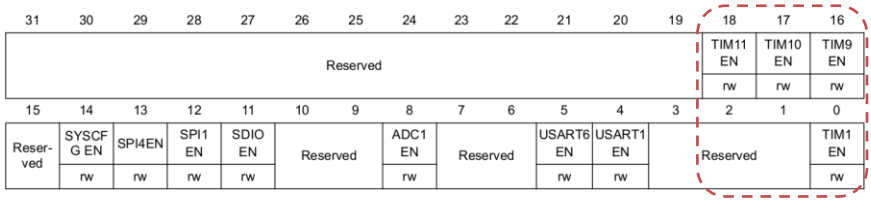
Offset	Register Name
0x00	TIMx_CR1 (Control 1)
0x04	TIMx_CR2 (Control 2)
0x10	TIMx_SR (Status)
0x18	TIMx_CCMR1
0x1C	TIMx_CCMR2
0x20	TIMx_CCER
0x24	TIMx_CNT (Count)
0x28	TIMx_PSC(Presacler)
0x2C	TIM_ARR (Auto Reload)
0x34	TIMx_CCR1
0x38	TIMx_CCR2
0x3C	TIMx_CCR3
0x40	TIMx_CCR4
0x20	TIMx_CCER

Registers Used to Enable Timer Clock

- RCC_APB1ENR

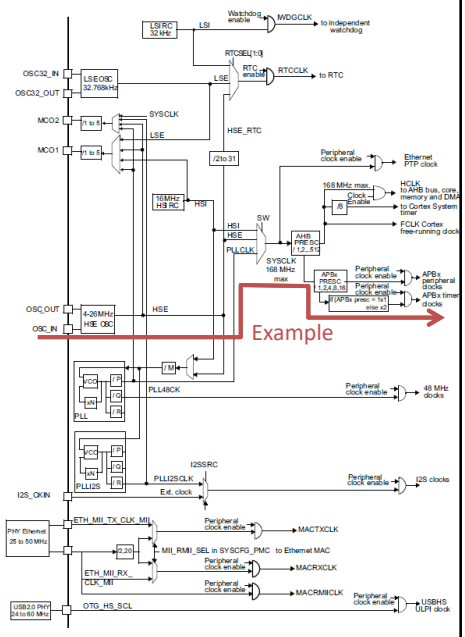


- RCC_APB2ENR



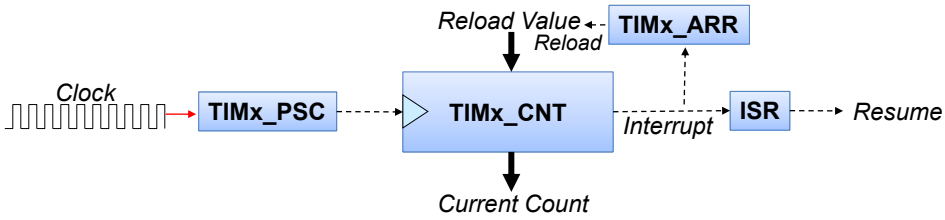
STM32F4 Clock Tree

- Possible to prescale
- Can output clock
- Various configurable System Clock (SYSCLK) sources
  - HSE (High Speed 4MHz to 26MHz External oscillator or crystal)
  - HSI (High Speed 16MHz Internal RC)
  - PLL
- Additional clock sources:
  - LSI (Low Speed 32KHz Internal RC)
  - LSE (Low Speed 32.768kHz External oscillator)



PERIODIC INTERRUPT

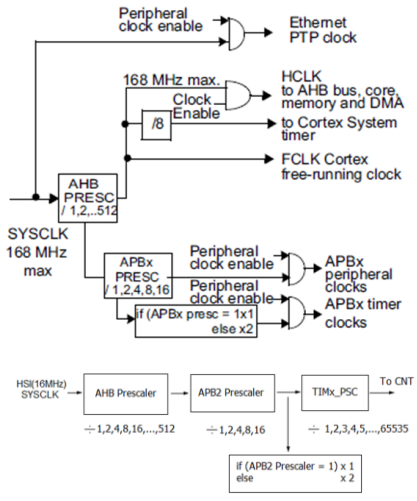
Timer as A periodic Interrupt Source



- STM32F4 families enjoy sophisticated and powerful timer
- One of the basic function of the timer is to cause independent and periodic interrupts
- Best for regularly repeating some certain small tasks

