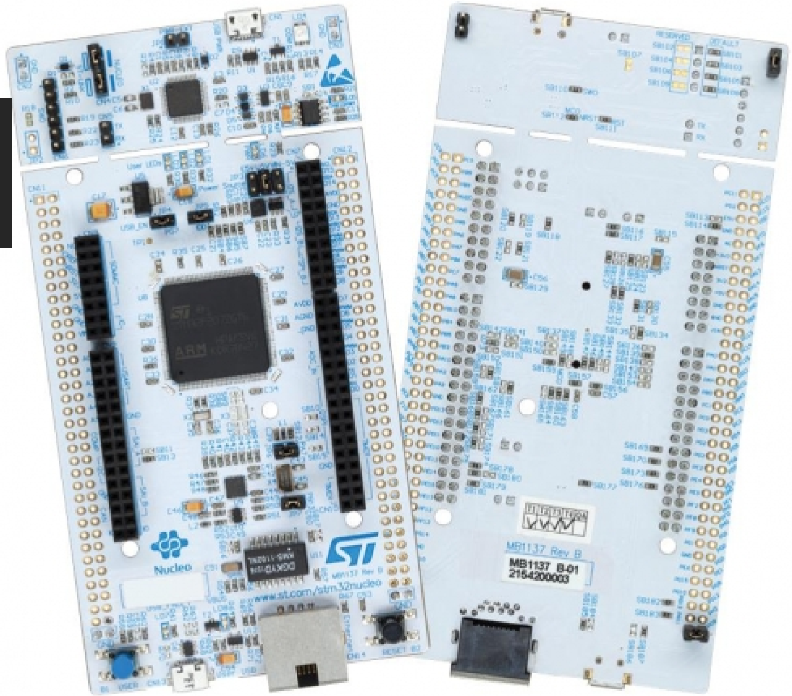
NEEEICUM
núcleo de estudantes de engenharia eletrónica industrial e computadores da universidade do minho

ESRG
EMBEDDED SYSTEMS RESEARCH GROUP

STM 32 F767ZI



CLOCK, TIMER AND ANALOG INTERFACE

ÁLVARO CASTRO LEITE
NOV 2019

Requirements

Micro-USB cable

STM32 development board

PC with all the tools installed

PC with terminal

Know everything from last workshop session

NEEEICUM
núcleo de estudantes de engenharia eletrónica industrial e computadores da universidade do minho

Agenda

Skill level: Beginner

Clocks

Clock Tree

Timers

Analog-to-Digital Converter

Digital-to-Analog Converter

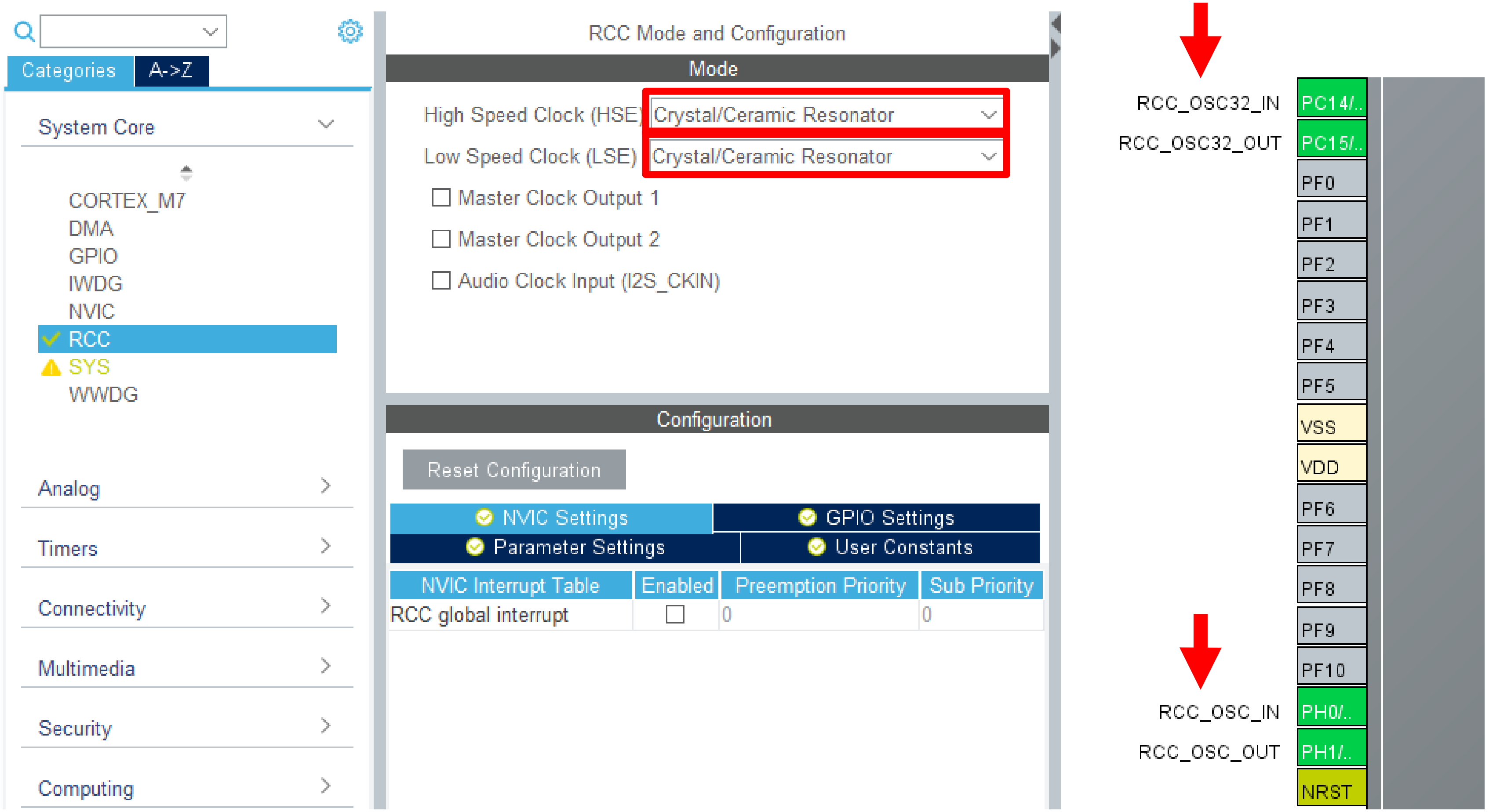
Clock

A clock is a device that generates periodic signals and it is the most widespread form of heart beat source in digital electronics. Almost every digital circuit needs a way to synchronize its internal circuitry or to synchronize itself with other circuits.

All STM32 MCUs can be clocked by different and distinct clock sources alternatively. They are: **Crystal oscillator** and **RC oscillator**.

The STM32 has internal clocks but it also allows external clocks.

Clock


Categories
A->Z
System Core
CORTEX_M7
DMA
GPIO
IWDG
NVIC
RCC
SYS
WWDG
Analog
Timers
Connectivity
Multimedia
Security
Computing
RCC Mode and Configuration
Mode
High Speed Clock (HSE) Crystal/Ceramic Resonator
Low Speed Clock (LSE) Crystal/Ceramic Resonator
Master Clock Output 1
Master Clock Output 2
Audio Clock Input (I2S_CKIN)
Configuration
Reset Configuration
NVIC Settings
GPIO Settings
Parameter Settings
User Constants
NVIC Interrupt Table
Enabled
Preemption Priority
Sub Priority
RCC global interrupt
0
0
RCC_OSC32_IN
PC14/..
RCC_OSC32_OUT
PC15/..
PF0
PF1
PF2
PF3
PF4
PF5
VSS
VDD
PF6
PF7
PF8
PF9
PF10
RCC_OSC_IN
PH0/..
RCC_OSC_OUT
PH1/..
NRST

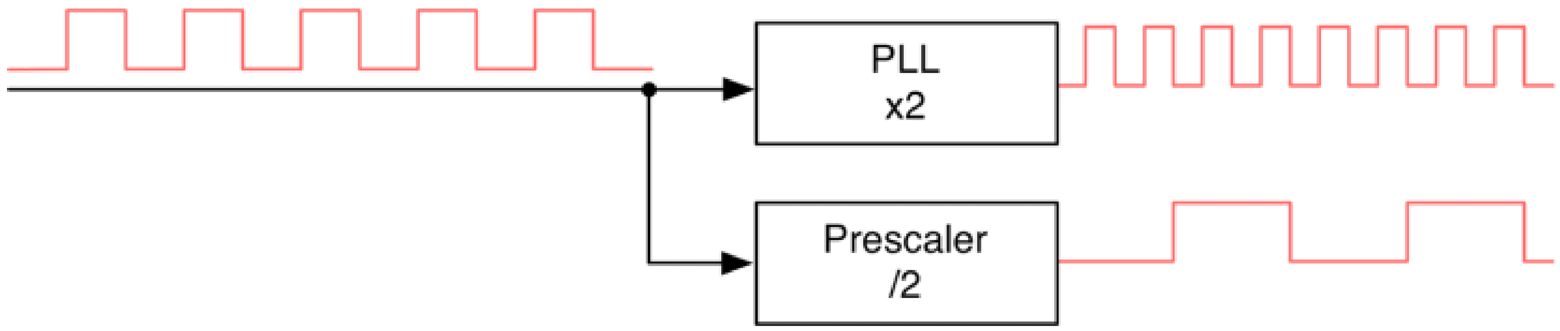
Clock Tree

The same clock signal, however, cannot be used to feed all components and peripherals provided by a modern microcontroller like STM32. A sophisticated distribution network, also called **clock tree**, is responsible for managing and feeding the signals inside an STM32 MCU.

