Bern University
of Applied Sciences

Exercise – 06

Basic Timers with Interrupts

Andreas Habegger | Adrian Steiner
BTS3230 | Version 1.0.0 of 17.03.2025

In this exercise you will learn the concepts of the **"Basic Timer"**. Combining the Basic Timer with Interrupt Events allows you to implement, for example, Pulse Frequency Modulation (PFM) or Pulse Width Modulation (PWM). This exercise examines the timers on a simple fictitious headlight application. The application should implement several blinking modes of an LED controlled by a push button.

Objectives

- Basic Timer "TIM"
- Nested Vectored Interrupt Controller "NVIC"
- External Interrupt/Event Controller "EXTI"
- Control logic to implement headlight application

Outcomes

- Configure basic timers
- Timer based interrupt generation
- Measure microcontroller clock output (MCO)
- Generate pulse frequency modulation (PFM)
- Combine several interrupt service routines

Description

The “Basic Timer” integrated as TIM6 & TIM7 on the STM32F446xx MCUs, provides the following features:

- 16-bit auto-reload up counter
- 16-bit prescaler
- Interrupt generation on update event

A basic timer, sometimes called a counter, is a circuit that counts the pulses of a clock signal. The count value is compared to a configurable reference value. If both are equal, the counter is reset. When the counter is reset, an event is triggered which can be used as a trigger. In the form used here, for example, as an IRQ calling an ISR. This can be used to process a function in fixed cycles. In this exercise, this function is used to make a LED controller flash with different frequencies depending on the SW center button.

> **Note**
>
> Only the **NULCEO-64 board** and **mbed Application Shield** are used in this exercise, but the temperature logger board has no effect on this exercise and can remain connected.

