Type A Board Dev Guide II

Wilson 2020.7.26

https://github.com/TAMU-Robomasters/Tutorial

Roadmap:

1. STM32CubeMX, Keil uvision
2. LED, GPIO
3. Timer
4. PWM, passive buzzer, servo
5. Buttons
6. USB
7. Flash
8. I2C, IST8310 (magnetic sensor)
9. OLED
10. BMI088 (gyroscope)
11. Motor control with CAN
12. freeRTOS
13. IMU
14. Chassis tasks
15. Gimbal control
16. BIG PICTURE

Workflow:

Recall last time, the workflow is:

1. Use CubeMX to enable/configure pinout.
2. Ues CubeMX to generate template code.
3. Open code files in Keil uvision. Edit code.
4. Generate binary and upload to board.

CubeMX configurations include:

1. Clock
2. Pin enabling
3. Pin renaming

My suggestion is to create a CubeMX project file with clock and debug wire configured and nothing else added. Use the file as a starting point for experimenting different features. Save the hassle to repeatedly configure clock!

Timer:

In tutorial I, we used `HAL_Delay(1000)` to act as a timer.

Usually this is not recommended. Use a hardware timer instead.

Timers will cause hardware interrupt. Before calling the interrupt, the program carries on without interruption, i.e. no waiting time wasted.

Timer concepts:

1. Clock division
2. Counter
3. Reloading

Think of those concepts as the hour hand, minute hand and second hand on a clock. They move at different speeds/frequencies within the same time interval. And their readings reset after different periods of time.

Timer in Implementation:

Registers that are used for timers:

1. `TIMx_PSC`: clock pre-scaler
2. `TIMx_CNT`: counter
3. `TIMx_ARR`: store the finishing value for counter

How it works:

1. Divide clock by value stored in `TIMx_PSC`
2. Incremented value stored in `TIMx_CNT` from 0
3. Increment counter value every clock cycle
4. Until counter value reaches value stored in `TIMx_ARR`
5. Trigger interrupt signal

Note:

After the clock division circuit, different timers may run at different clock frequencies. Hence another factor to affect interrupt-calling frequency is the timer clock frequency.

Interrupt:

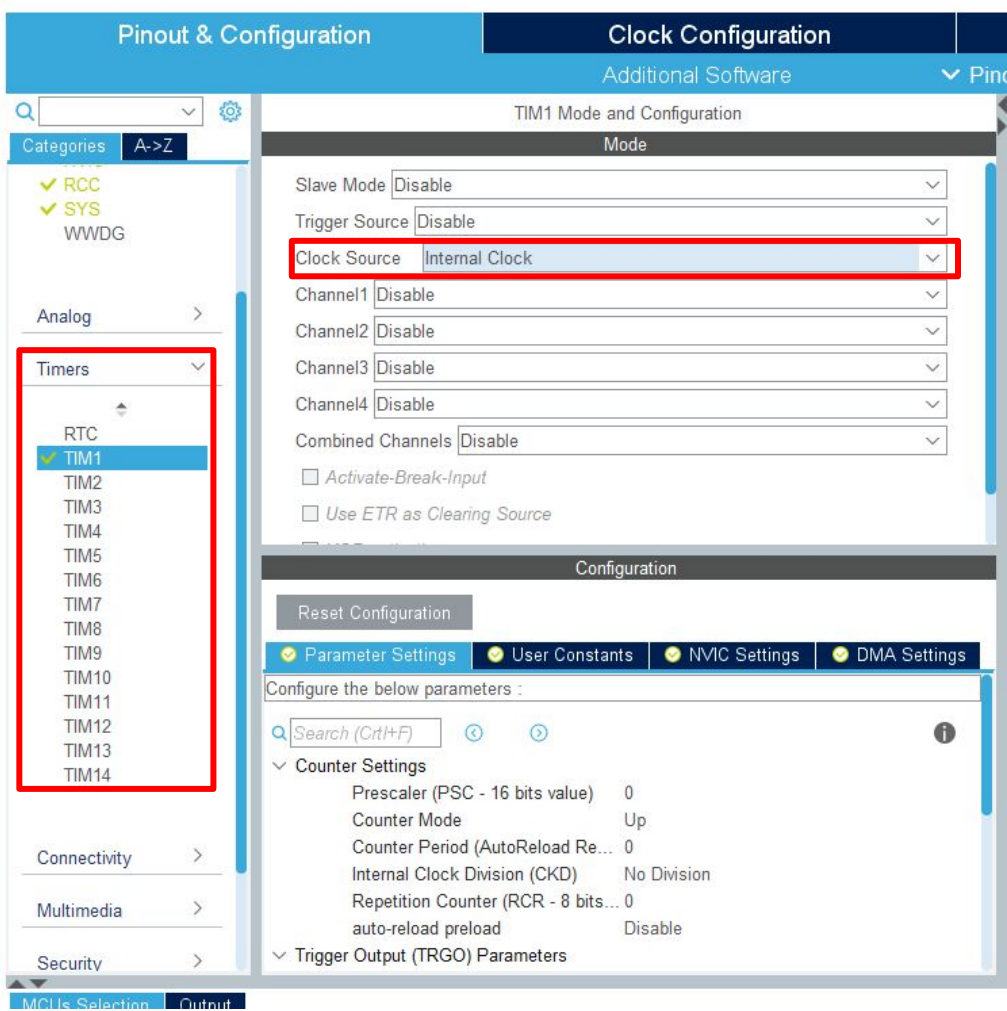
From Wiki:

In digital computers, an **interrupt** is a response by the processor to an event that needs attention from the software. An interrupt condition alerts the processor and serves as a request for the processor to interrupt the currently executing code when permitted, so that the event can be processed in a timely manner.

Summary:

Stop what you are doing if signaled, otherwise leave it running in the background.

CubeMX TIM1 Enabling:



CubeMX TIM1 Configuration:

Generate the code. In `main.c` go to the main function, there should be a line:

`MX_TIM1_Init()`

This is the line that enables TIM1 (timer 1).

Build the Keil project, right click on this line, go to its definition.

In `tim.c`, in function `HAL_TIM_Base_MspInit`, go to the definition of `__HAL_RCC_TIM1_CLK_ENABLE`

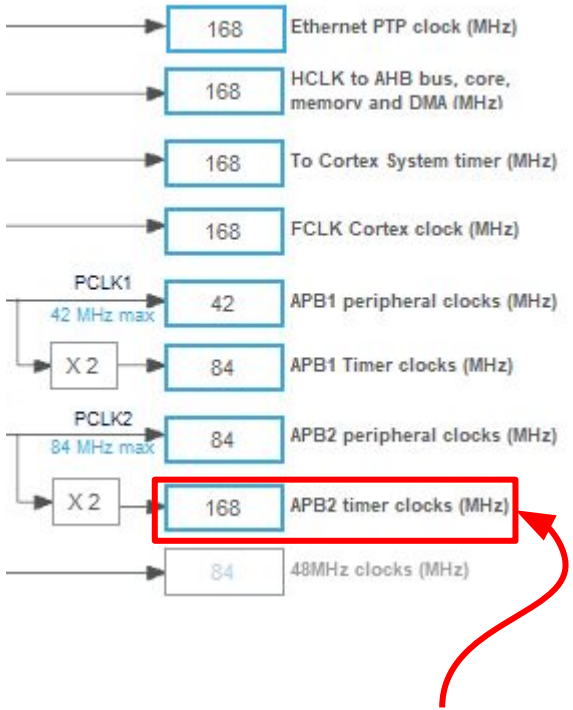
We get:

So TIM1 is on APB2 bus.

```
#define __HAL_RCC_TIM1_CLK_ENABLE()  do { \
  __IO uint32_t tmpreg = 0x00U; \
  SET_BIT(RCC->APB2ENR, RCC_APB2ENR_TIM1EN);\
  /* Delay after an RCC peripheral clock enabling */ \
  tmpreg = READ_BIT(RCC->APB2ENR, RCC_APB2ENR_TIM1EN);\
  UNUSED(tmpreg); \
  } while(0U)
```