Neither of the Cortex-M core nor of the other peripherals frequency is establish by the frequency of the high-speed oscillator.

Clock Tree

Using several programmable Phase-Locked Loops (PLL) and prescalers, it is possible to increase/decrease the source frequency at need, depending on the performance we want to reach, the maximum speed for a given peripheral or bus and the overall global power consumption.



Clock Tree

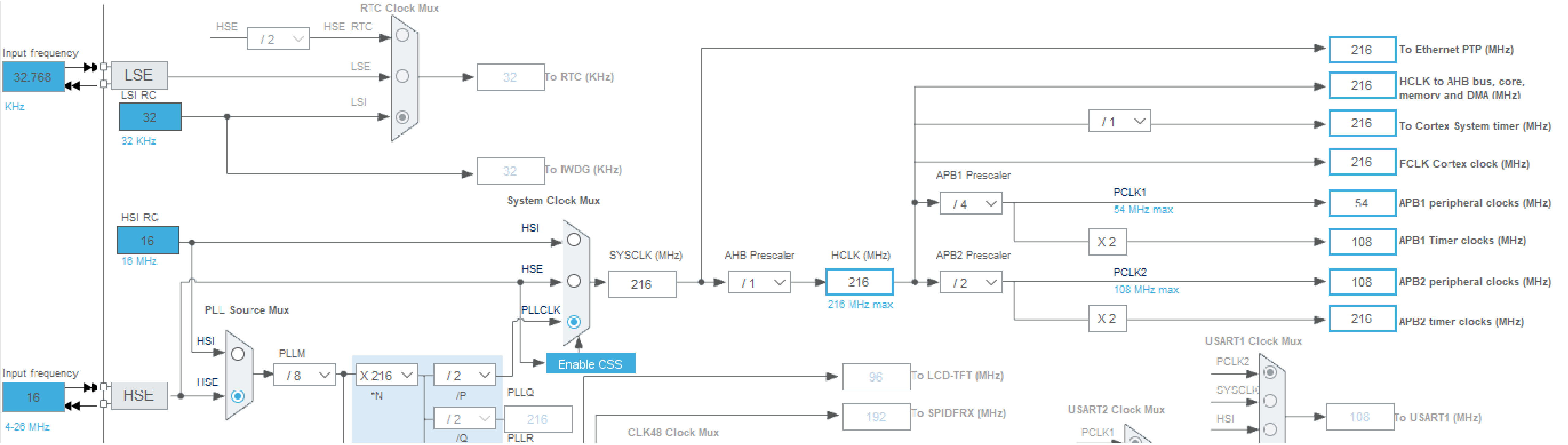
Question: What frequency the STM32 Core is working at?

Tip: Open “Clock Configuration”

Answer: 16MHz

Clock Tree

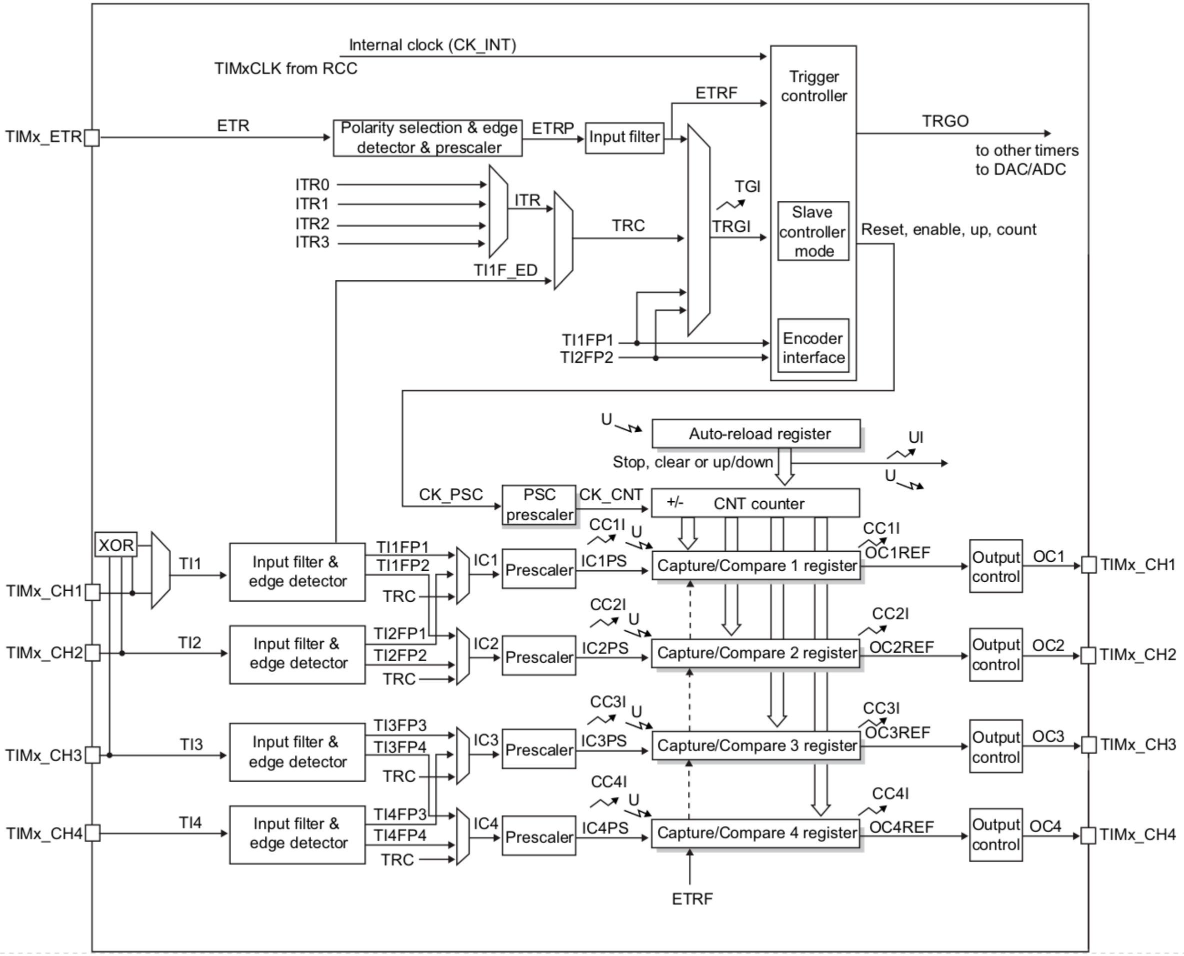
Example 1: Make the STM32 Core work at maximum clock speed (216MHz).



Timers

A timer is a free-running counter with a counting frequency that is a fraction of its source clock. The counting speed can be reduced using a dedicated prescaler for each timer. Depending on the timer type, it can be clocked by the internal clock (which is derived from the bus where it is connected), by an external clock source or by another timer used as “master”.

Timers



RM0410, page 962

Timers

STM32 timers can mainly be grouped into a few categories. Let us take a brief look at some of them:

- **Basic timers**: this are the simplest form of timers in STM32 MCUs. They are 16-bit timers used as time base generator, and they **don't** have output/input pins.
- **General purpose timers**: they are 16/32-bit timers providing all the classical features that a timer usually implements. They are used in any application for **Output Compare Mode** (timing), **One-Pulse Mode**, **Input Capture Mode** (for external signal), etc. Can also be be used like as a basic timer.