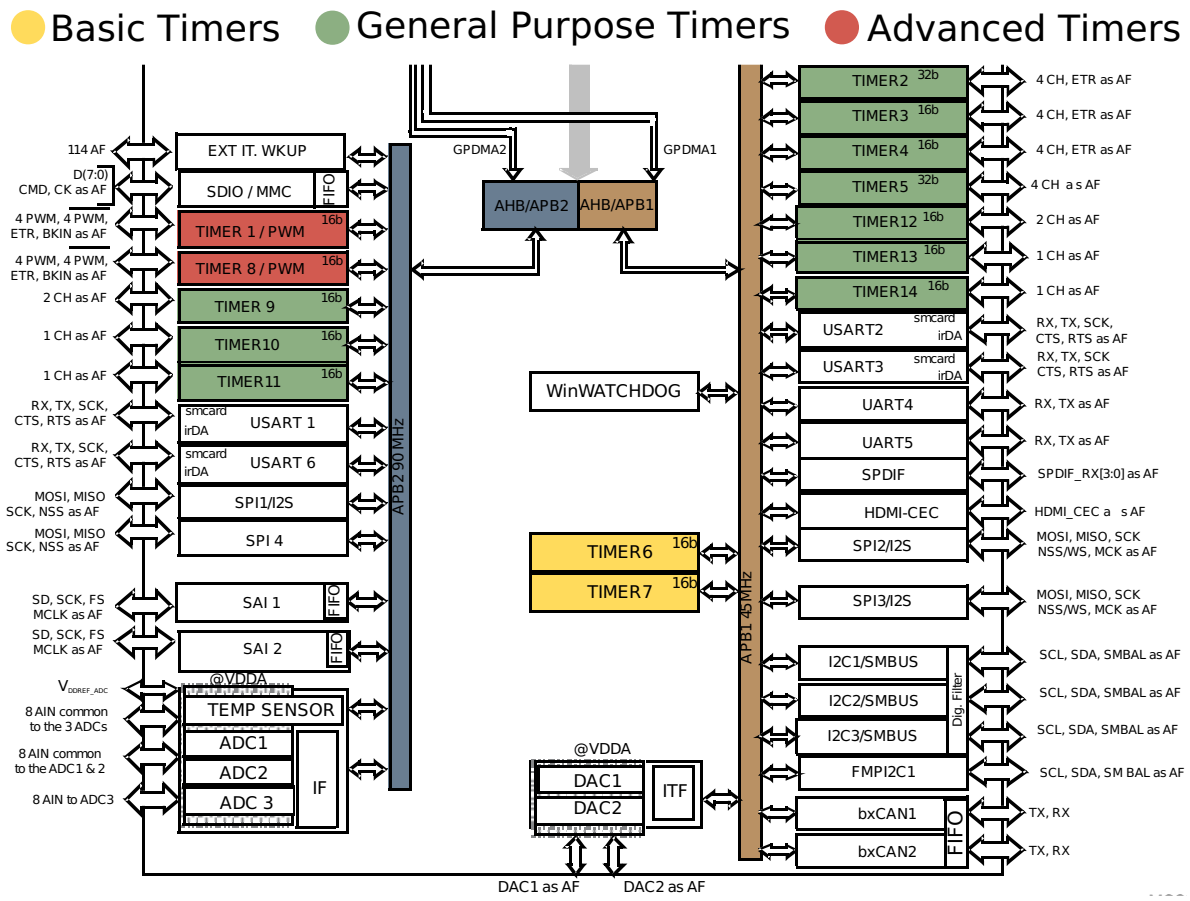
Tasks

The final goal is a headlight application. This is controlled by the SW center button. When the system is started, the lamp – in this case an LED – is switched off. When the button is pressed, the LED is switched on. The LED should flash at $1\,\mathrm{Hz}$. Pressing the button again should double the flashing frequency. Pressing the button a third time will change the frequency again to $4\,\mathrm{Hz}$. A fourth press will reset the flashing frequency to the initial flashing frequency of $1\,\mathrm{Hz}$. This cycle is repeated each time the button is pressed. To switch off the LED, press and hold the button for longer than $3\,\mathrm{s}$. Follow the description below, which will take you through the exercise in an iterative way. However, if you feel confident enough, you can go straight to the final implementation.

Measure current time

In order to set an accurate time with a timer, you need to know the current core clock of the system and which Advanced Peripheral Bus (APB) the peripheral is connected to, as our MCU has two of them. This information can be found in the STM32F446xC/E block diagram in the datasheet STM32F446re.

STM32F446cC/E block diagram excerpt



A snippet of the block diagram of the STM32F446cC/E from the Datasheet STM32-F446re: page 16

Question

a. On which APB are the basic timer located?

b. What is the max frequency of this bus?

