

NEEEICUM

núcleo de estudantes de engenharia  
eletrônica industrial e computadores  
da universidade do minho

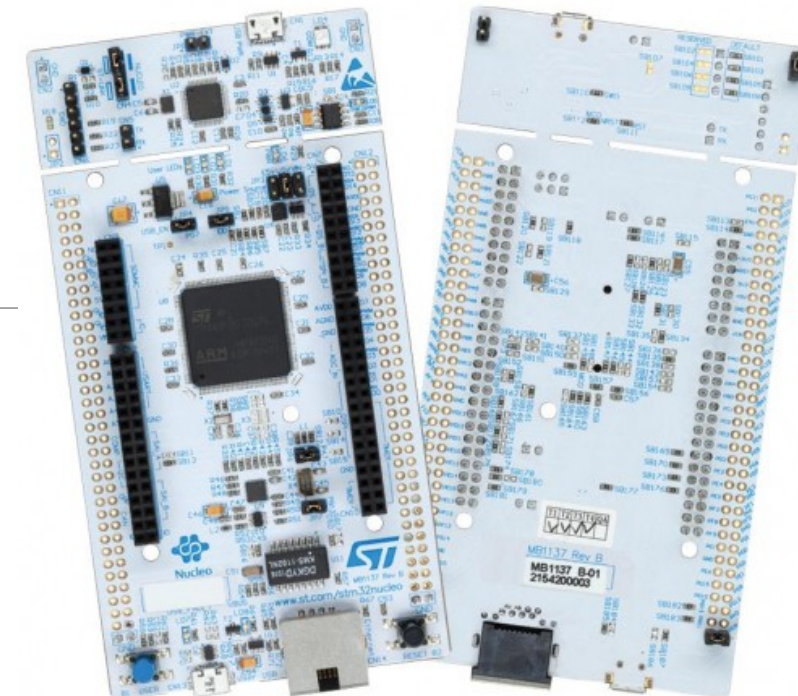
ESRG

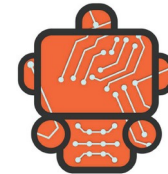
EMBEDDED SYSTEMS  
RESEARCH  
GROUP

# STM 32 F767ZI

CLOCK, TIMER AND ANALOG INTERFACE

ÁLVARO CASTRO LEITE  
DEC 2021





# Requirements

---

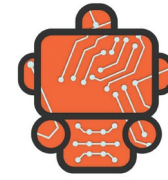
Micro-USB cable

STM32 development board

PC with all the tools installed

PC with terminal

Know everything from last workshop session



# Agenda

---

Skill level: Beginner

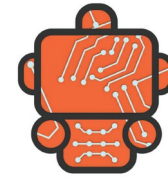
Clocks

Clock Tree

Timers

Analog-to-Digital Converter

Digital-to-Analog Converter



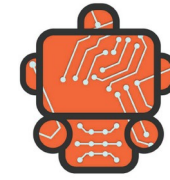
# Clock

---

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	<input type="checkbox"/>	0	0

↓

RCC\_OSC32\_IN

RCC\_OSC32\_OUT

PC14/..

PC15/..

PF0

PF1

PF2

PF3

PF4

PF5

VSS

VDD

PF6

PF7

PF8

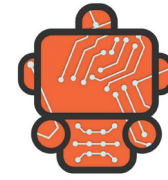
PF9

PF10

PH0/..

PH1/..

NRST

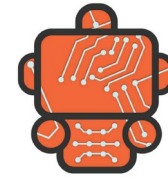


# Clock Tree

---

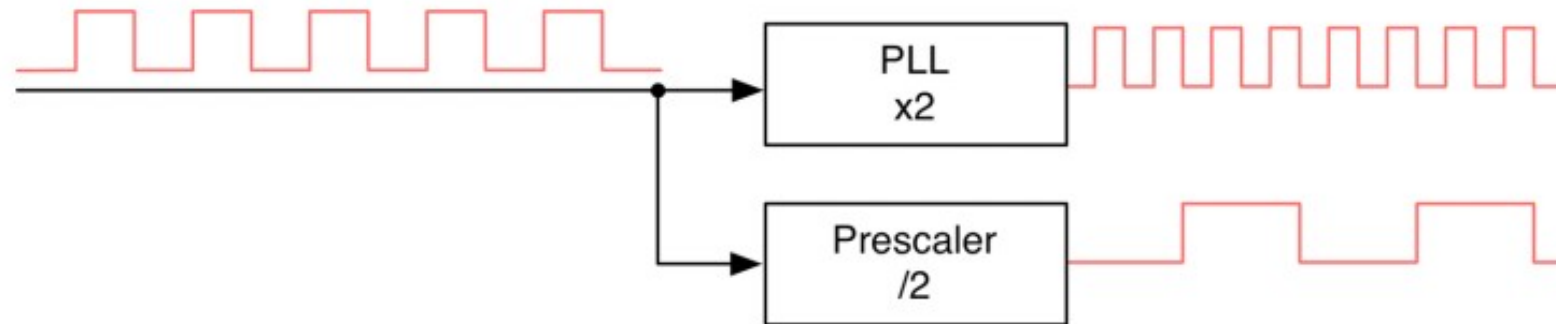
The same clock signal, however, cannot be used to feed all components and peripherals provided by a modern microcontroller like STM32. A sophisticated distribution network, also called **clock tree**, is responsible for managing and feeding the signals inside an STM32 MCU.