Timers

Timer type		STM32 F04x /F070x6 /F03x (excluding /F030x8 and /F030x)	STM32 F030xB /F030x8 /F05x /F09x /F07x (excluding F070x6)	STM32 F101 /F102 /F103 lines XL density (xF, xG)	STM32 F101 /F102 /F103 /F105 /F107 lines up to high density (x4-xE)	STM32 F100 value line	STM32 F2 /F4 (excluding /F401, /F411, /F410)	STM32 F401 /F411 /F410	STM32 F30X /F3x8 (excluding /F378)	STM32 F37x	STM32 F334	STM32 F31x	STM32 F7 Series	STM32 L05X /L06x /L07x /L08x lines	STM32 L03x /L02x /L01x lines	STM32 L1 Series	STM32 L4 Series
Advanced		TIM1	TIM1	TIM1[1] TIM8[1]	TIM1[1] TIM8[1]	TIM1	TIM1 TIM8	TIM1	TIM1 TIM8[1] TIM20[1]	-	TIM1	TIM1 TIM8[1]	TIM1 TIM8	-	-	-	TIM1 TIM8[1]
General purpose	32-bit	TIM2	TIM2	-	-	-	TIM2 TIM5	TIM2[1] TIM5	TIM2	TIM2 TIM5	TIM2	TIM2	TIM2 TIM5	-	-	TIM5[1]	TIM2 TIM5[1]
	16-bit	TIM3	TIM3	TIM2 TIM3 TIM4 TIM5	TIM2 TIM3 TIM4[1] TIM5[1]	TIM2 TIM3 TIM4 TIM5[1]	TIM3 TIM4	TIM3[1] TIM4[1]	TIM3[1] TIM4[1] TIM19[1]	TIM3 TIM4 TIM19	TIM3	TIM3 TIM4	TIM3 TIM4	TIM2 TIM3[1]	TIM2	TIM2 TIM3 TIM4	TIM3[1] TIM4[1]
Basic		-	TIM6 TIM7[1]	TIM6 TIM7	TIM6[1] TIM7[1]	TIM6 TIM7	TIM6 TIM7	TIM6[1]	TIM6 TIM7[1]	TIM6 TIM7 TIM18	TIM6 TIM7	TIM6 TIM7[1]	TIM6 TIM7	TIM6 TIM7[1]	-	TIM6 TIM7	TIM6 TIM7
1 channel		TIM14	TIM14	TIM10 TIM11 TIM13 TIM14	-	TIM13[1] TIM14[1]	TIM10 TIM11 TIM13 TIM14	TIM10[1] TIM11	-	TIM13 TIM14	-	-	TIM10 TIM11 TIM13 TIM14	-	-	TIM10 TIM11	-
2-channel		-	-	TIM9 TIM12	-	TIM12[1]	TIM9 TIM12	TIM9	-	TIM12	-	-	TIM9 TIM12	TIM21 TIM22	TIM21 TIM22[1]	TIM9	-
2-channel with complementary output		-	TIM15	-	-	TIM15	-	-	TIM15	TIM15	TIM15	TIM15	-	-	-	-	TIM15
1-channel with complementary output		TIM16 TIM17	TIM16 TIM17	-	-	TIM16 TIM17	-	-	TIM16 TIM17	TIM16 TIM17	TIM16 TIM17	TIM16 TIM17	-	-	-	-	TIM16 TIM17[1]
Low-power timer		-	-	-	-	-	-	LPTIM1[1]	-	-	-	-	LPTIM1	LPTIM1	LPTIM1	-	LPTIM1 LPTIM2

Timers

The Period and Prescaler registers determine the timer frequency, that is, how long it takes to overflow (or, if you prefer, how often an Update Event is generated), according to this simple formula:

$$UpdateEvent = \frac{Timer_{clock}}{(Prescaler + 1)(Period + 1)}$$

Timers

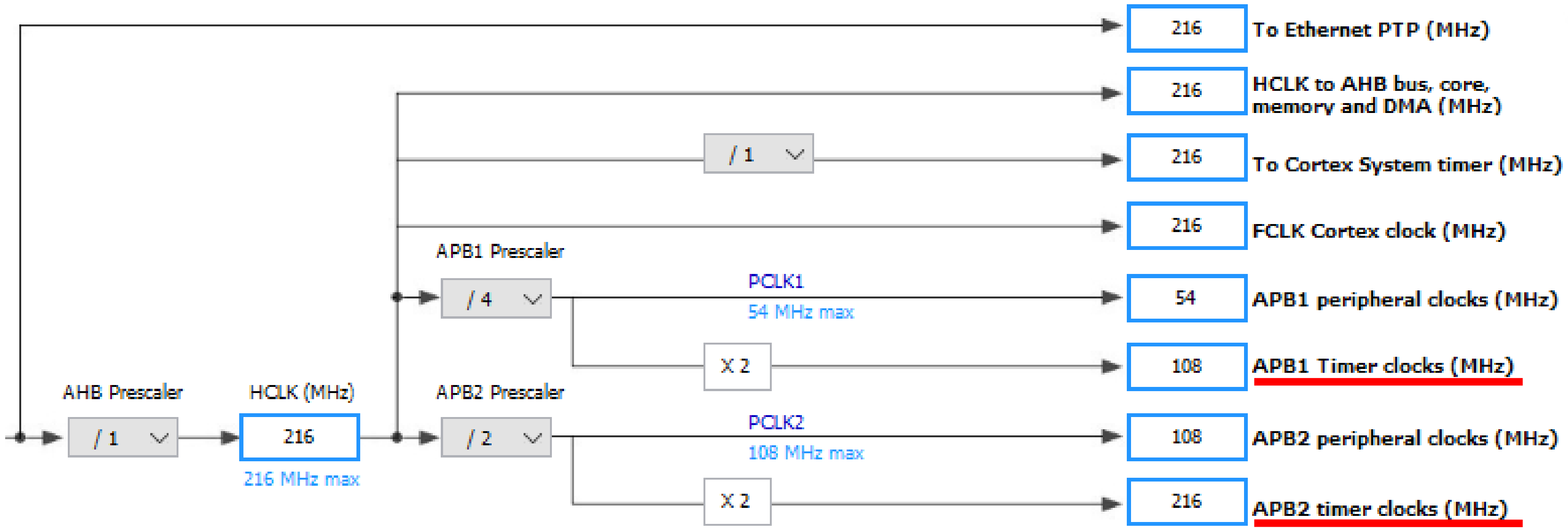
Question: What bus is connected to each timer and what’s the maximum clock frequency on each of them?

Tip: Take a look at DS11532, page 19

Answer:

Timer	1	2	3	4	5	6	7	8	9	10	11	12	13	14
APB1 (108 MHz)		x	x	x	x	x	x					x	x	x
APB2 (216 MHz)	x							x	x	x	x			

Timers



216
To Ethernet PTP (MHz)
216
HCLK to AHB bus, core, memory and DMA (MHz)
/ 1
216
To Cortex System timer (MHz)
216
FCLK Cortex clock (MHz)
APB1 Prescaler
/ 4
PCLK1
54 MHz max
54
APB1 peripheral clocks (MHz)
X 2
108
APB1 Timer clocks (MHz)
AHB Prescaler
/ 1
HCLK (MHz)
216
216 MHz max
APB2 Prescaler
/ 2
PCLK2
108 MHz max
108
APB2 peripheral clocks (MHz)
X 2
216
APB2 timer clocks (MHz)

Timers

Example 2: Make the Green Led blink exactly at a frequency of 10Hz.

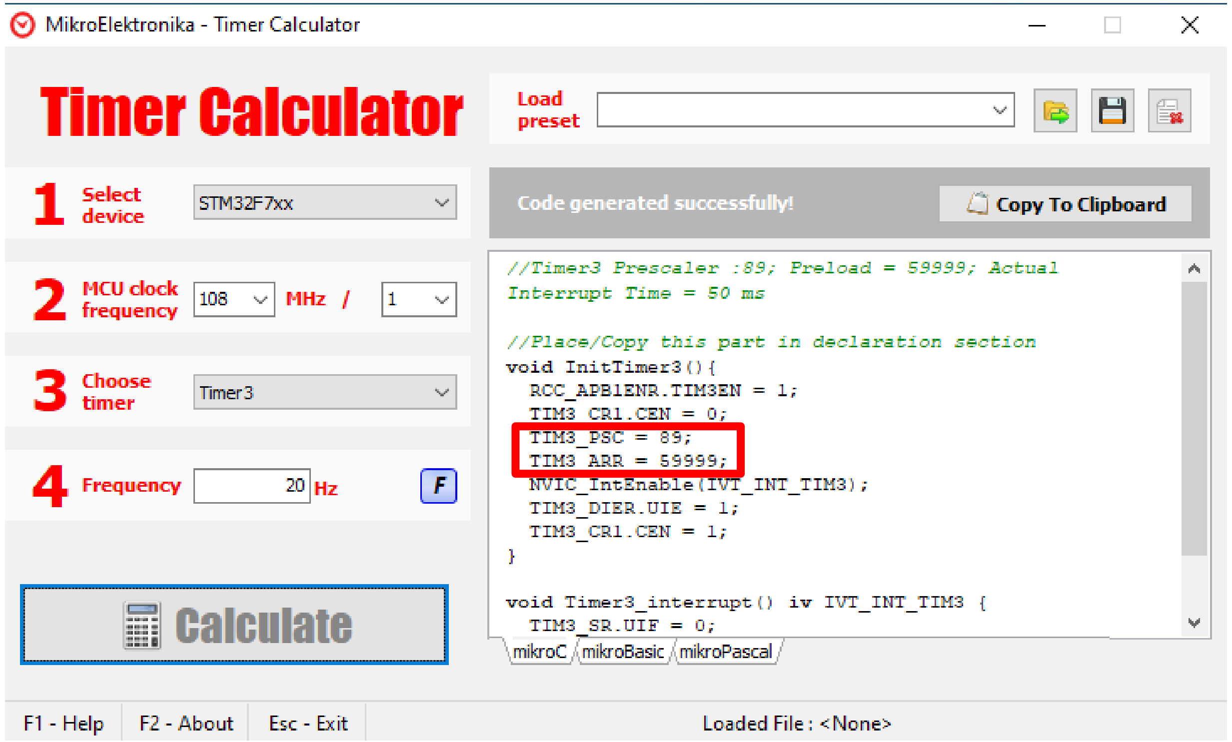
Question: Which timer can I connect to PB0 (Green Led pin)?