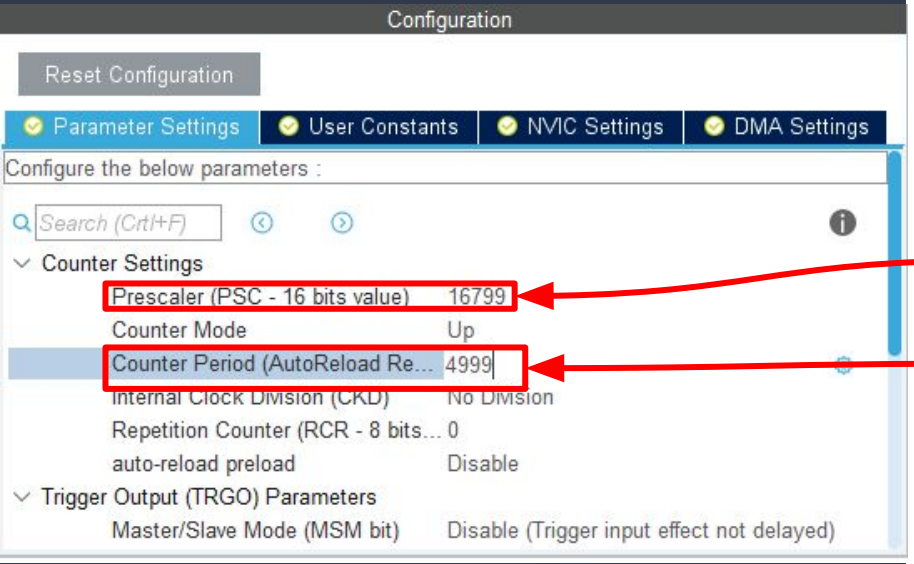
Calculating timer frequency 1/2:

Recall in Clock configuration in CubeMX:



APB2 frequency: 168 MHz

Calculating timer frequency 2/2:



If we want to trigger signal at 2Hz:

Trigger frequency = Bus frequency/((Pre-scale+1) * (Finishing value+1))

2 = 168000000/((Pre-scale+1) * (Finishing value+1))

Conclusion: any pair of pre-scale and finishing value that their product is 84000000

i.e. pre-scale = 16799, finishing value = 4999

Note:

1. `TIMx_PSC`: pre-scale
2. `TIMx_CNT`: counter
3. `TIMx_ARR`: finishing value

NVIC
Nested Vectored Interrupt Controller:

Device that handles interrupts.

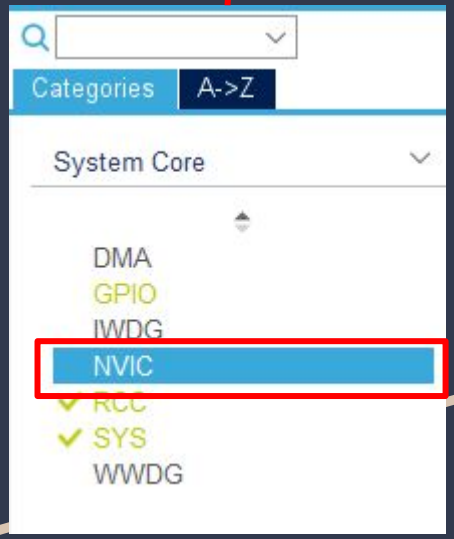
Different interrupts have different priority levels.

What is during the handling of an interrupt, what will happen if another interrupt happened?

Solution: assign each interrupt with `preemption priority` and `sub-priority`

First, handle interrupts according to `preemption priority,` if multiple interrupts with the same `preemption priority` took place, handle them according to their `sub-priority`

NVIC in CubeMX:



Configuration

NVIC | Code generation

Priority Group: 4 bits for pre-emption priority 0 bits for subpriority

Sort by Premption Priority and Sub Priority

Search: Search (Crt...)