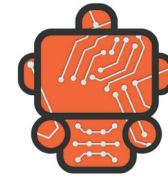
Neither of the Cortex-M core nor of the other peripherals frequency is establish by the frequency of the high-speed oscillator.



# 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

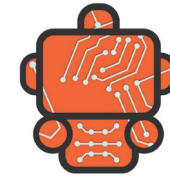
---

Question: What frequency the STM32 Core is working at?

Tip: Open “Clock Configuration”

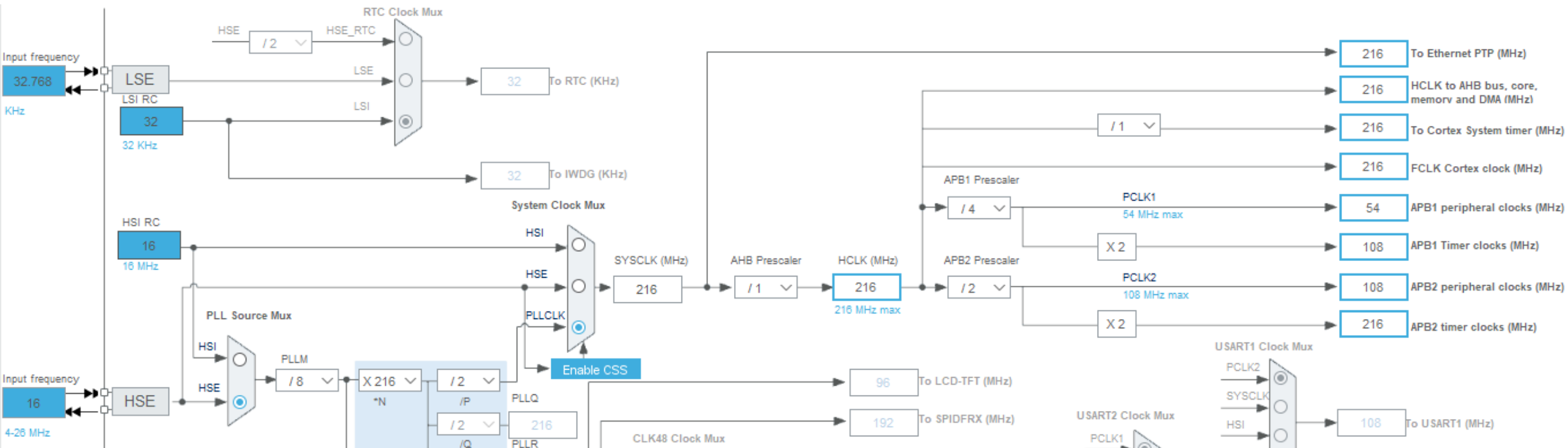
Answer: 16MHz

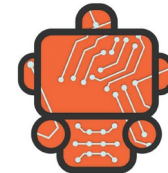




# Clock Tree

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

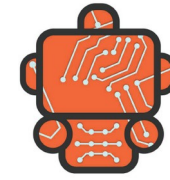




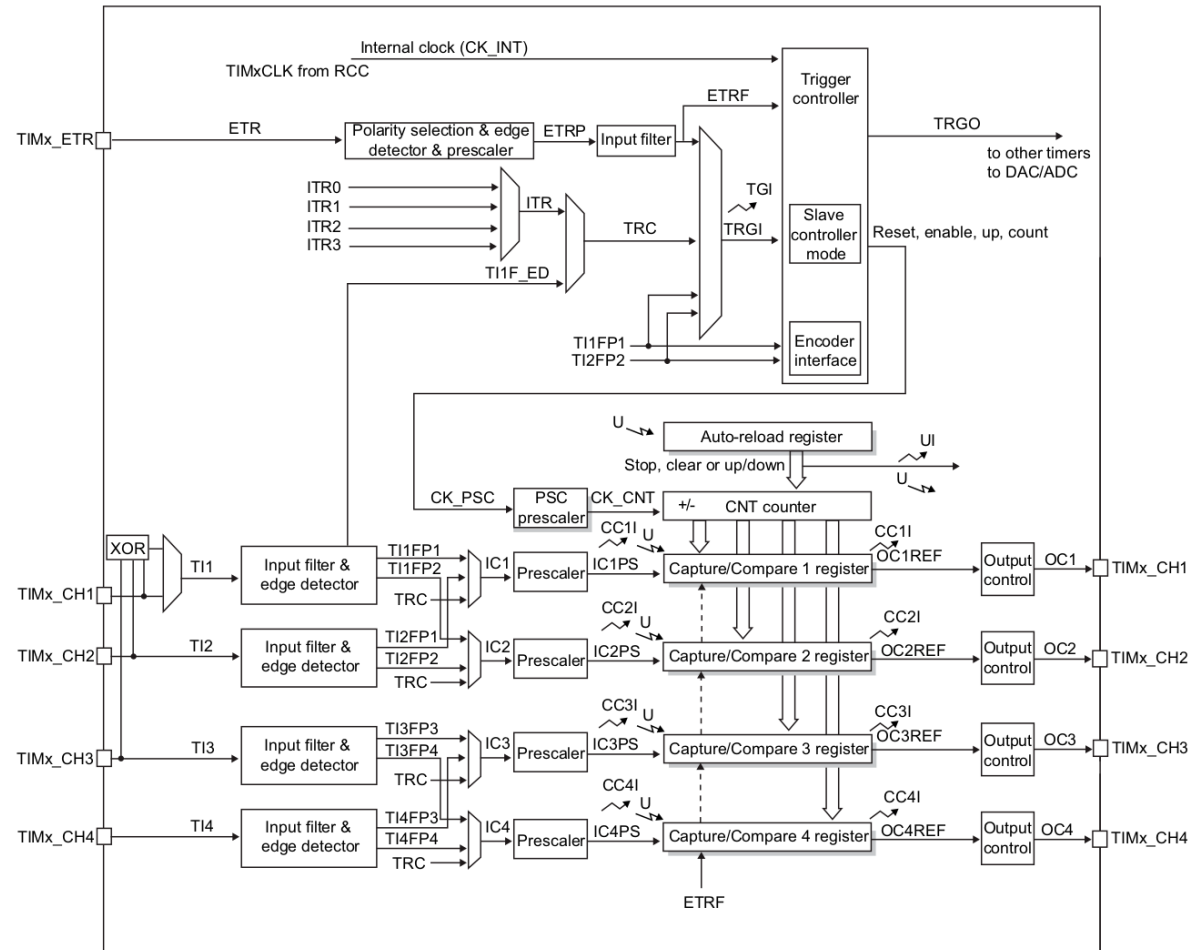
# Timers

---

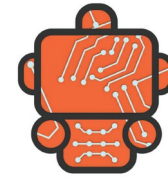
A timer is a free-running counter with a counting frequency that is a fraction of its source clock. The counting speed can be reduced using a dedicated prescaler for each timer. Depending on the timer type, it can be clocked by the internal clock (which is derived from the bus where it is connected),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



# Timers



RM0410,  
page 962

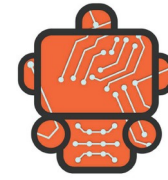


# Timers

---

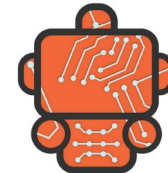
STM32 timers can mainly be grouped into a few categories. Let us take a brief look at some of them:

- **Basic timers:** these 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 used like as a basic timer.