Tip: DS11532, page 89.

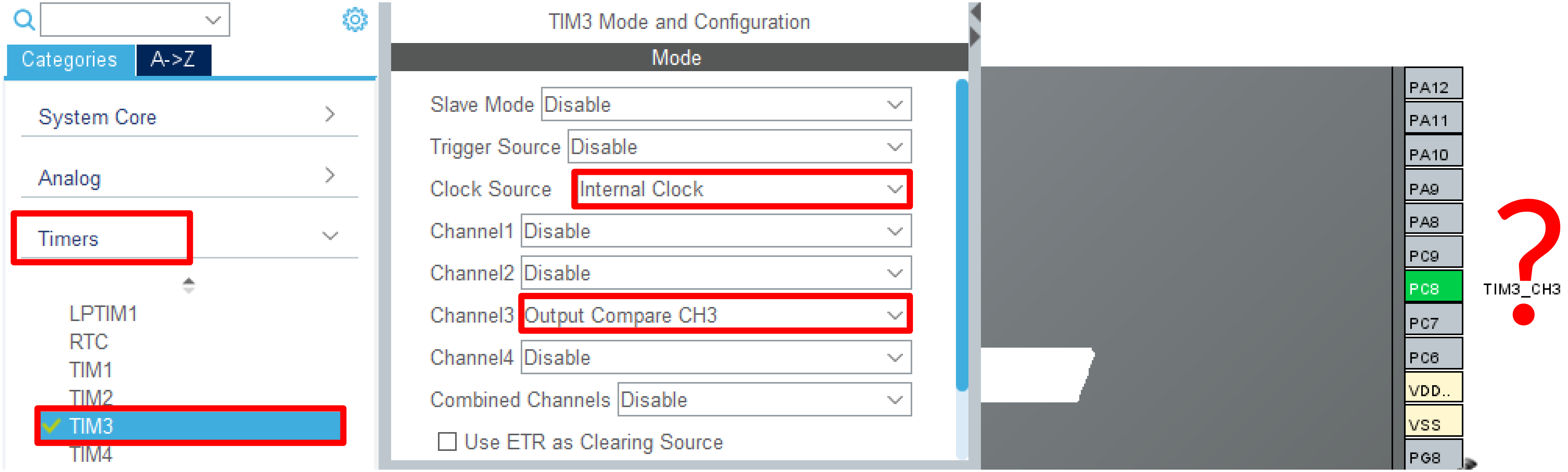
Answer: Timer 3 Chanel 3

Timers

Question: What value should I choose to the Prescaler Register (PSR) and to the Auto-reload Register (ARR)?



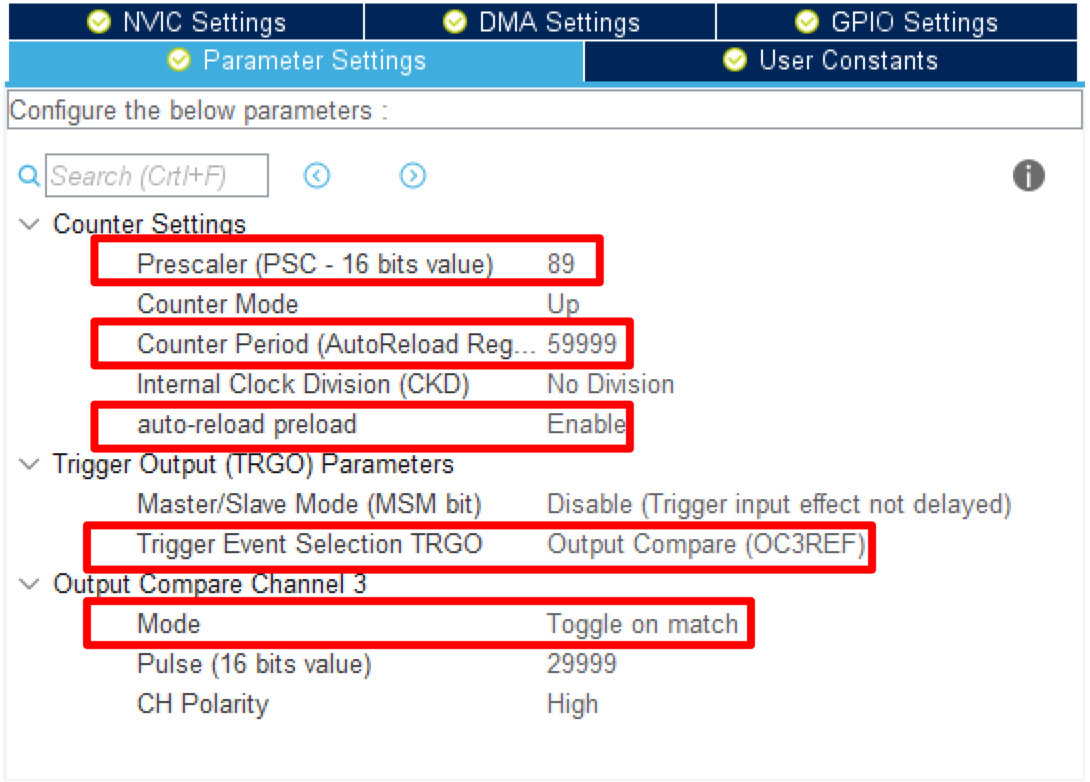
Timers



Wait!!! The output pin should appear at PB0...

Drag it to there

Timers

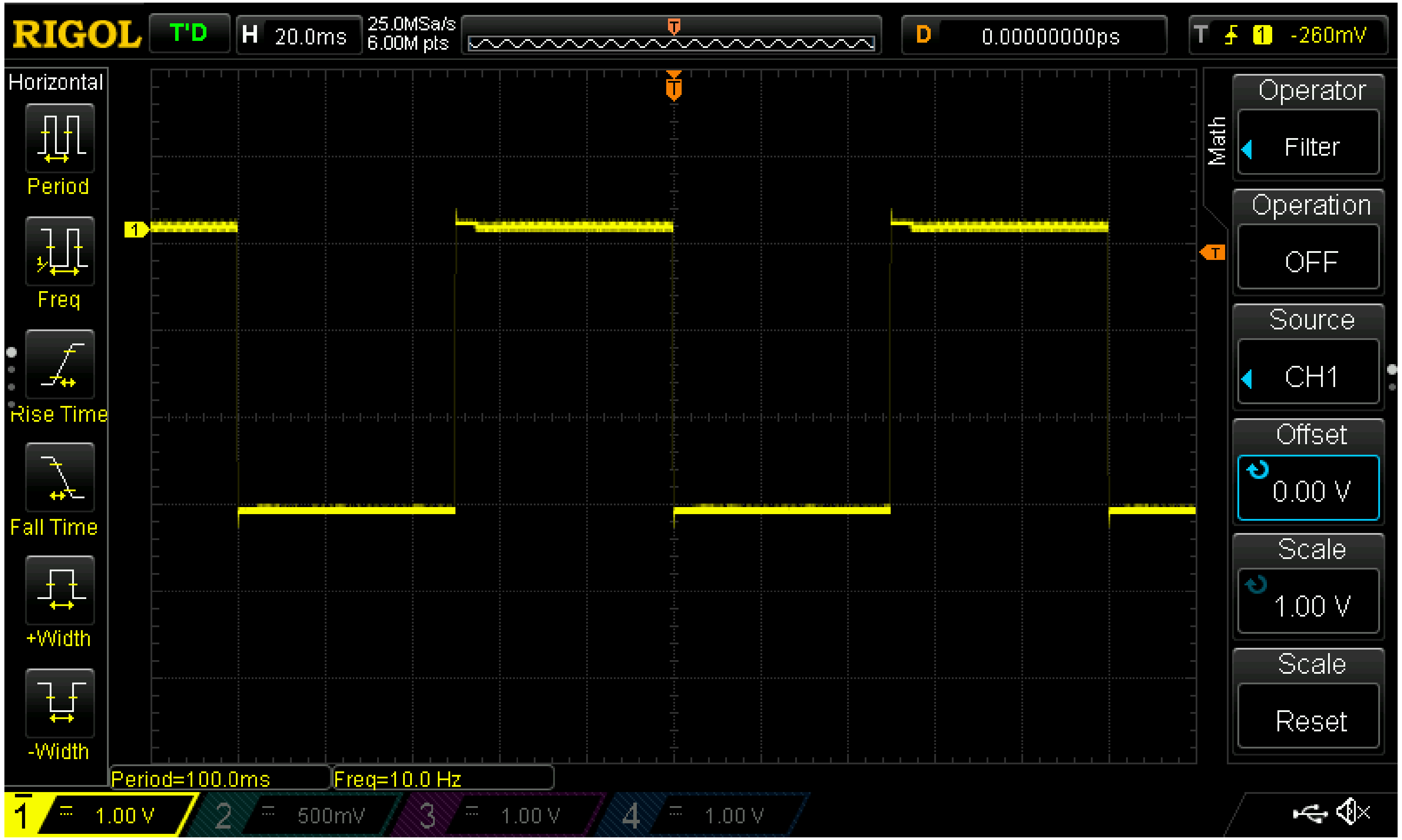

NVIC Settings
DMA Settings
GPIO Settings
Parameter Settings
User Constants
Configure the below parameters :
Search (Crtl+F)
Counter Settings
Prescaler (PSC - 16 bits value) 89
Counter Mode Up
Counter Period (AutoReload Reg... 59999
Internal Clock Division (CKD) No Division
auto-reload preload Enable
Trigger Output (TRGO) Parameters
Master/Slave Mode (MSM bit) Disable (Trigger input effect not delayed)
Trigger Event Selection TRGO Output Compare (OC3REF)
Output Compare Channel 3
Mode Toggle on match
Pulse (16 bits value) 29999
CH Polarity High

