The STM32F407VG has 14 timers that are classified as Advanced Control Timers (2), General Purpose (4 are either 16-bit or 32-bit timers and 6 are 16-bit timers), and Basic Timers (2).

**Advanced Control Timers:** Timers 1 and 8. These are 16-bit auto-reload counters with a prescaler. They can do input capture, output compare, PWM, and one-pulse mode output.

**General Purpose Timers:** Timers 3 and 4 are 16-bit timers and Timers 2 and 5 are 32-bit timers. All timers in this group have a 16-bit prescaler and auto-reload. They can count up or down. They can do input capture, output compare, PWM, and one-pulse mode output. Their unique feature is that they can do an interrupt for DMA for some events.

![General-purpose timer block diagram](image_url)

**Figure 1**
General purpose timers block diagram.
May 15, 2015

**General Purpose Timers:** Timers 9 to 14 are 16-bit up-counters with a prescaler and auto-reload. The can do input capture, output compare, PWM, and one-pulse mode output. They can also do interrupt generation for some events.

**Basic Timers:** Timers 6 and 7. These are 16-bit timers with a prescaler and an auto-reload register. Their unique feature is that they can be used to trigger a D/A conversion for waveform generation.

The timer registers are similar for all of the timers but there are too many of them to list here. See the User's Manual\(^1\) pp. 507-690 for complete details on all of the registers and modes. Table 1 shows the registers use for Timers 2, 3, 4, and 5.

**Example 1**
Use Timer 2 as a 32-bit timer in the polled mode to output a pulse train on port PA7 that has a 90 μsec high time and a 90μsec low time. The timer is used in the one-pulse mode with auto-reload. When it times out it triggers the CEN bit off which disables the timer. The program polls this bit and toggles output bit PA7 each time CEN goes to 0. It then sets CEN back to 1 for the next pulse.

**Example 2**
Use Timer 2 as a 32-bit timer in the interrupt mode to output a pulse train on port PA7 that has a 90 μsec high time and a 90 μsec low time. The timer is used in the one-pulse mode with auto-reload. When it times out it triggers an interrupt which sets a flag allowing bit PA7 to be toggled.

For reasons that remain mysterious, the interrupt registers are not listed in the User's Manual. They are given in the Programmer's Manual\(^2\) pp. 193-201.

---


<table>
<thead>
<tr>
<th><strong>Register</strong></th>
<th><strong>Mnemonic</strong></th>
<th><strong>Function</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Control Register 1</td>
<td>TIMx_CCR1</td>
<td>Main control register. Has enable, one-pulse mode, direction, and some PWM</td>
</tr>
<tr>
<td>Control Register 2</td>
<td>TIMx_CCR2</td>
<td>Master mode selection bits and bits for external connections</td>
</tr>
<tr>
<td>Slave mode control register</td>
<td>TIMx_SMCR</td>
<td>Allows the timer to be triggered as a result of events in other timers.</td>
</tr>
<tr>
<td>DMA/Interrupt enable register</td>
<td>TIMx_DIER</td>
<td>Interrupt and DMA enable bits</td>
</tr>
<tr>
<td>Status register</td>
<td>TIMx_SR</td>
<td>Has a trigger interrupt flag</td>
</tr>
<tr>
<td>Event generation register</td>
<td>TIMx_EGR</td>
<td>Used with capture mode to generate various events such as interrupts</td>
</tr>
<tr>
<td>Compare/Capture mode registers</td>
<td>TIMx_CCMR1,</td>
<td>These registers are used to set up the timer to capture external events.</td>
</tr>
<tr>
<td></td>
<td>TIMx_CCMR2</td>
<td></td>
</tr>
<tr>
<td>Compare/Capture re-enable register</td>
<td>TIMx_CCER</td>
<td></td>
</tr>
<tr>
<td>Capture Control register 1 to 4</td>
<td>TIMx_CCR1,</td>
<td>This is the timers count value</td>
</tr>
<tr>
<td></td>
<td>2, 3 &amp; 4</td>
<td></td>
</tr>
<tr>
<td>Counter register</td>
<td>TIMx_CNT</td>
<td></td>
</tr>
<tr>
<td>Prescale register</td>
<td>TIMx_PSC</td>
<td>This value prescales the timer clock</td>
</tr>
<tr>
<td>Auto-reload register</td>
<td>TIMx_ARR</td>
<td>This value is automatically reloaded into the timer when it runs out</td>
</tr>
<tr>
<td>DMA Control register</td>
<td>TIMx_DCR</td>
<td>DMA Control. Burst length and base address vector</td>
</tr>
<tr>
<td>DMA Address register</td>
<td>TIMx_DMAR</td>
<td>DMA address is loaded here</td>
</tr>
<tr>
<td>Timer 2 option register</td>
<td>TIM2_OR</td>
<td>Internal trigger options</td>
</tr>
<tr>
<td>Timer 5 option register</td>
<td>TIM5_OR</td>
<td>Internal trigger options</td>
</tr>
</tbody>
</table>
The interrupt set/enable register (NVIC_ISERx) is a 32-bit register which has enable bits for each interrupt. Since there are more than 32 interrupts available there are multiple registers named NVIC_ISER1, NVIC_ISER2, etc. The ISER1 register has bits for interrupts 0 to 31 corresponding to bits 0 to 31. The ISER2 register has bits for interrupts 32 to 63 which correspond to bits 0 to 31. Int 32 is bit 0 in ISER1, int 33 is bit 1 in ISER1 etc.

