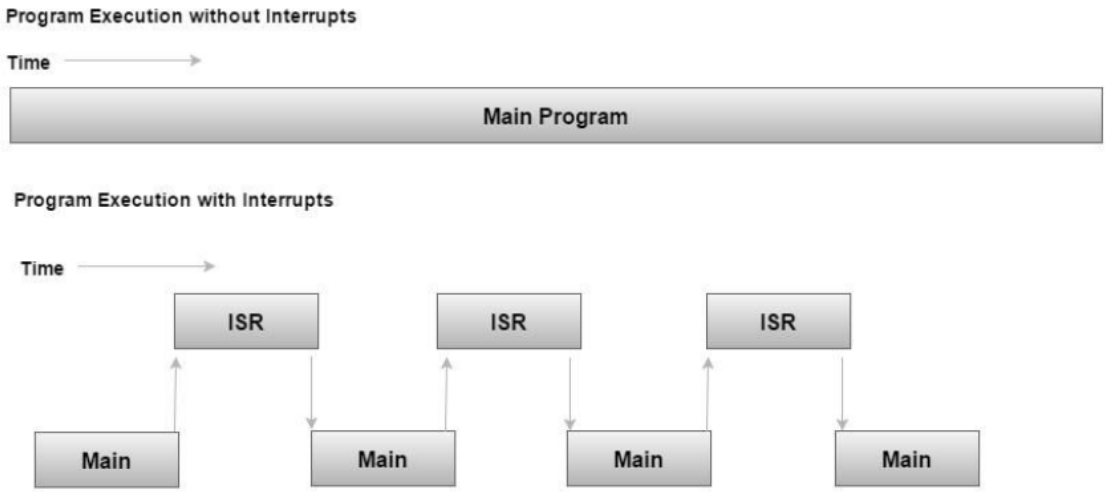
Bài 3: Interrupt – Timer

1.1 Định nghĩa ngắt

- Ngắt là một sự kiện khẩn cấp xảy ra bên trong hoặc bên ngoài vi điều khiển, yêu cầu dừng chương trình chính để thực thi chương trình xử lý ngắt (Trình phục vụ ngắt)