Timers

```c
/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_TIM3_Init();
/* USER CODE BEGIN 2 */
HAL_TIM_OC_Start(&htim3, TIM_CHANNEL_3);
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
  /* USER CODE END WHILE */
```

Timers


RIGOL
T'D
H 20.0ms
25.0MSa/s
6.00M pts
D 0.00000000ps
T 1 -260mV
Horizontal
Period
Freq
Rise Time
Fall Time
+Width
-Width
Math
Operator
Filter
Operation
OFF
Source
CH1
Offset
0.00 V
Scale
1.00 V
Scale
Reset
Period=100.0ms
Freq=10.0 Hz
1 1.00 V
2 500mV
3 1.00 V
4 1.00 V

Timers

Example 3: Use pulse-width modulation (PWM) to make a Dimming Led, with the blue Led.

Question: Is there a channel of some timer connected to the PB7 pin?

Tip: DS11532, page 90

Answer: Timer 4, channel 2

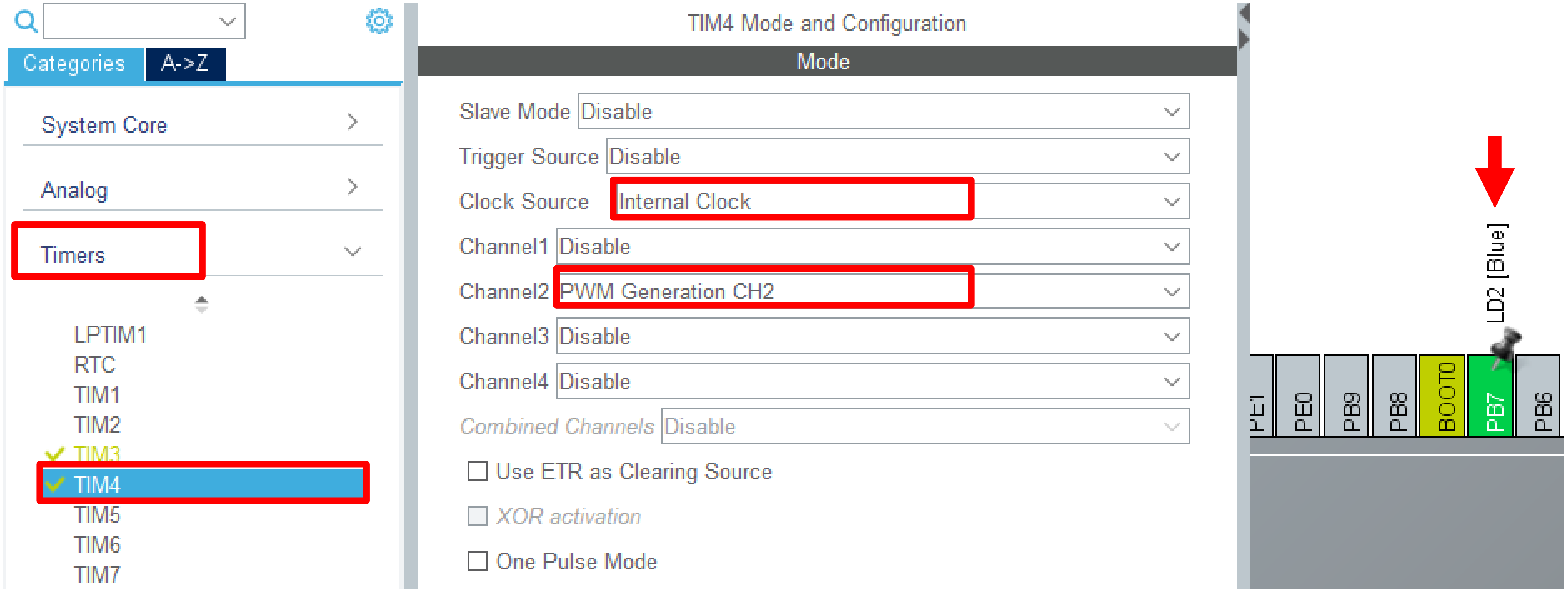
Timers

Question: Which PSC and ARR should I use?

Answer: There are trade-offs between very high frequencies (for example the usual L298N, which has a maximum frequency of 25kHz) and a very low frequencies. Different hardware might require different frequencies. Try to find a good frequency for your hardware and use that. Ideally, you should select an ARR such that it will be easy to convert from a duty-cycle percentage value.

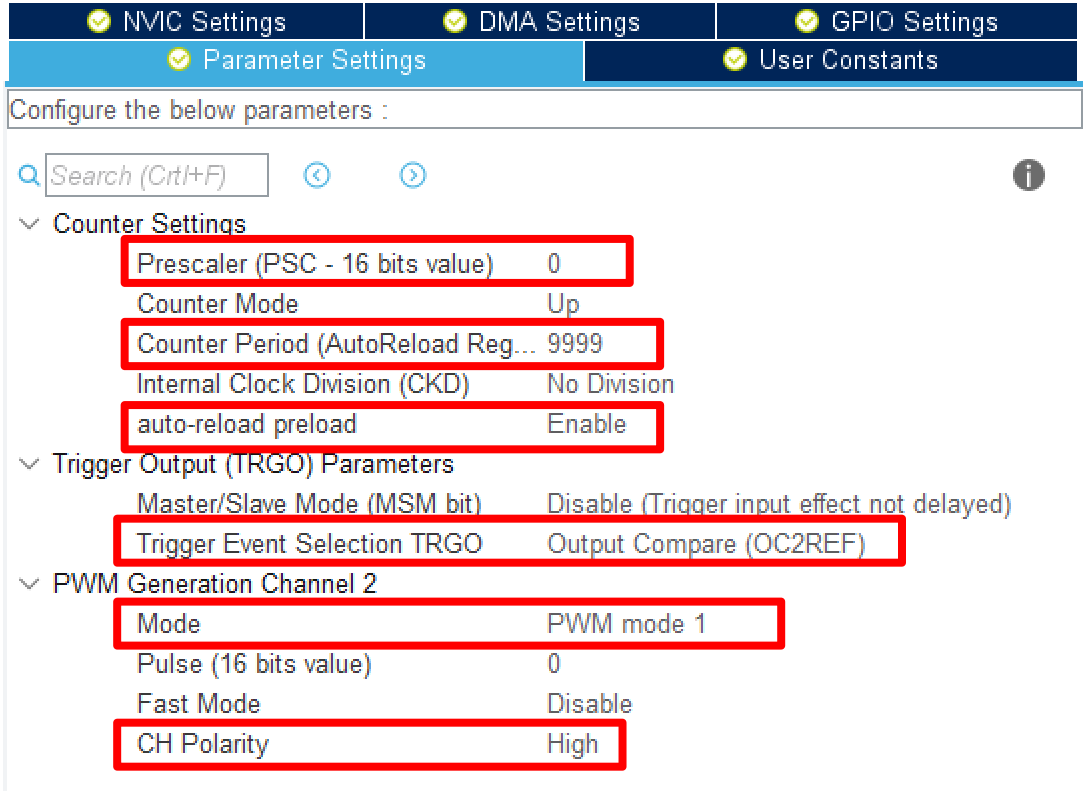
Lets select a frequency of 10kHz. PSC = 0; ARR = 9999. The frequency is 10,8kHz

Timers



Categories
A->Z
System Core
Analog
Timers
LPTIM1
RTC
TIM1
TIM2
TIM3
TIM4
TIM5
TIM6
TIM7
TIM4 Mode and Configuration
Mode
Slave Mode Disable
Trigger Source Disable
Clock Source Internal Clock
Channel1 Disable
Channel2 PWM Generation CH2
Channel3 Disable
Channel4 Disable
Combined Channels Disable
Use ETR as Clearing Source
XOR activation
One Pulse Mode
LD2 [Blue]
PE0
PB9
PB8
BOOT0
PB7
PB6