# Timers

Timer type		STM32 F04x /F070x6 /F03x (exclud- ing /F030x8 and /F030x)	STM32 F030xB /F030x8 /F05x /F09x /F07x (exclud- ing 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 (exclud- ing /F401, /F411, /F410)	STM32 F401 /F411 /F410	STM32 F30X /F3x8 (exclud- ing /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 <sup>(1)</sup> TIM8 <sup>(1)</sup>	TIM1 <sup>(1)</sup> TIM8 <sup>(1)</sup>	TIM1	TIM1 TIM8	TIM1	TIM1 TIM8 <sup>(1)</sup> TIM20 <sup>(1)</sup>	-	TIM1	TIM1 TIM8 <sup>(1)</sup>	TIM1 TIM8	-	-	-	TIM1 TIM8 <sup>(1)</sup>
General purpose	32- bit	TIM2	TIM2	-	-	-	TIM2 TIM5	TIM2 <sup>(1)</sup> TIM5	TIM2	TIM2 TIM5	TIM2	TIM2	TIM2 TIM5	-	-	TIM5 <sup>(1)</sup>	TIM2 TIM5 <sup>(1)</sup>
	16- bit	TIM3	TIM3	TIM2 TIM3 TIM4 TIM5	TIM2 TIM3 TIM4 <sup>(1)</sup> TIM5 <sup>(1)</sup>	TIM2 TIM3 TIM4 TIM5 <sup>(1)</sup>	TIM3 TIM4	TIM3 <sup>(1)</sup> TIM4 <sup>(1)</sup>	TIM3 <sup>(1)</sup> TIM4 <sup>(1)</sup> TIM19 <sup>(1)</sup>	TIM3 TIM4 TIM19	TIM3	TIM3 TIM4	TIM3 TIM4	TIM2 TIM3 <sup>(1)</sup>	TIM2	TIM2 TIM3 TIM4	TIM3 <sup>(1)</sup> TIM4 <sup>(1)</sup>
Basic		-	TIM6 TIM7 <sup>(1)</sup>	TIM6 TIM7	TIM6 <sup>(1)</sup> TIM7 <sup>(1)</sup>	TIM6 TIM7	TIM6 TIM7	TIM6 <sup>(1)</sup>	TIM6 TIM7 <sup>(1)</sup>	TIM6 TIM7 TIM18	TIM6 TIM7	TIM6 TIM7 <sup>(1)</sup>	TIM6 TIM7	TIM6 TIM7 <sup>(1)</sup>	-	TIM6 TIM7	TIM6 TIM7
1 channel		TIM14	TIM14	TIM10 TIM11 TIM13 TIM14	-	TIM13 <sup>(1)</sup> TIM14 <sup>(1)</sup>	TIM10 TIM11 TIM13 TIM14	TIM10 <sup>(1)</sup> TIM11	-	TIM13 TIM14	-	-	TIM10 TIM11 TIM13 TIM14	-	-	TIM10 TIM11	-
2-channel		-	-	TIM9 TIM12	-	TIM12 <sup>(1)</sup>	TIM9 TIM12	TIM9	-	TIM12	-	-	TIM9 TIM12	TIM21 TIM22	TIM21 TIM22 <sup>(1)</sup>	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 <sup>(1)</sup>
Low-power timer		-	-	-	-	-	-	LPTIM1 <sup>(1)</sup>	-	-	-	-	LPTIM1	LPTIM1	LPTIM1	-	LPTIM1 LPTIM2