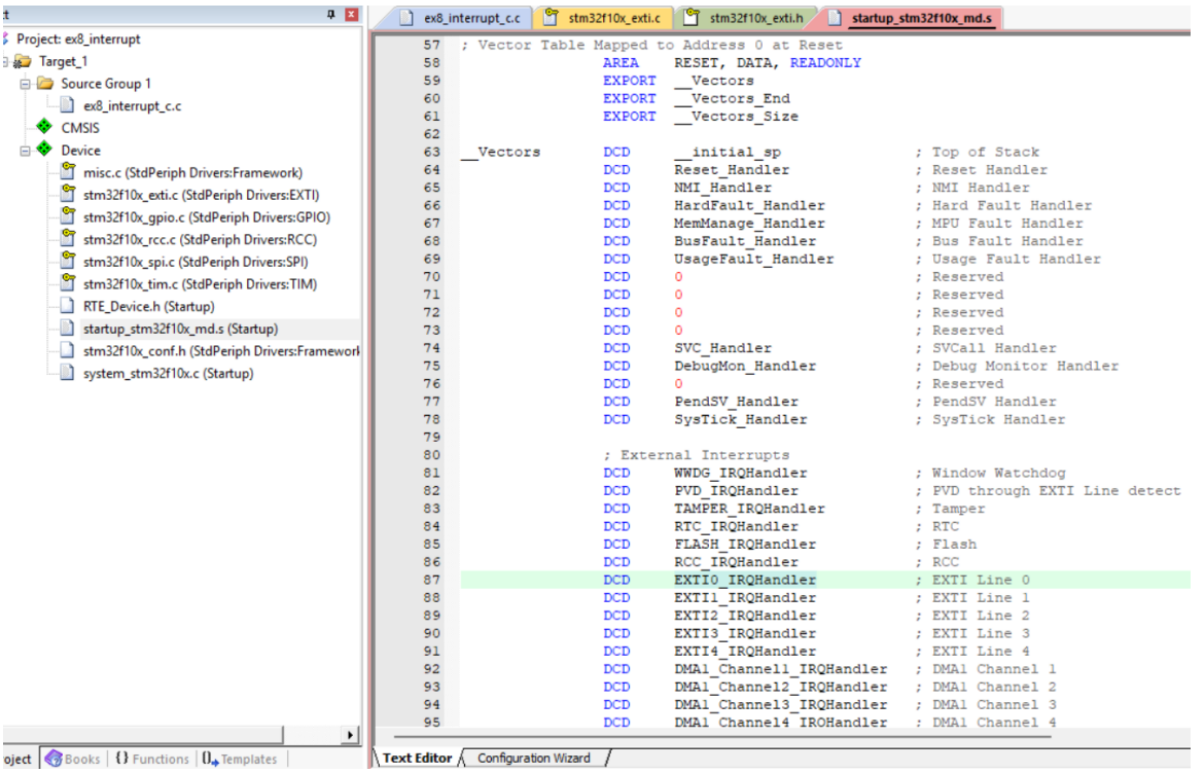
1.2 Các ngắt thông dụng

- Bao gồm ngắt Reset, ngắt ngoài, ngắt Timer1, và ngắt truyền thông.

- Mỗi ngắt sẽ có 1 trình phục vụ ngắt riêng
- **Trình phục vụ ngắt (Interrupt Service Routine - ISR)** là một đoạn chương trình được thực hiện khi ngắt xảy ra.
- Địa chỉ trong bộ nhớ của ISR được gọi là **vector ngắt**

Ngắt	Cờ ngắt	Vector ngắt	Độ ưu tiên ngắt
Reset	-	0000h	-
Ngắt ngoài	IE0	0003h	Lập trình được
Timer1	TF1	001Bh	Lập trình được
Ngắt truyền thông			

Vector ngắt của stm32 lưu trong file Device => startup_stm32f10x_md.s

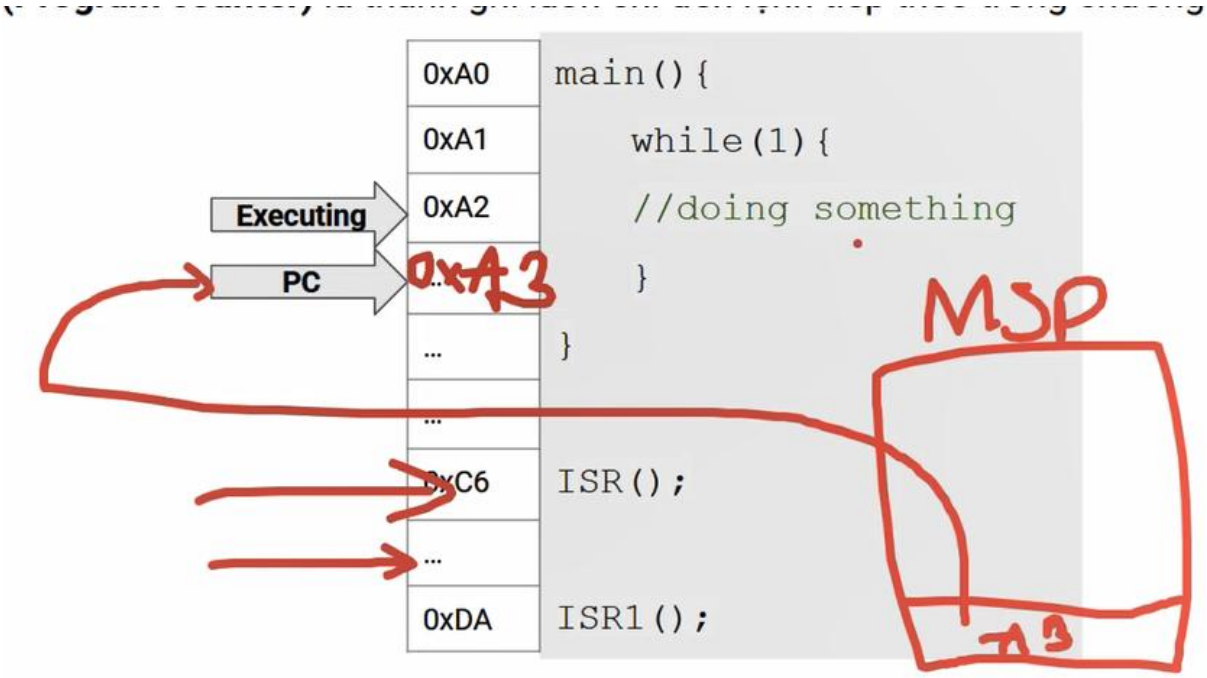


PC (Program counter) là thanh ghi luôn chỉ đến lệnh tiếp theo trong chương trình.