Table 61 on pp. 369-371 of the User's Manual gives the interrupt numbers (called positions in the table) for the STM32F407 processor. From that table we see that the Timer 2 interrupt is in position 18 which is bit 28 of ISER0.

Three things need to be set up to enable Timer 2 to generate an interrupt: 1) Set bit 28 to 1 in ISER0 to enable the interrupt to happen and 2) Set bit 0 of the Timer 2 DMA/Interrupt Enable Register (DIER) to allow the timer to generate an interrupt on overflow, and 3) Set bit 6 in DIER to 1 to enable the trigger interrupt.

```c
NVICISER0 |= (1 << 28); //Bit 28 in ISER0 corresponds to int 28 (TIM 2)
TIM2_DIER |= 1;        //Enable Timer 2 update interrupt enable
TIM2_DIER |= (1 << 6); //Enable Timer 2 trigger interrupt enable
```

We also need to write the interrupt service routine. The service routine in this example simple sets a flag bit to zero. The other thing is must do is to turn off the interrupt. Note that the flag bit has been declared globally so that it is available to both the interrupt service routine and the main program. On most systems that use interrupts the compiler also requires that you load the interrupt vector. This vector tells the system where the interrupt service routine is located. For this particular processor that is not necessary but you must name the service routine by a particular name. For timer 2 that name is `TIM2_IRQHandler`. This routine needs no prototype since it is not called by the main program. The name of the interrupt handler is given in Table 61 on pp. 361-372 in the User's Manual.

```c
void TIM2_IRQHandler()
{
    flag = 0;
    TIM2_SR &= 0xFFFF;            //Turn off interrupt
}
```

Timer 2 is a 32-bit timer so we can run the clock without prescaling and load any number from 1 to \(2^{32}-1\) into the Auto-Reload Register. The counter counts up from zero until it reaches the reload register at which time it can trigger the interrupt.

Examples 1 and 2 can be easily converted to use timers 3 or 4 which are 16-bit timers with 16-bit prescalers or it can be changed to use timer 5 which is also a 32-bit timer.
**EXAMPLE 1**

//TimerPoll.c
/* This program uses Timer 2 in a polled mode to toggle bit PA7
about every 90 useconds. The timer is programmed in the
one pulse mode. After each pulse it is restarted.
The timer is uses autoreloading
*/
#include "stm32f407vg.h"

int main()
{
    int tmp;
    //Clock bits
    RCC_AHB1ENR |= 1;         //Bit 0 is GPIOA clock enable bit
    RCC_APB1ENR |= 1;         //Enable peripheral timer for timer 2 (bit 0)
    //I/O bits
    GPIOA_MODER |= 0x4000;    //Bits 15-14 = 01 for digital output on PA7
    //OTYPER register resets to 0 so it is push/pull by default
    GPIOA_OSPEEDER |= 0xC000;  //Bits 15-14 = 11 for high speed on PA7
    //PUPDR defaults to no pull up no pull down
    //Timer 2 bits
    TIM2_CR1 |= (1 << 7);   //Auto reload is buffered
    TIM2_CR1 |= (1 << 3);   //One pulse mode is on.
    TIM2_PSC = 0;           //Don't use prescaling
    TIM2_ARR = 7619;        //(168 MHz/2)/7619 = 11025 Hz
    TIM2_ARR = 84000000;    //(168 MHz/2)/84000000 = 1 Hz
    TIM2_CR1 |= 1;          //Enable Timer 2
    //Main program loop
    tmp = 0;
    while(1)
    {
        GPIOA_ODR = tmp;            //Only PA7 is set up for output so other bits
        tmp = ~tmp;                 // have no effect.
        while((TIM2_CR1 & 1) != 0); //Wait here until timer runs out
        TIM2_CR1 |= 1;              //Restart timer
    }
}
**EXAMPLE 2**

//TimerInt2.c

/* This program uses Timer 2 in an interrupt mode to toggle bit PA7
   about every second. The timer is uses autoreloading. */

#include "stm32f407vg.h"

int flag;
int main()
{
    int tmp;
    //Clock bits
    RCC_AHB1ENR |= 1;         //Bit 0 is GPIOA clock enable bit
    RCC_APB1ENR |= 1;         //Enable peripheral timer for timer 2 (bit 0)
    //Interrupt bits
    NVICISER0 |= (1 << 28);   //Bit 28 in ISER0 corresponds to int 28 (TIM 2)
    TIM2_DIER |= 1;           //Enable Timer 2 update interrupt enable
    TIM2_DIER |= (1 << 6);    //Enable Timer 2 trigger interrupt enable
    //I/O bits
    GPIOA_MODER |= 0x4000;    //Bits 15-14 = 01 for digital output on PA7
    //OTYPER register resets to 0 so it is push/pull by default
    GPIOA_OSPEEDER |= 0xC000; //Bits 15-14 = 11 for high speed on PA7
    //PUPDR defaults to no pull up no pull down
    //Timer 2 bits
    TIM2_CR1 |= (1 << 7);    //Auto reload is buffered
    TIM2_PSC = 0;           //Don't use prescaling
    TIM2_ARR = 84000000;    //168 MHz/2)/84000000 = 1 Hz
    TIM2_CR1 |= 1;          //Enable Timer 2
    //Main program loop
    tmp = 0;
    while(1)
    {
        GPIOA_ODR = tmp;            //Only PA7 is set up for output so other bits
        tmp = ~tmp;                 // have no effect.
        flag = 1;
        while(flag == 1);          //Restart timer
    }
}

void TIM2_IRQHandler()
{
    flag = 0;
    TIM2_SR &= 0xFFFF;            //Turn off interrupt
}