Timers


NVIC Settings
DMA Settings
GPIO Settings
Parameter Settings
User Constants
Configure the below parameters :
Search (Crtl+F)
Counter Settings
Prescaler (PSC - 16 bits value) 0
Counter Mode Up
Counter Period (AutoReload Reg... 9999
Internal Clock Division (CKD) No Division
auto-reload preload Enable
Trigger Output (TRGO) Parameters
Master/Slave Mode (MSM bit) Disable (Trigger input effect not delayed)
Trigger Event Selection TRGO Output Compare (OC2REF)
PWM Generation Channel 2
Mode PWM mode 1
Pulse (16 bits value) 0
Fast Mode Disable
CH Polarity High

Timers

```c
/* USER CODE BEGIN 2 */
HAL_TIM_OC_Start(&htim3, TIM_CHANNEL_3);
HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_2);
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1) {
  int i;
  for(i = 0; i<9999; i += 10){
    __HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_2, i);
    HAL_Delay(1);
  }
  /* USER CODE END WHILE */
```

Analog-to-Digital Converter

STM32 F767ZI provides 3 Analog-to-Digital Converter (ADC). This peripheral is able to acquire several analog input voltages, compare them to a reference, and to convert them to a number.

The ADC's in this board have 12-bit of resolution, therefore we have a range of 4095 different values. We have that 3300mV is represented with 4095. The minimum step is 3300/4095 ≈ 0.8mV. To calculate the voltage with the representation value(x):

$$Vin = \frac{Vref}{4095} * x$$

Analog-to-Digital Converter

The time needed to perform a conversion depends on the selected resolution. The sampling time, in fact, is defined by a fixed number of cycles (3) plus a variable number of cycles depending the A/D resolution (12 bits → 15 clock cycles; 10 → 13; 8 → 11; 6 → 9).

ADCs implemented in STM32 MCUs provide several conversion modes useful to deal with different application scenarios. Now we are going to briefly introduce the most relevant of them (all modes that require DMA will not appear in this presentation). All about ADC in RM0410, page 438. You can read about ADC API in UM1905, page 88.

Analog-to-Digital Converter

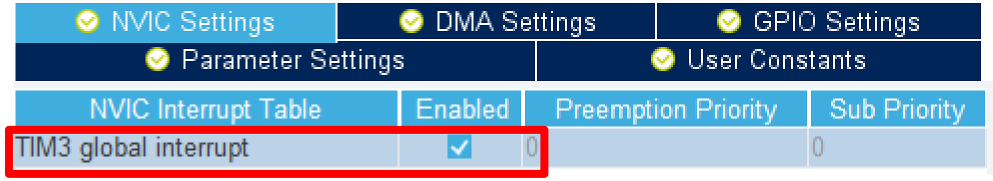
Example 4: Create a program that reads the processor temperature every 0,5sec and send it through the serial port.

Tip: DS11532, page 51

The temperature sensor is connected to ADC1 channel 18 (isn't the most accurate, but it's good enough to make an example with it).

Analog-to-Digital Converter

Problem: we have PWM duty-cycles updates in polling, let’s change that by creating an interrupt that will change the duty-cycle.



NEEEICUM
núcleo de estudantes de engenharia eletrónica industrial e computadores da universidade do minho

Analog-to-Digital Converter

Tim.c:

```
/* USER CODE BEGIN 0 */
static int i = 0;
/* USER CODE END 0 */
```

```
/* USER CODE BEGIN 1 */

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
  if(htim->Instance == TIM3){
    __HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_2, i);
    i += 200;
    if (i > 9999)
      i = 0;
  }
}

/* USER CODE END 1 */
```

Analog-to-Digital Converter

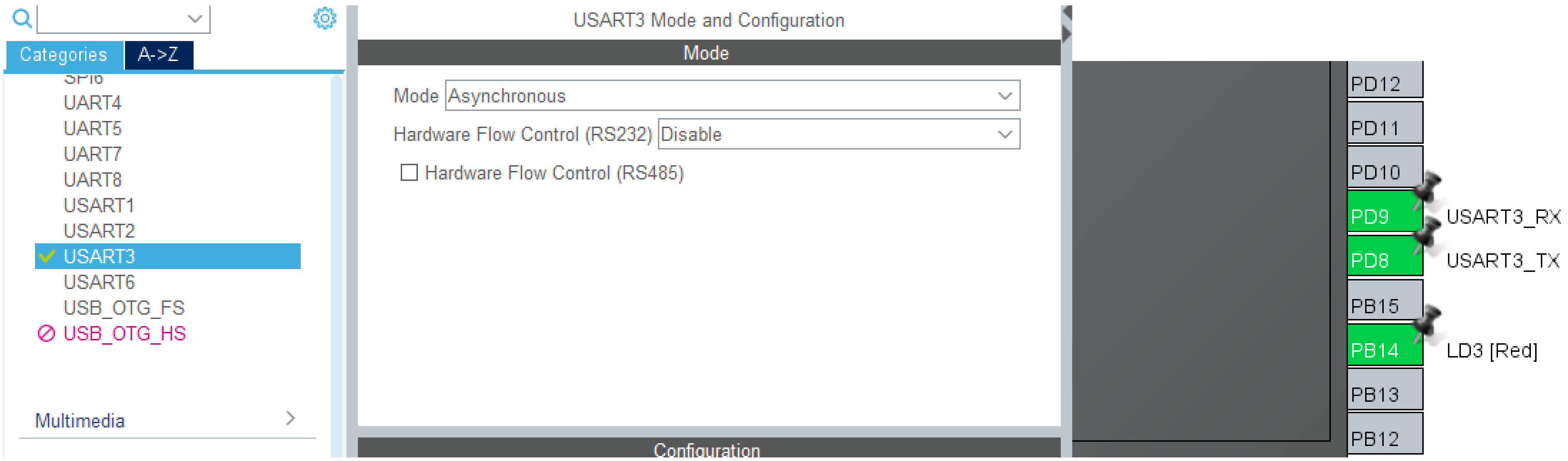
main.c:

```
HAL_TIM_Base_Start_IT(&htim3);
HAL_TIM_OC_Start(&htim3, TIM_CHANNEL_3);
HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_2);
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
  /* USER CODE END WHILE */
```

Analog-to-Digital Converter

We'll need a UART (USART 3)



Analog-to-Digital Converter

Redefine the fputc function in usart.c

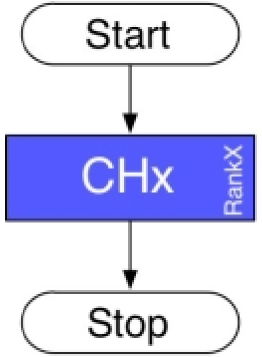
```c
/* USER CODE BEGIN 1 */

int fputc(int ch, FILE *f){
  HAL_UART_Transmit(&huart3, (uint8_t *)&ch, 1, 100);
  return ch;
}
```

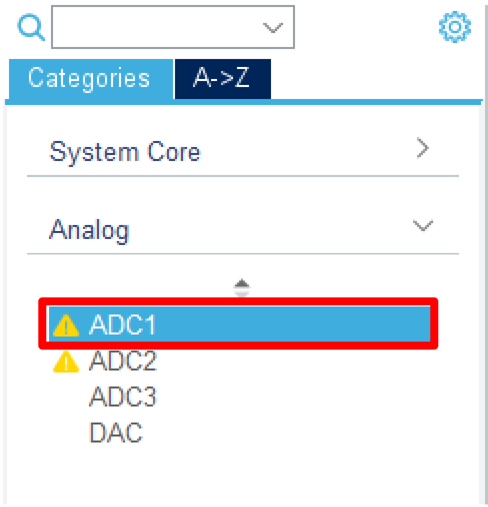
Analog-to-Digital Converter

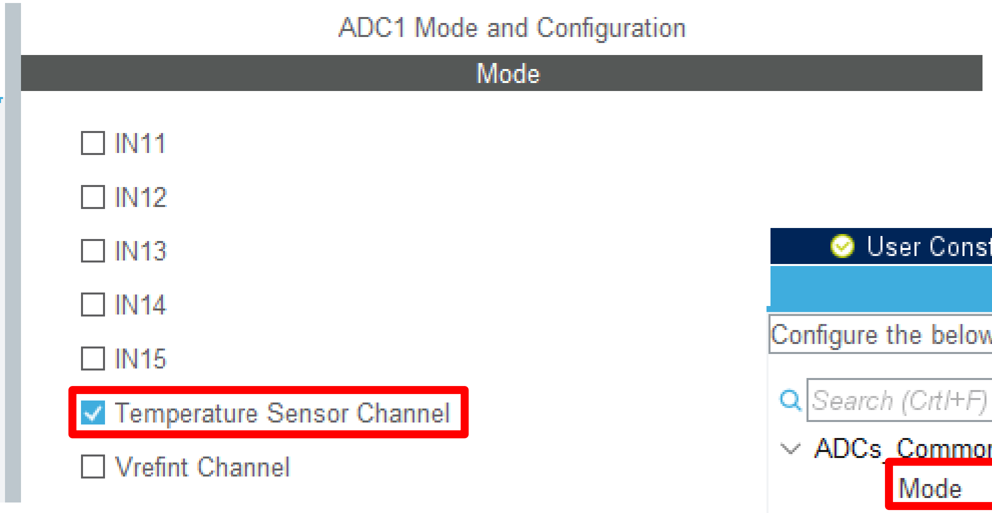
Single-Channel, Single Conversion Mode:

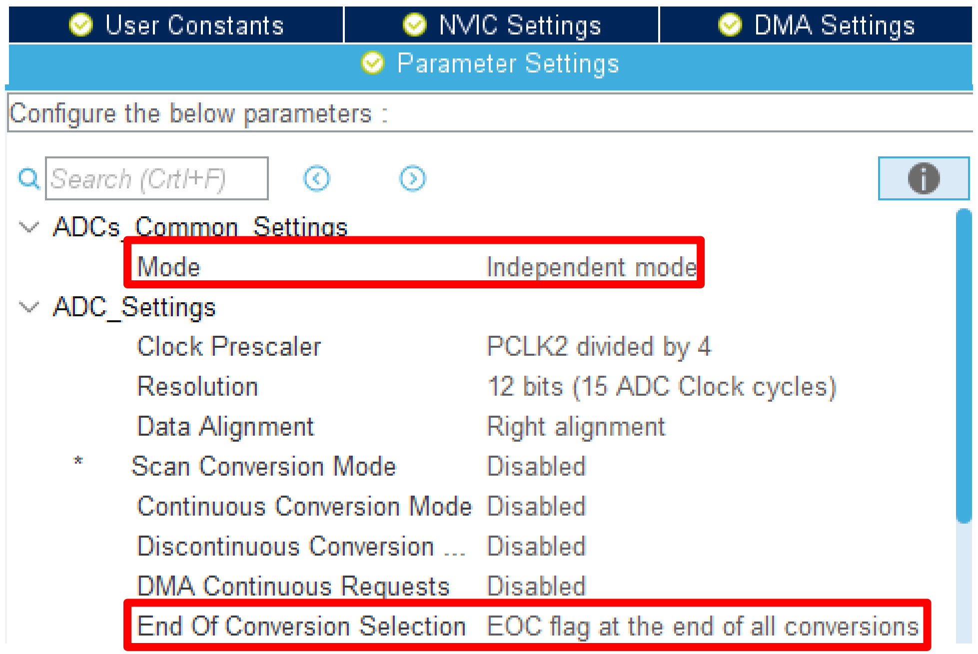
- This is the simplest ADC mode. In this mode, the ADC performs a single conversion (one sample) of a single channel and stops when that conversion is finished.



Analog-to-Digital Converter







NEEEICUM
núcleo de estudantes de engenharia eletrónica industrial e computadores da universidade do minho

Analog-to-Digital Converter

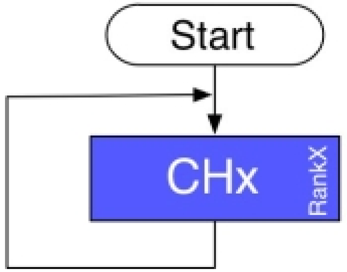
For more info about the formula to calculate the temperature go to RM0410, page 466 and to DS11532, page 171

```
/* USER CODE BEGIN WHILE */
while (1) {
  uint16_t adcValue = 1;
  double temp;
  if (HAL_ADC_Start(&hadc1) == HAL_OK){
    if (HAL_ADC_PollForConversion(&hadc1, 1000) == HAL_OK)
      adcValue = HAL_ADC_GetValue(&hadc1);
    HAL_ADC_Stop(&hadc1);
  }
  temp = ((((double) adcValue * 3300 / 4095) - 760.0) / 2.5) + 25;
  printf("adcValue: %hu\r\n", adcValue);
  printf("Temperature: %0.2lf %cC\r\n", temp, 0xBA);
  HAL_Delay(500);
  /* USER CODE END WHILE */
```