Khi chạy một chương trinh sẽ có thanh ghi lưu địa chỉ của hàm và biến gọi là Main stack Pointer (MSP).MSP chứa tất cả địa chỉ của ISR, MSP hoạt động theo phương pháp LIFO (last in first out). Bởi vậy nó luôn trỏ vào địa chỉ trên đỉnh của MSP

Con trỏ PC trước khi nhảy đến địa chỉ của ISR, nó phải lưu địa chỉ cũ của nó vào MSP (chẳng hạn 0xA3).

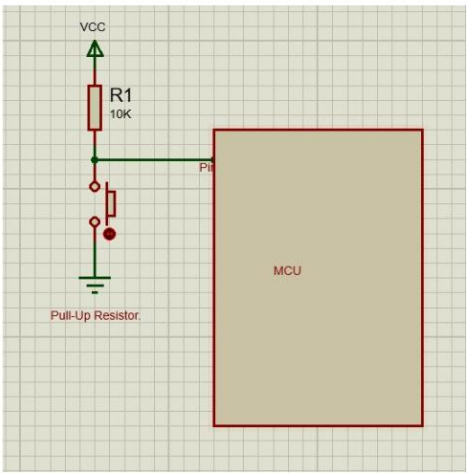
Sau khi nó thực thi dòng lệnh cuối trong ISR, thanh ghi PC lấy địa chỉ trên cùng của MSP và trở về vị trí đó



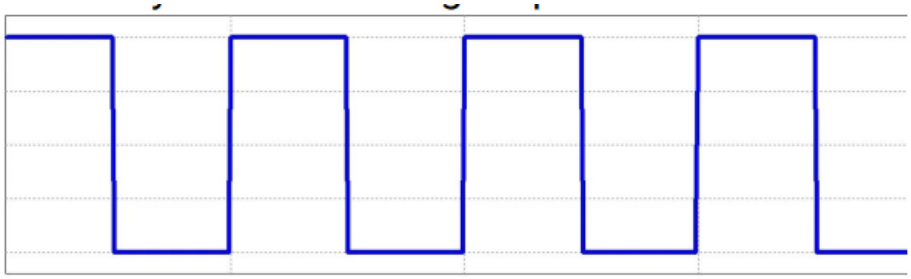
Hàm phục vụ ngắt :

- Mỗi ngắt có địa chỉ trình phục vụ ngắt riêng trong bộ nhớ, được gọi là vector ngắt.

1.2.1 Ngắt ngoài

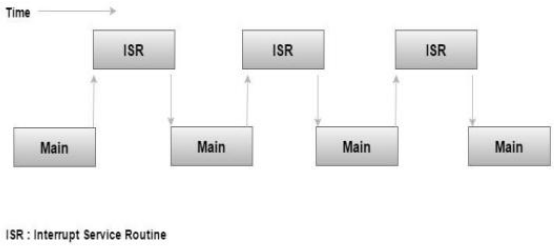


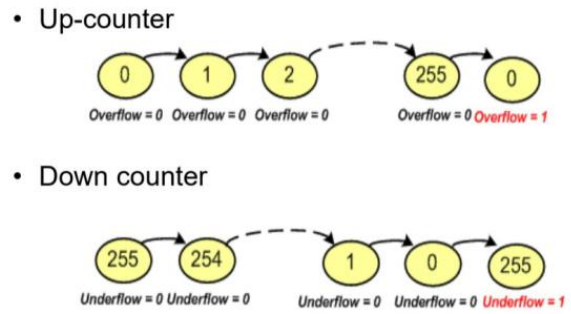
Xảy ra khi có thay đổi điện áp trên các chân GPIO được cấu hình làm **ngõ vào ngắt.** Có 4 dạng:
- **LOW**: kích hoạt ngắt liên tục khi chân ở mức thấp.
- **HIGH**: Kích hoạt liên tục khi chân ở mức cao.
- **RISING**: Kích hoạt khi trạng thái trên chân chuyển từ thấp lên cao.
- **FALLING**: Kích hoạt khi trạng thái trên chân chuyển từ cao xuống thấp.