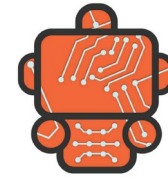


# Timers

---

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

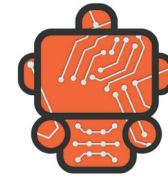
---

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

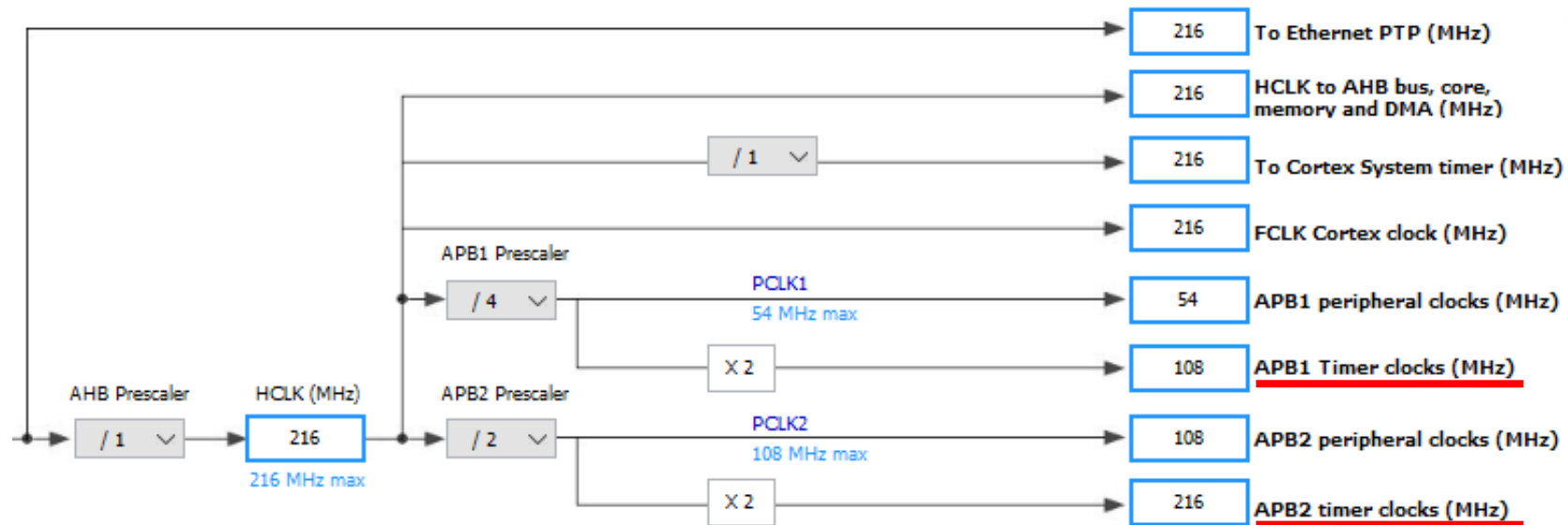
Tip: Take a look at DS11532, page 19

Answer:

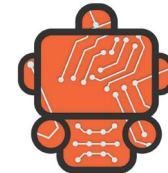
<i>Timer</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>	<i>7</i>	<i>8</i>	<i>9</i>	<i>10</i>	<i>11</i>	<i>12</i>	<i>13</i>	<i>14</i>
APB1 (108 MHz)		x	x	x	x	x	x					x	x	x
APB2 (216 MHz)	x							x	x	x	x			



# Timers







# Timers

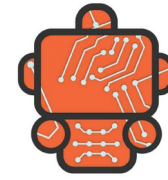
---

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)?

The screenshot shows the MikroElektronika - Timer Calculator application. The interface includes a sidebar with four numbered steps: 1. Select device (STM32F7xx), 2. MCU clock frequency (108 MHz / 1), 3. Choose timer (Timer3), and 4. Frequency (20 Hz). A large 'Calculate' button is at the bottom of the sidebar. The main area displays the generated code for Timer3, with a red box highlighting the lines `TIM3_PSC = 89;` and `TIM3_ARR = 59999;`. A status bar at the bottom shows 'Loaded File: <None>'.