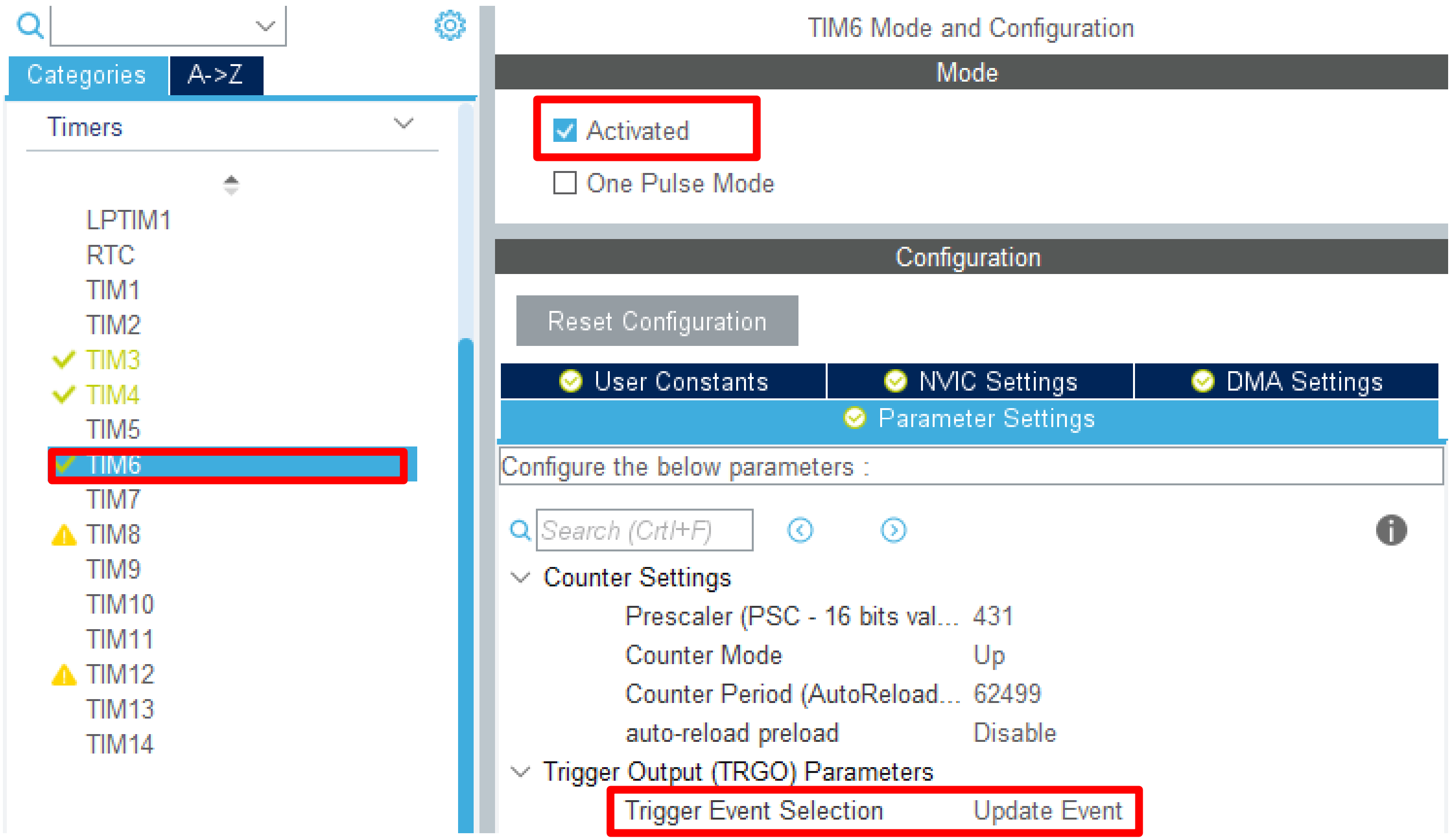
Analog-to-Digital Converter

Single-Channel, Continuous Conversion Mode

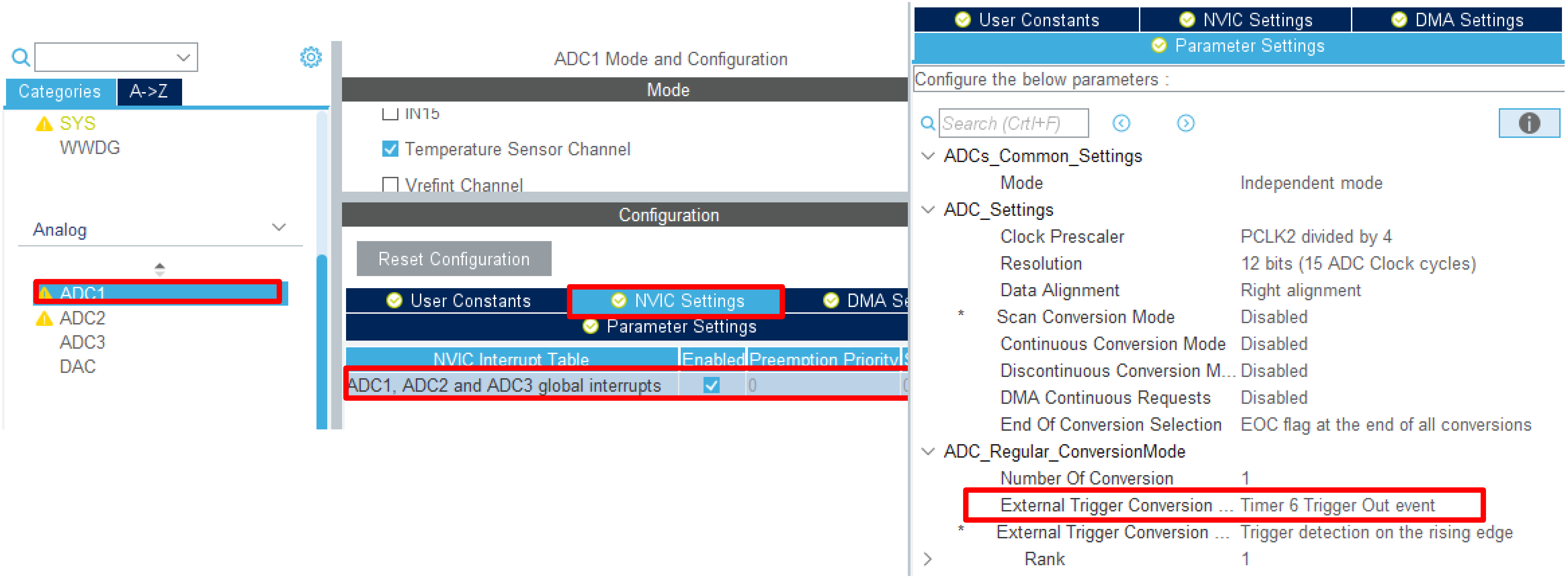
- This mode converts a single channel continuously and indefinitely in regular channel conversion. The continuous mode feature allows the ADC to work in the background. The ADC converts the channels continuously with less CPU intervention.



Analog-to-Digital Converter



Analog-to-Digital Converter



Analog-to-Digital Converter

In adc.c:

```c
/* USER CODE BEGIN 0 */
volatile uint32_t adcValue;
volatile uint8_t adcFlag;
/* USER CODE END 0 */
```

```c
/* USER CODE BEGIN 1 */

void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc){
  if(hadc->Instance == ADC1){
    adcValue = HAL_ADC_GetValue(&hadc1);
    adcFlag = 1;
  }
}

/* USER CODE END 1 */
```

Analog-to-Digital Converter

In adc.h:

```c
/* USER CODE BEGIN Private defines */
extern volatile uint32_t adcValue;
extern volatile uint8_t adcFlag;
/* USER CODE END Private defines */
```

Analog-to-Digital Converter

In main.c:

```c
HAL_TIM_Base_Start_IT(&htim6);
HAL_ADC_Start_IT(&hadc1);
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1) {
  double temp;
  if(adcFlag){
    adcFlag = 0;
    temp = ((((double) adcValue * 3300 / 4095) - 760.0) / 2.5) + 25;
    printf("adcValue: %u\r\n", adcValue);
    printf("Temperature: %0.2lf %cC\r\n", temp, 0xBA);
  }
  /* USER CODE END WHILE */
```

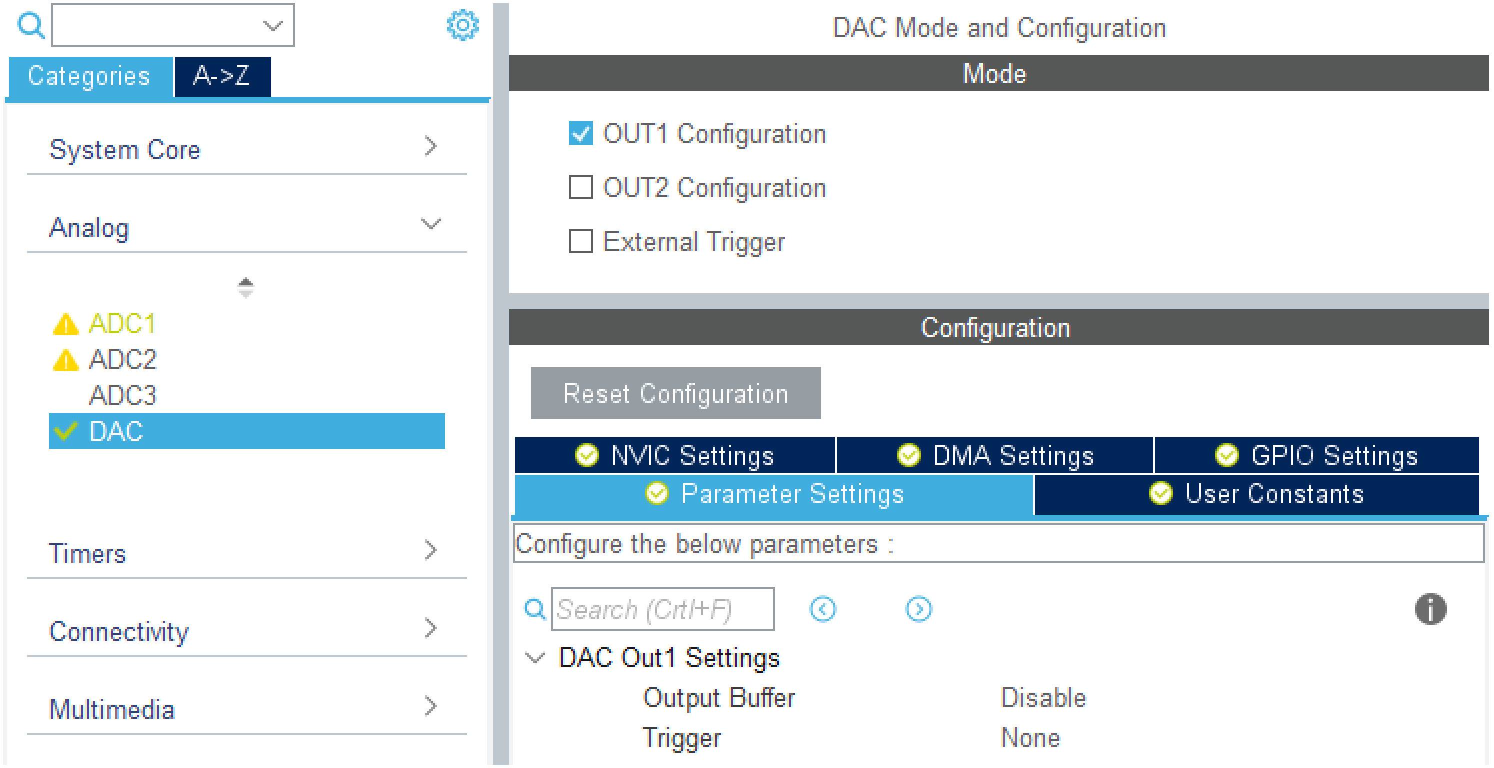
Digital-to-Analog Converter

A Digital to Analog Converter (DAC) converts a digital value to an analog one. DAC channels can be configured to work in 8/12-bit mode and the conversion of the two channels can be performed independently or simultaneously. Like the ADC peripheral, the DAC can also be triggered by a dedicated timer, in order to generate analog signals at a given frequency.

STM32F767ZI microcontrollers provide only a single dual channel DAC. The documentation is in RM0410 at page 486. The API for DAC is at page 198 in UM1905.

Digital-to-Analog Converter

Example 5: Put the value obtained by the ADC into an analog output.



Digital-to-Analog Converter

```c
/* USER CODE BEGIN WHILE */
while (1) {
  double temp;
  if(adcFlag){
    adcFlag = 0;
    temp = ((((double) adcValue * 3300 / 4095) - 760.0) / 2.5) + 25;
    printf("adcValue: %u\r\n", adcValue);
    printf("Temperature: %0.2lf %cC\r\n", temp, 0xBA);
    if(HAL_DAC_GetState(&hdac) != HAL_DAC_STATE_READY){
      HAL_DAC_Stop(&hdac,DAC1_CHANNEL_1);
    }
    if(HAL_DAC_Start(&hdac,DAC1_CHANNEL_1) == HAL_OK){
      HAL_DAC_SetValue(&hdac, DAC1_CHANNEL_1, DAC_ALIGN_12B_R, adcValue);
    }
  }
  /* USER CODE END WHILE */
```

Conclusion

Now that you have completed two lessons of this workshop, you've learned almost all the basics about STM32 (at least you are supposed to).

So you are ready to continue your study.

Keep in mind, there is much more to learn, this is just the beginning!

Resources

Timer Calculator:

- https://libstock.mikroe.com/projects/view/398/timer-calculator