1.2.2 Ngắt timer

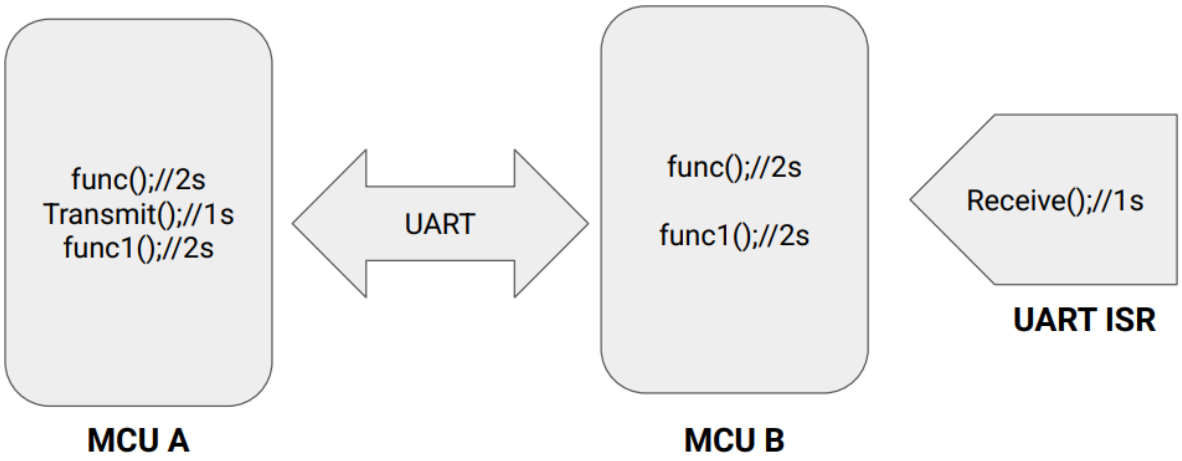
Xảy ra khi giá trị trong thanh ghi đếm của timer bị tràn. Sau mỗi lần tràn, cần phải reset giá trị thanh ghi để có thể tạo ngắt tiếp theo.





1.2.3.Ngắt truyền thông

Xảy ra khi có sự kiện truyền/nhận dữ liệu giữa MCU và các thiết bị khác, thường sử dụng cho các giao thức như UART, SPI, I2C để đảm bảo việc truyền/nhận được chính xác.



1.3.Độ ưu tiên ngắt:

- Các ngắt có độ ưu tiên khác nhau, quyết định ngắt nào được thực thi khi nhiều ngắt xảy ra đồng thời.
- Trên STM32, ngắt có số ưu tiên càng thấp thì có quyền càng cao.
- Độ ưu tiên ngắt có thể lập trình được
- Stack Pointer là thanh ghi trỏ tới đỉnh của vùng stack chứa các địa chỉ trả về của các hàm.