General Purpose Timer Clock Selection

- There are 4 possible sources of clock
  - Internal clock (CK_INT)
  - External clock mode1: external input pin (TIx)
  - External clock mode2: external trigger input (ETR) (some timers only)
  - Internal trigger inputs (ITRx)
- Access TIMx_SMCR (Bit 2:0 SMS) slave mode control register to select the clock source and mode
- For example, for the *periodic interrupt*, with SMS being 000, the counter will be clocked by the internal clock (APB1)
- If the prescaler for APB1 is 4 (defined by the system_stm32f4xx.c) and SYSCLK is 168Mhz, then CK_INT is 42Mhz



Slave Mode Control Register (TIMx_SMCR)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								MSM	TS[2:0]			Res.	SMS[2:0]		
								rw	rw	rw	rw		rw	rw	rw

Bits 2:0 **SMS:** Slave mode selection

When external signals are selected, the active edge of the trigger signal (TRGI) is linked to the polarity selected on the external input (see Input control register and Control register descriptions.

000: Slave mode disabled - if CEN = 1 then the prescaler is clocked directly by the internal clock

001: Reserved

010: Reserved

011: Reserved

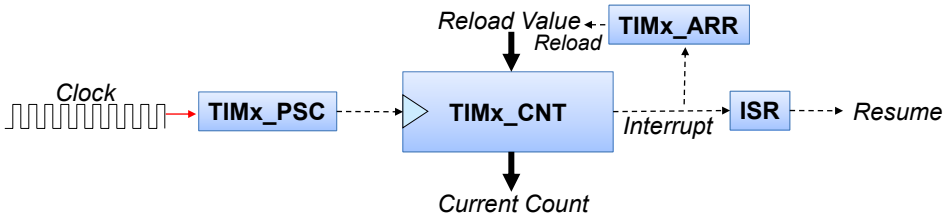
100: Reset mode - Rising edge of the selected trigger input (TRGI) reinitializes the counter and generates an update of the registers

101: Gated mode - The counter clock is enabled when the trigger input (TRGI) is high. The counter stops (but is not reset) as soon as the trigger becomes low. Counter starts and stops are both controlled

110: Trigger mode - The counter starts on a rising edge of the trigger TRGI (but it is not reset). Only the start of the counter is controlled

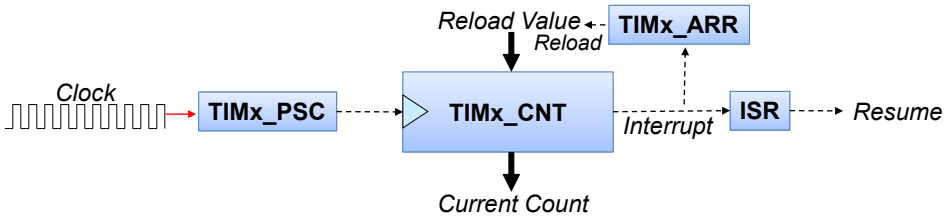
111: External clock mode 1 - Rising edges of the selected trigger (TRGI) clock the counter

Specify PSC and ARR



- TIMx_PSC prescale register stores the value which will be used to divide the clock input.
- In **count up mode**, overflow will occur if TIMx_CNT counter value reach the TIMx_ARR auto-reload value and then the TIMx_CNT will be updated with 0.
- Both TIMx_ARR and TIMx_PSC are 16-bit register!
- The total periodic time can be calculated as
  - $Tout=((ARR+1)\times(PSC+1))\div F_{clk}$
- When F_{clk} is 42MHz, setting ARR to 5999 and PSC to 13999 makes one second periodic time.

Specify PSC and ARR

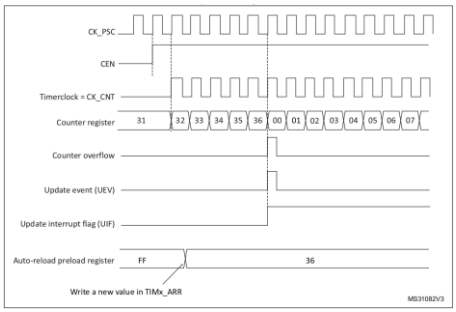


- Accessing the ARR *only write or read from a preload register of ARR*, there is another *shadow register* which is the register that actually performs the reloading.
- The content of the preload register will be transfer to the shadow register permanently *if the APRE bit in TIMx_CR1 is clear or at each UEV if APRE bit is set*.