**Timer Calculator**

1 Select device: STM32F7xx

2 MCU clock frequency: 108 MHz / 1

3 Choose timer: Timer3

4 Frequency: 20 Hz

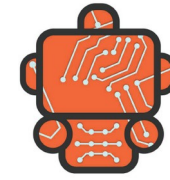
**Calculate**

Code generated successfully! [Copy To Clipboard](#)

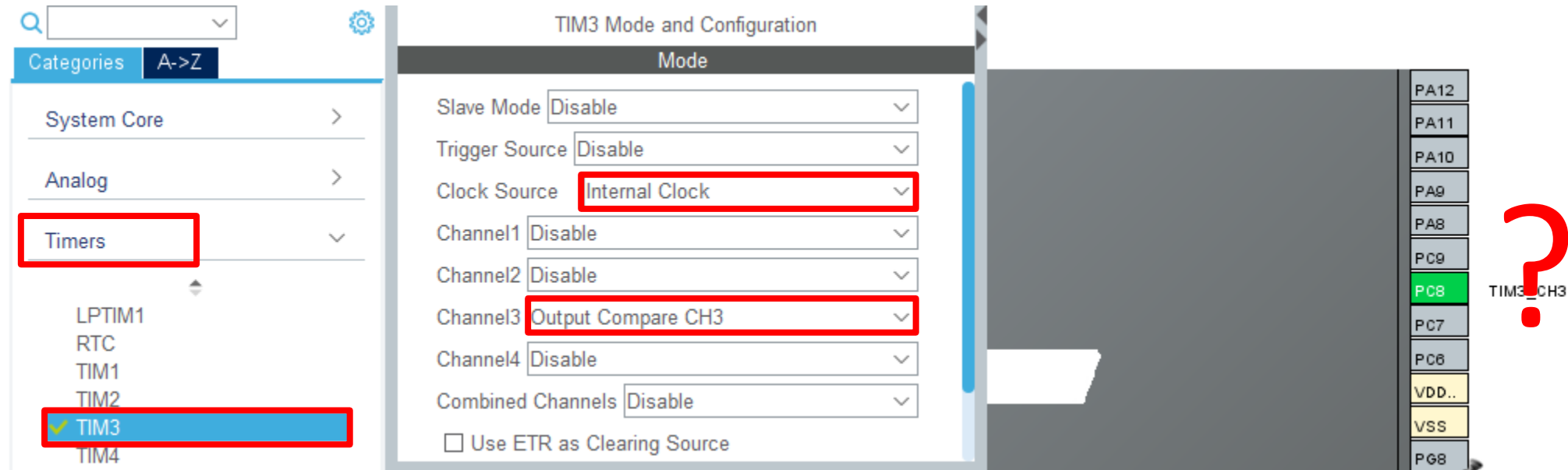
```
//Timer3 Prescaler :89; Preload = 59999; Actual  
Interrupt Time = 50 ms  
  
//Place/Copy this part in declaration section  
void InitTimer3(){  
    RCC_APB1ENR.TIM3EN = 1;  
    TIM3_CR1.CEN = 0;  
    TIM3_PSC = 89;  
    TIM3_ARR = 59999;  
    NVIC_IntEnable(IVT_INT_TIM3);  
    TIM3_DIER.UIE = 1;  
    TIM3_CR1.CEN = 1;  
}  
  
void Timer3_interrupt() iv IVT_INT_TIM3 {  
    TIM3_SR.UIF = 0;  
}
```

mikroC / mikroBasic / mikroPascal

F1 - Help F2 - About Esc - Exit Loaded File: <None>

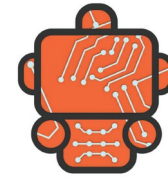


# Timers



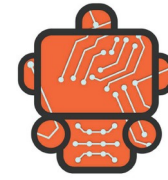
Wait!!! The output pin should appear at PB0...

Drag it to there



# Timers

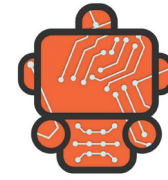
✓ NVIC Settings	✓ DMA Settings	✓ GPIO Settings
✓ Parameter Settings	✓ User Constants	
Configure the below parameters :		
<input type="text" value="Search (Ctrl+F)"/> <span>⏪</span> <span>⏩</span> <span>?</span>		
✓ Counter Settings		
Prescaler (PSC - 16 bits value)		89
Counter Mode		Up
Counter Period (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