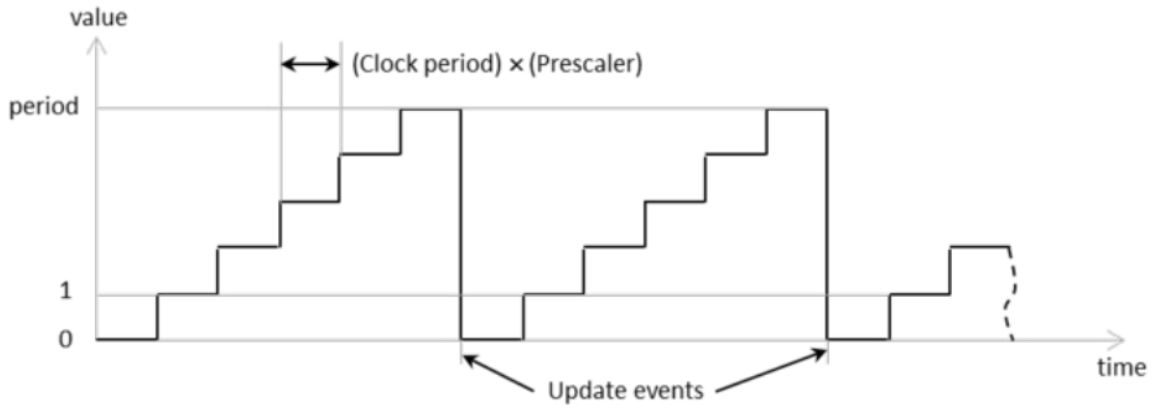
Stack Pointer

0x01 ... 0xA1	main(){}	
0xB2 ... 0xB9	Timer_ISR();	Ưu tiên cao 0
0xD4 ... 0xE2	UART_ISR();	Ưu tiên thấp 1

1.4.Timer (Dùng Đếm, PWM, Định Thời…)

Timer là 1 mạch digital logic có vai trò đếm mỗi chu kỳ clock (đếm lên hoặc đếm xuống). Timer còn có thể hoạt động ở chế độ nhận xung clock từ các tín hiệu ngoài. Ngoài ra còn các chế độ khác như PWM, định thời …vv.

STM32F103 có 7 timer



Timer trong Stm32F1:

Cấu hình Timer:

- Các thông số như Prescaler, Period, và Clock Division được cấu hình để điều chỉnh cách thức đếm của Timer.
- Ví dụ cấu hình timer với thư viện STD.

- CPU sẽ cấp xung với tần số khác nhau cho các ngoại vi hoạt động. Đối với Timer, CPU sẽ cấp cho nó xung có tần số bằng với tần số CPU đang hoạt động.

- CPU của STM32 hoạt động vs xung tần số 72 MHz => Nó sẽ cấp chó Timer đúng tần số 72 MHz để hoạt động. Timer sẽ dùng nó đếm thời gian, đếm sự kiện, so sánh ngõ vào ra.

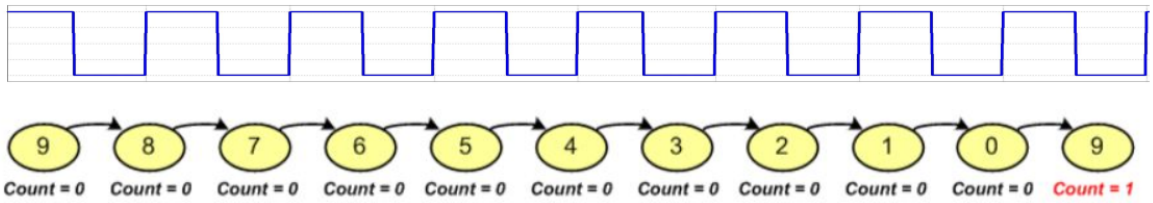
```
typedef struct
{
  uint16_t TIM_Prescaler;          /*!< Specifies the prescaler value used to divide the TIM clock.
                                        This parameter can be a number between 0x0000 and 0xFFFF */

  uint16_t TIM_CounterMode;        /*!< Specifies the counter mode.
                                        This parameter can be a value of @ref TIM_Counter_Mode */

  uint16_t TIM_Period;             /*!< Specifies the period value to be loaded into the active
                                        Auto-Reload Register at the next update event.
                                        This parameter must be a number between 0x0000 and 0xFFFF.  */

  uint16_t TIM_ClockDivision;      /*!< Specifies the clock division.
                                       This parameter can be a value of @ref TIM_Clock_Division_CKD */

} TIM_TimeBaseInitTypeDef;
```



TIM_ClockDivision:

Dùng chia xung mà CPU cung cấp (72Mhz) để cấp cho bộ lọc đầu vào, bộ lọc đầu vào dùng cho những chế độ IC, OC.

