The Timers of the STM32 Microcontrollers

Corrado Santoro

**ARSLAB - Autonomous and Robotic Systems Laboratory**
Dipartimento di Matematica e Informatica - Università di Catania, Italy
santoro@dmi.unict.it

L.S.M. Course
What is a “timer”?:

- A **TIMER** is a circuit that enables the software to have the “knowledge of time”

- It is basically a **global variable** (timer counter) that **increments** (or **decrements**) on the basis of a **programmable clock source**

- The **global variable** (timer counter) can be read or written by the software

- A timer can generate **interrupts**

- A timer can be used by a **slave circuit**:
  - to generate particular periodic signals
  - to measure the period or pulse of input signals
The hardware of **TIMER** is composed by three basic programmable parts:

- The **clock source**, the circuit that generates the clock tick for the timer
- The **time base**, the circuit that derive the *time granularity* from the clock source and contains the **timer counter variable**
- The **slave circuits**, that provide additional functions (pulse measure, signal generation, etc.) by exploiting the timer variable
STM32 MCUs offer up to 11 different timer/counters with the following features:

- Clock selection (internal, external, other)
- 16/32-bit counter resolution
- Programmable prescaler
- Four independent channels configurable as:
  - Input Capture
  - Output Compare
  - PWM Mode
  - One-pulse Output
- Interrupt generation on the basis of the various events that can occur
Each timer has several special function registers

All of them are accessible by means of global variables called `TIMx`, where `x` is the number of the timer (TIM1, TIM2, ...)

The type of these variables is `TIM_TypeDef *`, i.e. pointers to a structure whose field are the SFR of the timer
Clock source can be:

- **Internal** (System Peripheral Clock, default setting)
- **External** (External Pin)
- **External in QEI mode** (Quadrature-encoder interface)
- Several Gate/Trigger inputs can be configured in order to start/stop the clock on the basis of events
Counting is handled in the time-base by the following registers:

- **TIMx→PSC**: the prescaler register, it directly specified the division factor
- **TIMx→CNT**: the counter register, it holds the counter value and increments (or decrements) according to the input clock
- **TIMx→ARR**: the auto-reload register, **CNT** counts from 0 to **ARR**, then **CNT** is set to 0 again
- When **CNT** is reloaded an update event is generated (the “U” in figure), that can trigger interrupt generation
Note: Timer functions of stm32_unict_lib currently support timers from TIM2 to TIM5 (but TIM5 is also used by the display)

- **Initialize a TIMER:**
  ```c
  void TIM_init(TIM_TypeDef * timer);
  ```

- **Configure the timebase:**
  ```c
  void TIM_config_timebase(TIM_TypeDef * timer,
                          uint16_t prescaler,
                          uint16_t autoreload);
  ```

- **Start a timer:**
  ```c
  void TIM_on(TIM_TypeDef * timer);
  ```

- **Stop a timer:**
  ```c
  void TIM_off(TIM_TypeDef * timer);
  ```
stm32_unict_lib Functions for Timers

- **Read the counter:**
  ```c
  int16_t TIM_get(TIM_TypeDef * timer);
  ```

- **Write the counter:**
  ```c
  void TIM_set(TIM_TypeDef * timer, int16_t value);
  ```

- **Check if an update event occurred:**
  ```c
  int TIM_update_check(TIM_TypeDef * timer);
  ```

- **Clears the update event notification:**
  ```c
  void TIM_update_clear(TIM_TypeDef * timer);
  ```
Example: let’s flash a LED at 500 ms

- Default clock source **CK_PSC** is at 84 MHz (about 19 ns)
- We must derive a period of 500 ms
- We could use a **division factor** of 84000 in order to have a clock count signal (**CK_CNT**) at 1 ms, but the PSC register has only 16 bits...
- Let’s used instead a **division factor** of 8400 in order to have a clock count signal (**CK_CNT**) at 0.1 ms
- So we must have 5000 counts in order to have a period of 500 ms
Example: let’s flash a LED at 500 ms

Let’s configure the timebase with \textbf{prescaler}=8400 and \textbf{autoreload}=5000

Then poll the \textbf{“update event”}

When it occurs, toggle the led and \textbf{clear the event}
#include "stm32_unict_lib.h"

int main()
{
    // LED at PC3
    GPIO_init(GPIOC);
    GPIO_config_output(GPIOC, 3);