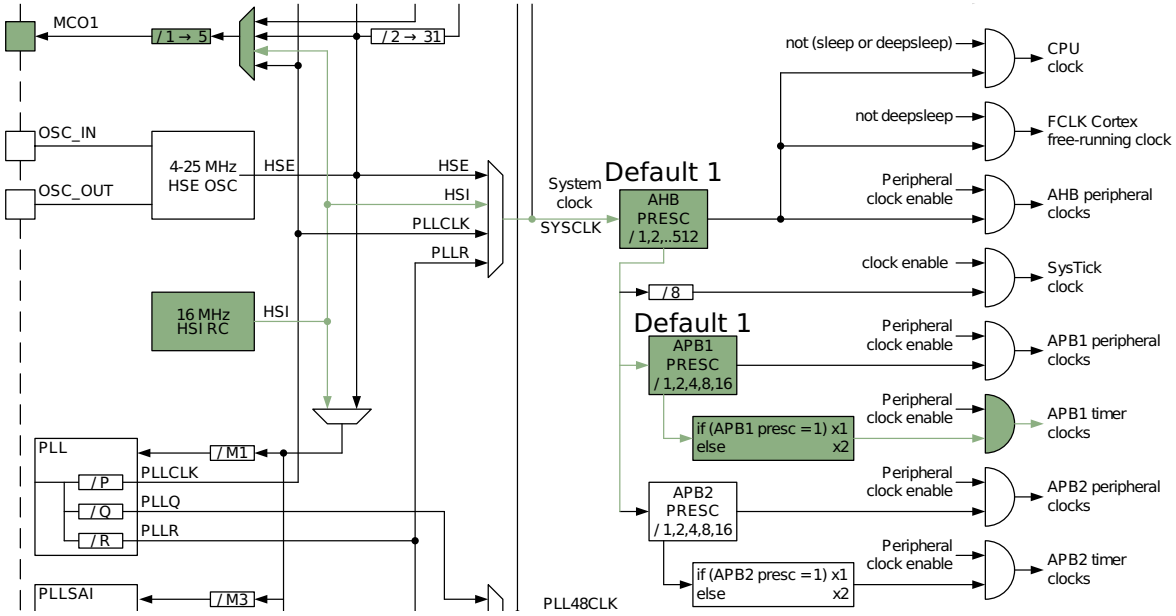
Solution

a. The basic timer TIM6 and TIM7 are connected to the APB1 found in extraction from block diagram

b. The max frequency of this bus is $45\,\mathrm{MHz}$. This solution is found in the same document.

With this information we have a look at the clock tree of reference manual RM0390. Our stm32f446x_template uses the default clock configurations, which results in all prescaler values being set to 1 and the HSI (High Speed Internal) oscillator being used.

Extraction from clock tree



Clock tree extraction from the Reference Manual, page 118

To answer the first question, we can read the clock frequency from the clock tree or measure the current output of the HSI using the microcontroller clock outputs (MCO). As you can see in the clock tree, only MCO1 is connected to the HSI. To find out which pin and alternate function the MCO1 is wired to, we have a look at the alternate function table in the Datasheet STM32-F446re: page 57. In this figure we can see that the MCO1 is only connected to the GPIO PA8 with the alternative function AF0. To know how to activate an alternative function on a GPIO, have a look at reference manual RM0390.

Config PA8 as alternate function AF0

```
/* ADD MCO1 at PA8 as AF0*/
MODIFY_REG(GPIOA->MODER, GPIO_MODER_MODER8_Msk, 0b10 << GPIO_MODER_MODER8_Pos);
MODIFY_REG(GPIOA->AFR[1], GPIO_AFRH_AFSEL8_Msk, 0b00 << GPIO_AFRH_AFSEL8_Pos);
```

The MCO configuration is included in the RCC peripheral. The description how to configure this functionality is described in the Reference Manual: page 131. The clock source and a prescaler value for the output can be configured.

Important

For the MCO1, use a prescaler of a size that allows the signal to be measured with your standard equipment.

Enable MCO1

```
/* Set MCO1 to check HSI clock source, set prescaler to 4*/
MODIFY_REG(RCC->CFGR, RCC_CFGR_MCO1_Msk, 0b00 << RCC_CFGR_MCO1_Pos);
MODIFY_REG(RCC->CFGR, RCC_CFGR_MCO1PRE_Msk, 0b110 << RCC_CFGR_MCO1PRE_Pos);
```

Once configured, you can measure the HSI clock speed divided by the prescaler with your saleae logic analyzer.

Hint

To get the clock signal at the desired GPIO output pin you need to enable the bus of the corresponding GPIO port.

Question

c. What is the frequency of the HSI and system clock?

d. What is the clock speed on the APB1 timer clock?

Solution

c. The frequency of the HSI and the system clock is $16\,\mathrm{MHz}$

d. As all prescalers (AHB PRESC and APB1 PRESC) are 1, the clock doubler for timer clocks is not enabled and will result in a factor of 1. With this configuration the APB1 bus speed has is the same clock speed as the system clock.

Basic TIM example

The following is an example of how to configure a basic timer with an interrupt at update time.