Đối với đếm thời gian bình thường sẽ không cần bộ lọc đầu vào. Thường chọn TIM_CKD_DIV1

```
344 /** @defgroup TIM_Clock_Division_CKD
345   * @{
346   */
347
348 #define TIM_CKD_DIV1                     ((uint16_t)0x0000)
349 #define TIM_CKD_DIV2                     ((uint16_t)0x0100)
350 #define TIM_CKD_DIV4                     ((uint16_t)0x0200)
351 #define IS_TIM_CKD_DIV(DIV) (((DIV) == TIM_CKD_DIV1) || \
352                              ((DIV) == TIM_CKD_DIV2) || \
353                              ((DIV) == TIM_CKD_DIV4))
354 /**
355   * @}
```

TIM_Prescaler:

Giống các mức thang đo chiều dài 1cm, 1m…Prescaler thực hiện chia thang đếm cho timer. Tức là bạn chọn bao nhiêu chu kỳ xung clock thì timer đếm lên 1. Số chu kỳ xung clock có thể chọn cho 1 lần đếm là 0x0000 - 0xFFFF. = 0-65535. => moi an dem max = 0.9ms

Thanh ghi TIMx_CNT dùng đếm số chu kỳ xung clock. Ví dụ bạn chọn 3 chu kì clock thì CNT sẽ đếm lên 1.

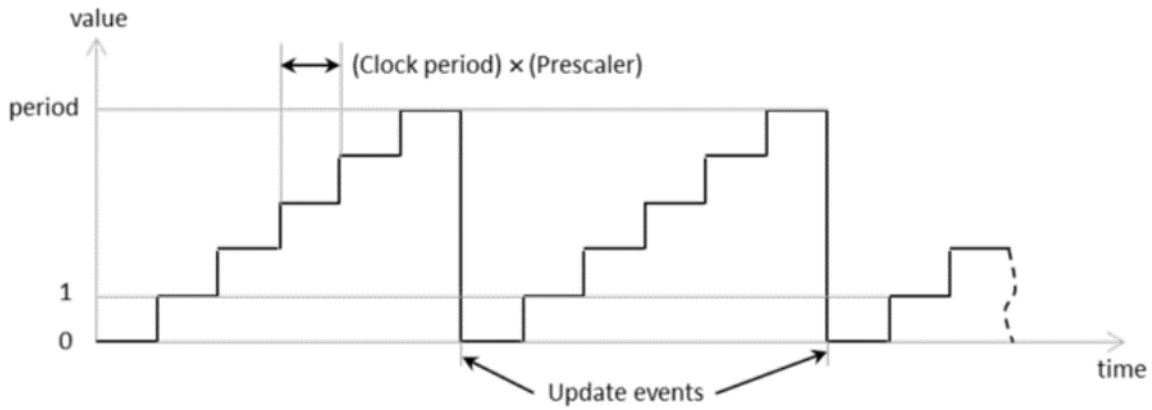
Timer dùng tần số 72MHz => 1 chu kỳ xung clock = 1/(72M)=1/72.000.000 (s)= (1/72) us

Chọn prescaler = 7200 => cứ (1/72)*7200=100 us= 0.1ms thì CNT đếm lên 1 lần.

0x20	TIMx_CCER	Reserved											CC4P	CC4E	CC3NP	CC3NE	CC3P	CC3E	CC2NP	CC2NE	CC2P	CC2E	CC1NP	CC1NE	CC1P	CC1E
	Reset value												0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x24	TIMx_CNT	Reserved											CNT[15:0]													
	Reset value												0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x28	TIMx_PSC	Reserved											PSC[15:0]													
	Reset value												0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TIM_Period:

Thanh ghi CNT Đếm lên bao nhiêu lần thì sẽ tạo ra cờ ngắt Update Event. thanh ghi PC trỏ vào chương trình ngắt ISR. parameter must be a number between 0x0000 and 0xFFFF



TIM_Counter_Mode:

Chọn cách đếm lên hoặc đếm xuống.

```
358 /** @defgroup TIM_Counter_Mode
359   * @{
360   */
361
362 #define TIM_CounterMode_Up                 ((uint16_t)0x0000)
363 #define TIM_CounterMode_Down               ((uint16_t)0x0010)
364 #define TIM_CounterMode_CenterAligned1     ((uint16_t)0x0020)
365 #define TIM_CounterMode_CenterAligned2     ((uint16_t)0x0040)
366 #define TIM_CounterMode_CenterAligned3     ((uint16_t)0x0060)
367 #define IS_TIM_COUNTER_MODE(MODE) (((MODE) == TIM_CounterMode_Up) ||  \
368                                    ((MODE) == TIM_CounterMode_Down) || \
369                                    ((MODE) == TIM_CounterMode_CenterAligned1) || \
370                                    ((MODE) == TIM_CounterMode_CenterAligned2) || \
371                                    ((MODE) == TIM_CounterMode_CenterAligned3))
```

void **TIM_SetCounter**(TIM_TypeDef* TIMx, uint16_t Counter);
// Đặt giá trị ban đầu cho timer
//Chọn timer và cho nó bắt đầu đếm từ đâu