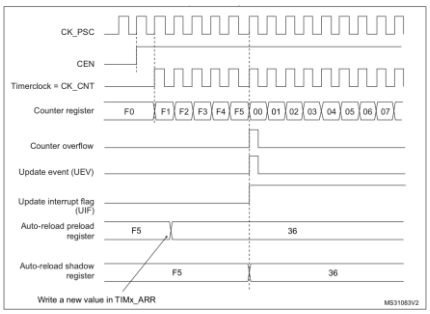
- PSC, on the other hand, is *always buffered*. Which means though it can be changed on the fly, the new ratio will only be taken into account at the next UEV.

APRE Bit in Upcounting Mode

APRE=0



APRE=1



Initialize the TIM2 with CMSIS

- Enable clock to Timer2
  `RCC->APB1ENR|=RCC_APB1ENR_TIM2EN;`
- Set the auto-reload
  `TIM2->ARR=arr;`
- Set the prescaler
  `TIM2->PSC=psc;`
- Enable the update interrupt
  `TIM2->DIER|=TIM_DIER_UIE;`
- Enable counting
  `TIM2->CR1|=TIM_CR1_CEN;`

TIMx Control 1 and Status Registers

- TIMx control register 1 (TIMx_CR1)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						CKD[1:0]		ARPE	CMS		DIR	OPM	URS	UDIS	CEN
						rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

bit	Name	Description
0	CEN	**Counter enable** 0: Counter disabled 1: Counter enabled
3	OPM	**One-pulse mode** 0: Counter is not stopped at update event 1: Counter stops counting at the next update event (clearing the bit CEN)
4	DIR	**Direction** **0**: Counter used as up counter **1**: Counter used as down counter

- TIMx status register (TIMx_SR)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			CC4OF	CC3OF	CC2OF	CC1OF	Reserved		TIF	Res	CC4IF	CC3IF	CC2IF	CC1IF	UIF
			rc_w0	rc_w0	rc_w0	rc_w0			rc_w0		rc_w0	rc_w0	rc_w0	rc_w0	rc_w0

bit	Name	Description
0	UIF	Update interrupt Flag This bit is set by hardware on an update event. It is cleared by software. No update occurred. 0: Update interrupt pending. 1:

Example

- Toggle LED using TIM2
  - Configure TIM2 with prescaler dividing by 1600
  - Counter wraps around at 10000
    - The 16 MHz system clock is divided by 1600 then divided by 10000 and becomes 1Hz
  - Every time the counter wraps around, it sets UIF flag (bit 0 of TIM2_SR register)
  - The program waits for the UIF to set then toggles the LED (PA5).


```
#include "stm32f4xx.h"

int main(void) {
    // configure PA5 as output to drive the LED
    RCC->AHB1ENR |=  1;             /* enable GPIOA clock */
    GPIOA->MODER &= ~0x00000C00;    /* clear pin mode */
    GPIOA->MODER |=  0x00000400;    /* set pin to output mode */

    // configure TIM2 to wrap around at 1 Hz
    RCC->APB1ENR |= 1;              /* enable TIM2 clock */
    TIM2->PSC     = 1600 - 1;       /* divided by 1600 */
    TIM2->ARR     = 10000 - 1;      /* divided by 10000 */
    TIM2->CNT     = 0;              /* clear timer counter */
    TIM2->CR1     = 1;              /* enable TIM2 */

    while (1) {
        while(!(TIM2->SR & 1)) {}   /* wait until UIF set */
        TIM2->SR &= ~1;             /* clear UIF */
        GPIOA->ODR ^= 0x00000020;   /* toggle LED */
    }
}
```

Interrupt Handler

- Enable the interrupts
  ```
  __enable_irq();
  ```
- CMSIS ISR name: TIM2_IRQHandler
  ```
  NVIC_EnableIRQ(TIM2_IRQn);
  ```
- ISR activities
  - Clear pending IRQ
    ```
    NVIC_ClearPendingIRQ(TIM2_IRQn);
    ```
  - Do the ISR's work
  - Clear pending flag for timer
    ```
    TIM2->SR&=~TIM_SR_UIF;
    ```