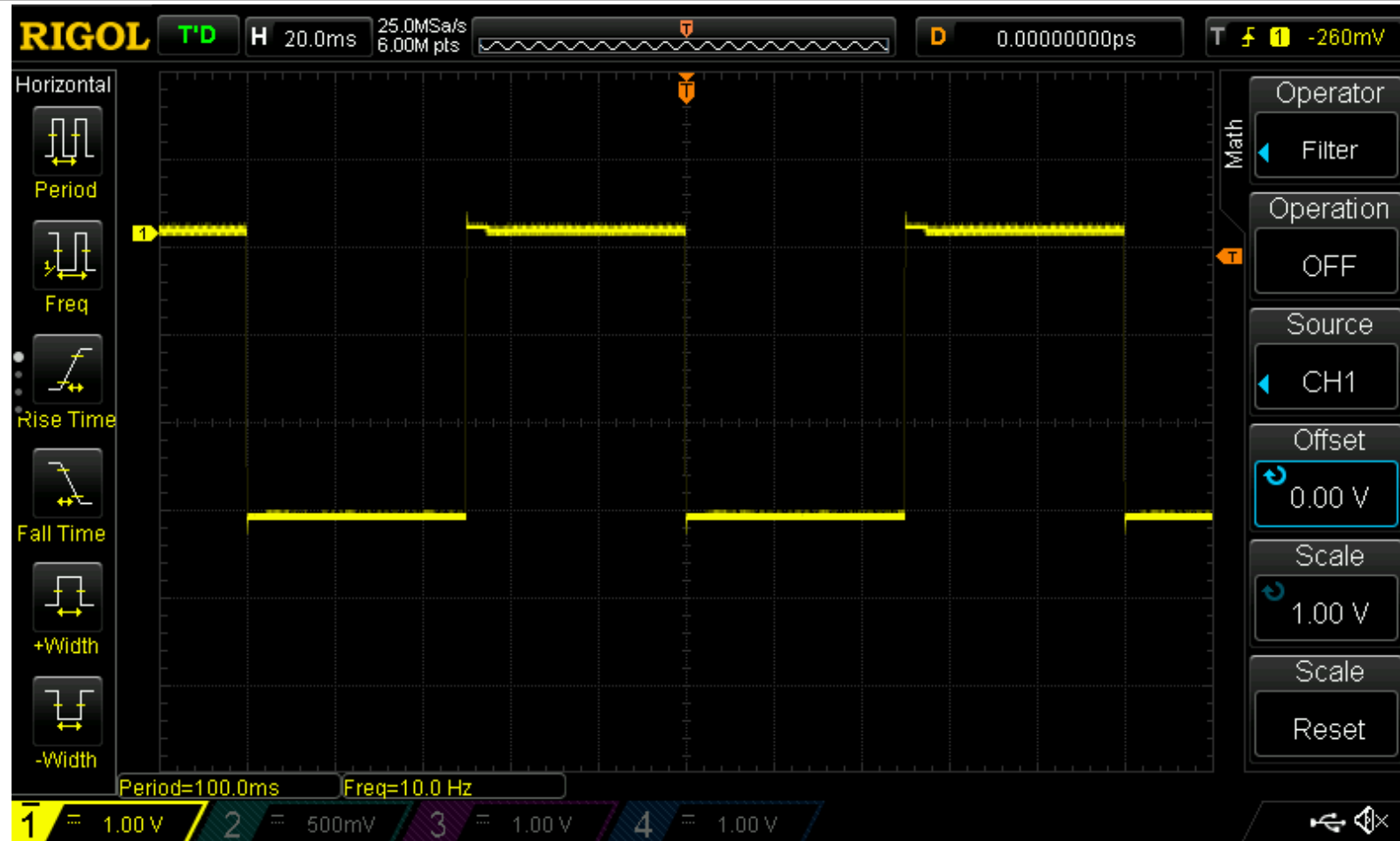
# Timers

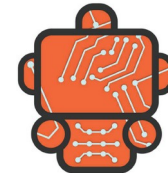
---

```
90  /* Initialize all configured peripherals */
91  MX_GPIO_Init();
92  MX_TIM3_Init();
93  /* USER CODE BEGIN 2 */
94  HAL_TIM_OC_Start(&htim3, TIM_CHANNEL_3);
95  /* USER CODE END 2 */
96
97  /* Infinite loop */
98  /* USER CODE BEGIN WHILE */
99  while (1)
100 {
101     /* USER CODE END WHILE */
```



# Timers





# Timers

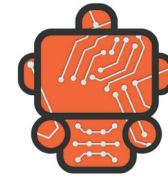
---

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

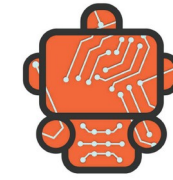
---

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

**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

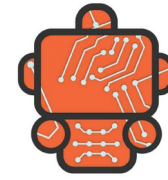
Combined Channels: Disable

☐ Use ETR as Clearing Source

☐ XOR activation

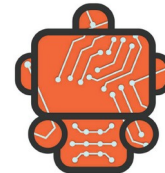
☐ One Pulse Mode

Pin List: PE1, PE0, PB9, PB8, BOOT0, **PB7 [Blue]**, PB6



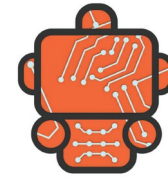
# Timers

✓ NVIC Settings	✓ DMA Settings	✓ GPIO Settings
✓ Parameter Settings	✓ User Constants	
Configure the below parameters :		
<input type="text" value="Search (Ctrl+F)"/> <span>⏪</span> <span>⏩</span> <span>?</span>		
✓ 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

```
101  /* USER CODE BEGIN 2 */
102  HAL_TIM_OC_Start(&htim3, TIM_CHANNEL_3);
103  HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_2);
104  /* USER CODE END 2 */
105
106  /* Infinite loop */
107  /* USER CODE BEGIN WHILE */
108  while (1) {
109      int i;
110      for(i = 0; i<9999; i += 10){
111          __HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_2, i);
112          HAL_Delay(1);
113      }
114      /* USER CODE END WHILE */
```



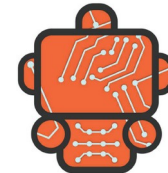
# Analog-to-Digital Converter

---

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  $3300/4095 \approx 0.8\text{mV}$ . To calculate the voltage with the representation value( $x$ ):

$$V_{in} = \frac{V_{ref}}{4095} * x$$

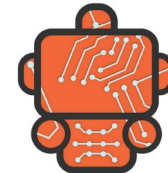


# Analog-to-Digital Converter

---

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  $\rightarrow$  15 clock cycles; 10  $\rightarrow$  13; 8  $\rightarrow$  11; 6  $\rightarrow$  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.