//Được define trong fiel timer.h, triển khai code cụ thể trong file timer.c

timer.c* | stm32f10x_tim.h | stm32f10x_tim.c

```
1120 void TIM_SelectOnePulseMode(TIM_TypeDef* TIMx, uint16_t TIM_OPMode);
1121 void TIM_SelectOutputTrigger(TIM_TypeDef* TIMx, uint16_t TIM_TRGOSource);
1122 void TIM_SelectSlaveMode(TIM_TypeDef* TIMx, uint16_t TIM_SlaveMode);
1123 void TIM_SelectMasterSlaveMode(TIM_TypeDef* TIMx, uint16_t TIM_MasterSlaveMode);
1124 void TIM_SetCounter(TIM_TypeDef* TIMx, uint16_t Counter);
1125 void TIM_SetAutoreload(TIM_TypeDef* TIMx, uint16_t Autoreload);
1126 void TIM_SetCompare1(TIM_TypeDef* TIMx, uint16_t Compare1);
1127 void TIM_SetCompare2(TIM_TypeDef* TIMx, uint16_t Compare2);
1128 void TIM_SetCompare3(TIM_TypeDef* TIMx, uint16_t Compare3);
```

```
2263  void TIM_SetCounter(TIM_TypeDef* TIMx, uint16_t Counter)
2264 {
2265   /* Check the parameters */
2266   assert_param(IS_TIM_ALL_PERIPH(TIMx));
2267   /* Set the Counter Register value */
2268   TIMx->CNT = Counter;
2269 }
2270
2271 /**
2272   * @brief  Sets the TIMx Autoreload Register value
2273   * @param  TIMx: where x can be 1 to 17 to select the TIM peripheral.
2274   * @param  Autoreload: specifies the Autoreload register new value.
2275   * @retval None
2276   */
```

uint16_t **TIM_GetCounter**(TIM_TypeDef* TIMx);

// Lấy giá trị đếm hiện tại của timer

timer.c* | stm32f10x_tim.h | stm32f10x_tim.c

```
1120 void TIM_SelectOnePulseMode(TIM_TypeDef* TIMx, uint16_t TIM_OPMode);
1121 void TIM_SelectOutputTrigger(TIM_TypeDef* TIMx, uint16_t TIM_TRGOSource);
1122 void TIM_SelectSlaveMode(TIM_TypeDef* TIMx, uint16_t TIM_SlaveMode);
1123 void TIM_SelectMasterSlaveMode(TIM_TypeDef* TIMx, uint16_t TIM_MasterSlaveMode);
1124 void TIM_SetCounter(TIM_TypeDef* TIMx, uint16_t Counter);
1125 void TIM_SetAutoreload(TIM_TypeDef* TIMx, uint16_t Autoreload);
1126 void TIM_SetCompare1(TIM_TypeDef* TIMx, uint16_t Compare1);
1127 void TIM_SetCompare2(TIM_TypeDef* TIMx, uint16_t Compare2);
1128 void TIM_SetCompare3(TIM_TypeDef* TIMx, uint16_t Compare3);
1129 void TIM_SetCompare4(TIM_TypeDef* TIMx, uint16_t Compare4);
1130 void TIM_SetIC1Prescaler(TIM_TypeDef* TIMx, uint16_t TIM_ICPSC);
1131 void TIM_SetIC2Prescaler(TIM_TypeDef* TIMx, uint16_t TIM_ICPSC);
1132 void TIM_SetIC3Prescaler(TIM_TypeDef* TIMx, uint16_t TIM_ICPSC);
1133 void TIM_SetIC4Prescaler(TIM_TypeDef* TIMx, uint16_t TIM_ICPSC);
1134 void TIM_SetClockDivision(TIM_TypeDef* TIMx, uint16_t TIM_CKD);
1135 uint16_t TIM_GetCapture1(TIM_TypeDef* TIMx);
1136 uint16_t TIM_GetCapture2(TIM_TypeDef* TIMx);
1137 uint16_t TIM_GetCapture3(TIM_TypeDef* TIMx);
1138 uint16_t TIM_GetCapture4(TIM_TypeDef* TIMx);
1139 uint16_t TIM_GetCounter(TIM_TypeDef* TIMx);
1140 uint16_t TIM_GetPrescaler(TIM_TypeDef* TIMx);
1141 FlagStatus TIM_GetFlagStatus(TIM_TypeDef* TIMx, uint16_t TIM_FLAG);
1142 void TIM_ClearFlag(TIM_TypeDef* TIMx, uint16_t TIM_FLAG);
1143 ITStatus TIM_GetITStatus(TIM_TypeDef* TIMx, uint16_t TIM_IT);
1144 void TIM_ClearITPendingBit(TIM_TypeDef* TIMx, uint16_t TIM_IT);
1145
```

```
2508  uint16_t TIM_GetCounter(TIM_TypeDef* TIMx)
2509  {
2510    /* Check the parameters */
2511    assert_param(IS_TIM_ALL_PERIPH(TIMx));
2512    /* Get the Counter Register value */
2513    return TIMx->CNT;
2514  }
2515
2516  /**
2517    * @brief  Gets the TIMx Prescaler value.
2518    * @param  TIMx: where x can be 1 to 17 to select the TIM peripheral.
2519    * @retval Prescaler Register value.
2520    */
2521  uint16_t TIM_GetPrescaler(TIM_TypeDef* TIMx)
```