    // init the timer
    TIM_init(TIM2);

    // Configure the timebase
    // Counter clock set to 0.1 ms
    TIM_config_timebase(TIM2, 8400, 5000);

    TIM_set(TIM2, 0); // resets the counter
    TIM_on(TIM2); // starts the timer

    // infinite loop
    for (;;) {
        // check the update event
        if (TIM_update_check(TIM2)) {
            GPIO_toggle(GPIOC, 3);
            // clear the update event
            TIM_update_clear(TIM2);
        }
    }
}
#include "stm32_unict_lib.h"

int main()
{
    int last_key_state, flashing = 0;
    // LED at PC3
    GPIO_init(GPIOC);
    GPIO_config_output(GPIOC, 3);

    // pushbutton X (PB10)
    GPIO_init(GPIOB);
    GPIO_config_input(GPIOB, 10);

    TIM_init(TIM2); // init the timer

    // Configure the timebase, counter clock set to 0.1 ms
    TIM_config_timebase(TIM2, 8400, 5000);
    TIM_set(TIM2, 0); // resets the counter
    TIM_on(TIM2); // starts the timer

    last_key_state = GPIO_read(GPIOB, 10);
    for (;;) {
        int current_key_state = GPIO_read(GPIOB, 10);
        if (last_key_state == 1 && current_key_state == 0) flashing = !flashing;
        last_key_state = current_key_state;
        if (TIM_update_check(TIM2)) {
            if (flashing) GPIO_toggle(GPIOC, 3);
            else GPIO_write(GPIOC, 3, 0);
            TIM_update_clear(TIM2);
        }
    }
}
In a timer, many **events** (apart of the update event) occur.

Any event can be used generate an **IRQ** and thus trigger a proper **interrupt service routine**.

These functionalities are activated by setting proper bits in a timer SFR.
To enable timer IRQ, the following function can be used:

```c
void TIM_enable_irq(TIM_TypeDef * timer,
                    int irq_type);
```

where `irq_type` is set to the constant `IRQ_UPDATE`

Once the event is triggered, a specific interrupt service routine (ISR) is called, with name `TIMx_IRQHandler`

The ISR must handle the event and then notify handing via `TIM_update_clear()`
#include "stm32_unict_lib.h"

int flashing = 0;

int main()
{
    int last_key_state;
    // LED at PC3
    GPIO_init(GPIOC);
    GPIO_config_output(GPIOC, 3);
    // pushbutton X (PB10)
    GPIO_init(GPIOB);
    GPIO_config_input(GPIOB, 10);

    // init the timer
    TIM_init(TIM2);

    // Configure the timebase
    // Counter clock set to 0.1 ms
    TIM_config_timebase(TIM2, 8400, 2500);

    TIM_enable_irq(TIM2, IRQ_UPDATE);
    TIM_set(TIM2, 0); // resets the counter
    TIM_on(TIM2); // starts the timer

    last_key_state = GPIO_read(GPIOB, 10);
    for (; ;) {
        int current_key_state = GPIO_read(GPIOB, 10);
        if (last_key_state == 1 && current_key_state == 0) flashing = !flashing;
        last_key_state = current_key_state;
    }
}
... // Configure the timebase
// Counter clock set to 0.1 ms
TIM_config_timebase(TIM2, 8400, 2500);

TIM_enable_irq(TIM2, IRQ.UPDATE);
TIM_set(TIM2, 0); // resets the counter
TIM_on(TIM2); // starts the timer

last_key_state = GPIO_read(GPIOB, 10);
for (;;) {
    int current_key_state = GPIO_read(GPIOB, 10);
    if (last_key_state == 1 && current_key_state == 0) flashing = !flashing;
    last_key_state = current_key_state;
}

void TIM2_IRQHandler(void)
{
    if (flashing) GPIO_toggle(GPIOC, 3);
    else GPIO_write(GPIOC, 3, 0);
    TIM_update_clear(TIM2);
}
The Timers of the STM32 Microcontrollers

Corrado Santoro

ARSLAB - Autonomous and Robotic Systems Laboratory
Dipartimento di Matematica e Informatica - Università di Catania, Italy
santoro@dmi.unict.it

L.S.M. Course