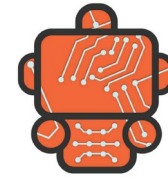
# Analog-to-Digital Converter

---

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

Tip: DS11532, page 51

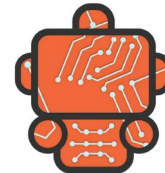
The temperature sensor is connected to ADC1 channel 18 (isn't the most accurate, but it's good enough to make an example with it).



# 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.

✔ NVIC Settings	✔ DMA Settings	✔ GPIO Settings	
✔ Parameter Settings	✔ User Constants		
NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
TIM3 global interrupt	✔	0	0



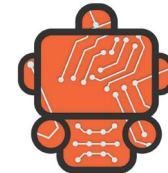
# Analog-to-Digital Converter

Tim.c:

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

231 /* USER CODE BEGIN 1 */
232
233 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
234     if(htim->Instance == TIM3) {
235         HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_2, i);
236         i += 200;
237         if (i > 9999)
238             i = 0;
239     }
240 }
241
242 /* USER CODE END 1 */
```

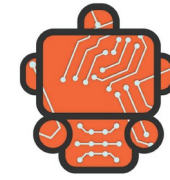




# Analog-to-Digital Converter

main.c:

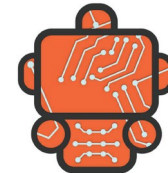
```
99     HAL_TIM_Base_Start_IT(&htim3);
100    HAL_TIM_OC_Start(&htim3, TIM_CHANNEL_3);
101    HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_2);
102    /* USER CODE END 2 */
103
104    /* Infinite loop */
105    /* USER CODE BEGIN WHILE */
106    while (1)
107    {
108        /* USER CODE END WHILE */
```



# Analog-to-Digital Converter

We'll need a UART (USART 3)

The screenshot displays the STM32CubeMX configuration tool. On the left, a sidebar lists various components under 'Categories'. 'USART3' is selected and highlighted with a blue bar and a checkmark. Below it, 'USB\_OTG\_HS' is marked with a red circle and a slash. The main area is titled 'USART3 Mode and Configuration'. It features a 'Mode' section with two dropdown menus: 'Mode' set to 'Asynchronous' and 'Hardware Flow Control (RS232)' set to 'Disable'. There is also an unchecked checkbox for 'Hardware Flow Control (RS485)'. To the right of the configuration panel is a pinout diagram of the microcontroller. It shows a vertical stack of pins: PD12, PD11, PD10, PD9, PD8, PB15, PB14, PB13, and PB12. PD9 and PD8 are highlighted in green and labeled 'USART3\_RX' and 'USART3\_TX' respectively. PB14 is also highlighted in green and labeled 'LD3 [Red]'. The bottom of the interface shows a 'Configuration' tab.

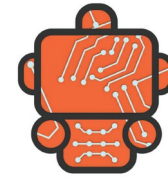


# Analog-to-Digital Converter

---

Redefine the fputc function in usart.c