Ví dụ: Hàm delay_ms:

```
// Delay function
void delay_ms(uint8_t timedelay)
{
TIM_SetCounter(TIM2,0);
while(TIM_GetCounter(TIM2) < timedelay * 10){}
}
```

```
// Chuong trinh nhay led pin13 tren stm32, tan suat nhay 100s/lan.
#include "stm32f10x.h" // Device header
#include "stm32f10x_rcc.h" // Keil::Device:StdPeriph Drivers:RCC
#include "stm32f10x_gpio.h" // Keil::Device:StdPeriph Drivers:GPIO
#include "stm32f10x_tim.h" // Keil::Device:StdPeriph Drivers:TIM
void RCC_Config()
{
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE); // Cap clock cho TIM2,
clock nay tan so 72MHz
}
void GPIO_Config()
{
GPIO_InitTypeDef GPIO_InitStruct;
GPIO_InitStruct.GPIO_Pin = GPIO_Pin_13;
GPIO_InitStruct.GPIO_Mode = GPIO_Mode_Out_PP;
GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOC,&GPIO_InitStruct);
}
void TIM_Config() // Cau hinh TIM de moi lan dem duoc 0.1ms, tuc tao thang
chia 0.1ms
{
TIM_TimeBaseInitTypeDef TIM_InitStruct;
TIM_InitStruct.TIM_ClockDivision = TIM_CKD_DIV1; // chia nho xung do CPU cap
cho. o day chia 1
TIM_InitStruct.TIM_Prescaler = 7200 - 1; // Dem 7200 xung clock thi timer dem
len 1. 7200 *1/72M = 100 us =0.1ms. 7200-1 do timer dem tu 0x0000
TIM_InitStruct.TIM_Period = 0xFFFF; // chon enough de ngat khong xay ra.
```

```
TIM_InitStruct.TIM_CounterMode = TIM_CounterMode_Up; // Dem len
TIM_TimeBaseInit(TIM2,&TIM_InitStruct); // Nap cau hinh vao TIM2
TIM_Cmd(TIM2,ENABLE); // Cho phep TIM2 hoat dong
}
void delay_ms(uint32_t time)
{
TIM_SetCounter(TIM2,0); // set TIM2 bat dau dem tu 0
    while(TIM_GetCounter(TIM2)< time * 10) {} // khi nao dem den time*10 thoat
khoi vong lap while. note : 10 x 0.1ms = 1s.
}
int main()
{
RCC_Config();
GPIO_Config();
TIM_Config();
while(1){
GPIO_ResetBits(GPIOC,GPIO_Pin_13);
delay_ms(100);
GPIO_SetBits(GPIOC,GPIO_Pin_13);
delay_ms(100);
}
}
```