Example

- LED blinking each sec with Timer TIM2 interrupt
- The system clock is running at 16 MHz.
  - The prescaler is set to divide by 16,000 that gives a 1kHz clock to the counter
  - The counter auto-reload is set to 999
  - When the counter counts to 999, it updates the counter to zero and sets the update interrupt flag (UIF)
  - The UIE bit of TIM2->DIER is set so that the UIF triggers a timer interrupt
  - The interrupt frequency is 1Hz
  - In the timer interrupt handler, the green LED (PA5) is toggled and the UIF is cleared

```c
#include "stm32f4xx.h"

int main(void) {
    __disable_irq();                /* global disable IRQs */
    RCC->AHB1ENR |=  1;             /* enable GPIOA clock */

    GPIOA->MODER &= ~0x00000C00;
    GPIOA->MODER |=  0x00000400;

    /* setup TIM2 */
    RCC->APB1ENR |= 1;              /* enable TIM2 clock */
    TIM2->PSC     = 16000 - 1;      /* divided by 16000 */
    TIM2->ARR     = 1000 - 1;       /* divided by 1000 */
    TIM2->CR1     = 1;              /* enable counter */

    TIM2->DIER |= 1;                /* enable UIE */
    NVIC_EnableIRQ(TIM2_IRQn);      /* enable interrupt in NVIC */

    __enable_irq();                 /* global enable IRQs */

    while(1) {
    }
}

void TIM2_IRQHandler(void) {
    TIM2->SR    = 0;                /* clear UIF */
    GPIOA->ODR ^= 0x20;             /* toggle LED */
}
```


SysTick as Periodic Interrupt Source

- Alternatively, we can use the SysTick provided by Cortex-M4 core to generate exactly the same periodic interrupt.
- The configuration of Systick is however much simpler.
- Four registers:

```c
typedef struct
{
  __IO uint32_t CTRL;                /*!< Offset: 0x000 (R/W)  SysTick Control and Status Register */
  __IO uint32_t LOAD;                /*!< Offset: 0x004 (R/W)  SysTick Reload Value Register       */
  __IO uint32_t VAL;                 /*!< Offset: 0x008 (R/W)  SysTick Current Value Register      */
  __I  uint32_t CALIB;               /*!< Offset: 0x00C (R/ )  SysTick Calibration Register        */
} SysTick_Type;
```

- Since this is part of the core, it is fully supported by CMSIS
  - **`SysTick_Config(uint32_t ticks)`** Initialize the Systick
  - **`SysTick_Config(SystemCoreClock / 1000)`** makes 1ms
  - 168M/1000*(1/f) = 1 ms

Example

- Toggle LED (LD2) using SysTick
- Uses SysTick to generate multiples of millisecond delay
- System clock is running at 16 MHz.
  - SysTick is configured to count down from 16000 to zero to give a 1 ms delay.
  - A for loop counts how many millisecond the delay should be.
  - When 1000 is used for loop count, the delay is 1000ms or 1 second.


```
#include "stm32f4xx.h"

void delayMs(int n);

int main(void) {
    RCC->AHB1ENR |=  1;             /* enable GPIOA clock */
    GPIOA->MODER &= ~0x00000C00;    /* clear pin mode */
    GPIOA->MODER |=  0x00000400;    /* set pin to output mode */

    while (1) {
        delayMs(1000);              /* delay 1000 ms */
        GPIOA->ODR ^= 0x00000020;   /* toggle LED */
    }
}

void delayMs(int n) {
    int i;

    /* Configure SysTick */
    SysTick->LOAD = 16000;  /* reload with number of clocks per millisecond */
    SysTick->VAL  = 0;      /* clear current value register */
    SysTick->CTRL = 0x5;    /* Enable the timer */

    for(i = 0; i < n; i++) {
        while((SysTick->CTRL & 0x10000) == 0) /* wait until the COUNTFLAG is set */
            { }
    }
    SysTick->CTRL = 0;      /* Stop the timer (Enable = 0) */
}
```