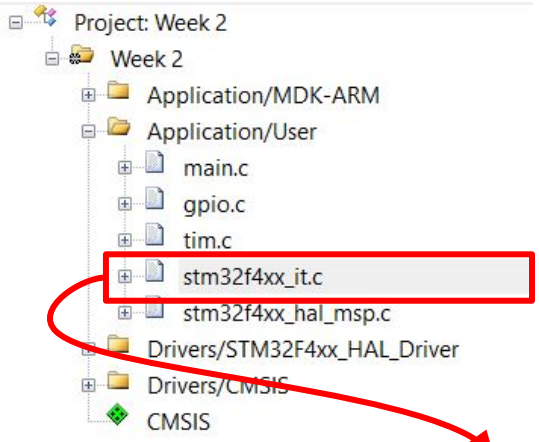
Show only enabled interrupts

Force DMA channels Interrupts

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
Non maskable interrupt	☑	0	0
Hard fault interrupt	☑	0	0
Memory management fault	☑	0	0
Pre-fetch fault, memory access fault	☑	0	0
Undefined instruction or illegal state	☑	0	0
System service call via SWI instruction	☑	0	0
Debug monitor	☑	0	0
Pendable request for system service	☑	0	0
Time base: System tick timer	☑	0	0
PVD interrupt through EXTI line 16	☐	0	0
Flash global interrupt	☐	0	0
RCC global interrupt	☐	0	0
TIM1 break interrupt and TIM9 global interrupt	☐	0	0
TIM1 update interrupt and TIM10 global interrupt	☑	0	0
TIM1 trigger and commutation interrupts and TIM11 global interrupt	☐	0	0
TIM1 capture compare interrupt	☐	0	0
FPU global interrupt	☐	0	0

Use timer in code 1/3:

Generate code using CubeMX. Open Keil.

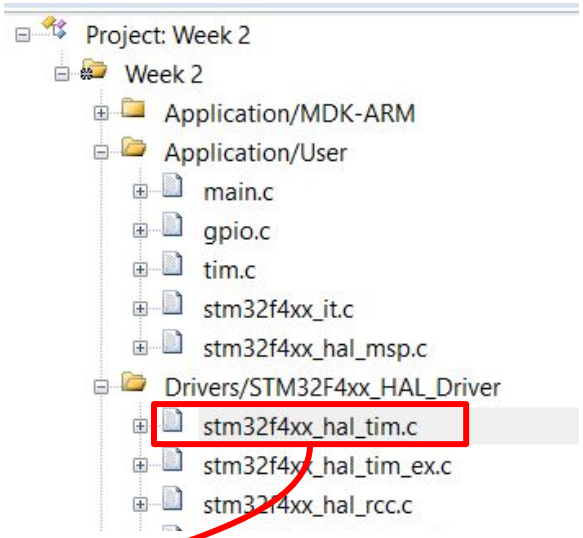



```
void TIM1_UP_TIM10_IRQHandler(void)
{
  /* USER CODE BEGIN TIM1_UP_TIM10_IRQn 0 */

  /* USER CODE END TIM1_UP_TIM10_IRQn 0 */
  HAL_TIM_IRQHandler(&htim1);
  /* USER CODE BEGIN TIM1_UP_TIM10_IRQn 1 */

  /* USER CODE END TIM1_UP_TIM10_IRQn 1 */
}
```

Use timer in code 2/3:



Overwrite the function in main.c

```
__weak void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  /* Prevent unused argument(s) compilation warning */
  UNUSED(htim);

  /* NOTE : This function should not be modified, when the callback is needed,
            the HAL_TIM_PeriodElapsedCallback could be implemented in the user file
   */
}
```

Use timer in code 3/3:

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    if(htim == &htim1)
    {
      //500ms trigger
      bsp_led_toggle();
    }
}
```

```
MX_TIM1_Init(); // auto generated

HAL_TIM_Base_Start_IT(&htim1); // add this!

// the above 2 function calls shall be placed
// in main() in main.c
```

References:

https://en.wikipedia.org/wiki/Interrupt