- At the beginning we configure the basic timer used

Configuration Basic timer

```
SET_BIT(TIM6->CR1, TIM_CR1_URS);            // only CNT overflow/underflow
generates update interrupt
SET_BIT(TIM6->CR1, TIM_CR1_ARPE);           // ARR is buffered
SET_BIT(TIM6->DIER, TIM_DIER_UIE);          // Update interrupt enable
CLEAR_BIT(TIM6->SR, TIM_SR_UIF);            // Clear pending interrupt
WRITE_REG(TIM6->PSC, custom_DIVIDER_VALUE); // Prescaler value
WRITE_REG(TIM6->ARR, custom_BLINK_PERIOD);  // Auto Reload Register
```

To complete the configuration, the prescaler and auto-reload registers must be calculated. These values and the APBx frequency give the timer frequency or period duration. The formula for calculating the solution is

$$f_{TIM} = \frac{f_{APBx}}{(PSC + 1) * (ARR + 1)}$$

- f_{TIM} = Frequency of timer update event
- f_{APBx} = Frequency of APB bus
- PSC = prescaler register value
- ARR = auto-reload register value

Hint

Why adding 1 to PSC and ARR?

The PSC is defined as adding one because the calculation starts with one. For the ARR you also add one, because the roll over from the ARR value to 0 is also a count.

- Next, enable the interrupt on the NVIC for your timer

NVIC enable interrupt

```
NVIC_SetPriority(TIM6_DAC_IRQn, 1);
NVIC_ClearPendingIRQ(TIM6_DAC_IRQn);
NVIC_EnableIRQ(TIM6_DAC_IRQn);
```

- Activate the timer

Enable timer

```
/* Start timer */
SET_BIT(TIM6->CR1, TIM_CR1_CEN);
```

- Callback function from ISR vector table (see in the stm32_startup.c)

IRQ callback handler

```
/* TIM6 callback function*/
void TIM6_DAC_IRQHandler(void){
    /*Clear update interrupt flag*/
    if (READ_BIT(TIM6->SR, TIM_SR_UIF) != 0)    {
        CLEAR_BIT(TIM6->SR, TIM_SR_UIF);
        /* Implementation of ISR */
    }
}
```

Implementation

Flashing LED

As a first iteration we suggest the implementation of a blinking LED. Use one of the two basic timers. See the Tasks section for more information on how to configure a timer. Follow the procedure below to implement a timed blinking LED.

Exercise

1. Configure the LED GPIO as an output.
2. Configure one of the base timers to flash at $1\,\mathrm{Hz}$. Set the prescaler value according to the system clock.

   Hint

   Do not forget to enable the clocks on **ALL** used peripherals.

3. Configure the timer to fire interrupts when it reloads the value of the auto-reload register.
4. Configure the interrupt controller (NVIC) to call the ISR on timer IRQs.
5. Implement the ISR to clear the IRQ and toggle the LED.

Control of the flashing frequency

Based on the previous implementation, add control logic to change the blink frequency. The pre-defined blink frequencies from the task description are used. Events generated by the button will trigger a change in the blink frequency. Follow the procedure below to successfully implement this task.

Exercise

6. Configure the EXTI controller to trigger interrupts when we press the SW center button.
7. Modify the NVIC configuration to also trigger on EXTI events of SW center button.
8. Define the ARR values of the timer, starting at $1\,\mathrm{Hz}$, doubling it until ending at $4\,\mathrm{Hz}$.
9. Implement an ISR to check if the SW center button IRQ has been fired, clear the interrupt request and update the auto-reload register accordingly.

On/Off control

A second base timer is required to implement the on/off control. This timer is used to detect whether the button press was short or long. A short key press will change the flashing frequency, whereas a long key press will switch off the flashing LED. The threshold value that distinguishes between a short and a long keystroke is $3\,\mathrm{s}$.

Exercise

10. Configure the second base timer to fire interrupt requests when the threshold time has elapsed.
11. Use the rising edge of the key to start the timer.
12. When the timer fires an interrupt, the LED is switched off and the base timer used for flashing is switched off.
13. Use the falling edge to stop the timer and switch the LED on. If the LED is already on, change the flashing frequency.

(optional) Using only 1 timer

This exercise is for motivated students and is therefore optional! For simplicity, the suggestions made in the exercise force an implementation with two basic timers. Since timer resources are limited and battery-powered systems require energy-efficient implementations, we may need to rethink the current implementation.

Question

- How can we reduce the dynamic power consumption?
- Is it possible to use only one base timer instead of two?

14. Redesign the implementation to use only one base timer instead of two.