```
104  /* USER CODE BEGIN 1 */
105
106  int fputc(int ch, FILE *f){
107      HAL_UART_Transmit(&huart3, (uint8_t *)&ch, 1, 100);
108      return ch;
109  }
```

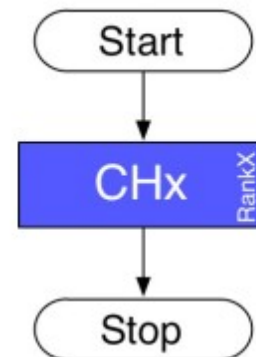


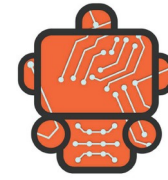
# Analog-to-Digital Converter

---

## 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

ADC1 Mode and Configuration

Mode

☐ IN11  
☐ IN12  
☐ IN13  
☐ IN14  
☐ IN15  
☒ Temperature Sensor Channel  
☐ Vrefint Channel

Categories: A->Z

System Core >  
Analog >  
ADC1  
ADC2  
ADC3  
DAC

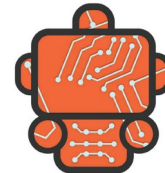
User Constants | NVIC Settings | DMA Settings  
Parameter Settings

Configure the below parameters :

Search (Ctrl+F)

ADCs Common Settings  
Mode Independent mode

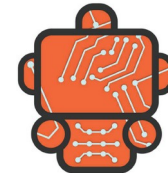
ADC Settings  
Clock Prescaler PCLK2 divided by 4  
Resolution 12 bits (15 ADC Clock cycles)  
Data Alignment Right alignment  
\* Scan Conversion Mode Disabled  
Continuous Conversion Mode Disabled  
Discontinuous Conversion ... Disabled  
DMA Continuous Requests Disabled  
End Of Conversion Selection EOC flag at the end of all conversions



# Analog-to-Digital Converter

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

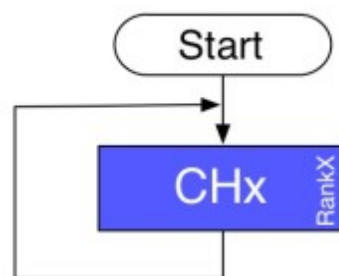
```
107  /* USER CODE BEGIN WHILE */
108  while (1) {
109      uint16_t adcValue = 1;
110      double temp;
111      if (HAL_ADC_Start(&hadc1) == HAL_OK) {
112          if (HAL_ADC_PollForConversion(&hadc1, 1000) == HAL_OK)
113              adcValue = HAL_ADC_GetValue(&hadc1);
114          HAL_ADC_Stop(&hadc1);
115      }
116      temp = (((double) adcValue * 3300 / 4095) - 760.0) / 2.5 + 25;
117      printf("adcValue: %hu\r\n", adcValue);
118      printf("Temperature: %0.2lf %cC\r\n", temp, 0xBA);
119      HAL_Delay(500);
120      /* USER CODE END WHILE */
```

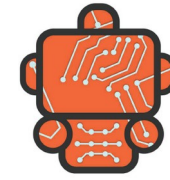


# Analog-to-Digital Converter

## 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.

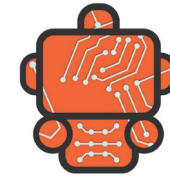




# Analog-to-Digital Converter

The screenshot shows the STM32CubeMX software interface for configuring the TIM6 timer. On the left, a list of timers is shown under the 'Timers' category, with TIM6 selected and highlighted by a red rectangle. The main panel on the right is titled 'TIM6 Mode and Configuration' and is divided into two sections: 'Mode' and 'Configuration'. In the 'Mode' section, the 'Activated' checkbox is checked and highlighted by a red rectangle, while 'One Pulse Mode' is unchecked. The 'Configuration' section includes a 'Reset Configuration' button and three tabs: 'User Constants', 'NVIC Settings', and 'DMA Settings', all of which are checked. Below these tabs is a 'Parameter Settings' section with a search bar and a list of parameters. The 'Counter Settings' section is expanded, showing 'Prescaler (PSC - 16 bits val...)' set to 431, 'Counter Mode' set to 'Up', 'Counter Period (AutoReload...)' set to 62499, and 'auto-reload preload' set to 'Disable'. The 'Trigger Output (TRGO) Parameters' section is also expanded, showing 'Trigger Event Selection' and 'Update Event', both of which are highlighted by a red rectangle.





# Analog-to-Digital Converter

The screenshot displays the STM32CubeMX configuration tool. On the left, the 'Categories' pane shows 'SYS' and 'Analog' sections. Under 'Analog', 'ADC1' is selected and highlighted with a red box. The main workspace shows the 'ADC1 Mode and Configuration' tab. The 'Mode' section has 'Temperature Sensor Channel' checked. The 'Configuration' section has a 'Reset Configuration' button. Below this, three tabs are visible: 'User Constants', 'NVIC Settings' (highlighted with a red box), and 'DMA Settings'. The 'NVIC Settings' tab shows the 'NVIC Interrupt Table' with 'ADC1, ADC2 and ADC3 global interrupts' checked and highlighted with a red box. On the right, the 'Parameter Settings' pane is open, showing various ADC configuration parameters. The 'ADCs\_Common\_Settings' section shows 'Mode' set to 'Independent mode'. The 'ADC\_Settings' section shows 'Clock Prescaler' as 'PCLK2 divided by 4', 'Resolution' as '12 bits (15 ADC Clock cycles)', and 'Data Alignment' as 'Right alignment'. The 'ADC\_Regular\_ConversionMode' section shows 'Number Of Conversion' as '1'. The 'External Trigger Conversion ...' parameter is highlighted with a red box and set to 'Timer 6 Trigger Out event'.

**ADC1 Mode and Configuration**

**Mode**

- ☐ IN15
- ☒ Temperature Sensor Channel
- ☐ Vrefint Channel

**Configuration**

Reset Configuration

**Parameter Settings**

Configure the below parameters :

Search (Ctrl+F)

**ADCs\_Common\_Settings**

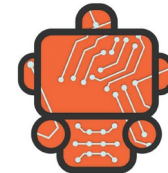
Mode: Independent mode

**ADC\_Settings**

- Clock Prescaler: PCLK2 divided by 4
- Resolution: 12 bits (15 ADC Clock cycles)
- Data Alignment: Right alignment
- \* Scan Conversion Mode: Disabled
- Continuous Conversion Mode: Disabled
- Discontinuous Conversion M...: Disabled
- DMA Continuous Requests: Disabled
- End Of Conversion Selection: EOC flag at the end of all conversions

**ADC\_Regular\_ConversionMode**

- Number Of Conversion: 1
- External Trigger Conversion ...: Timer 6 Trigger Out event
- \* External Trigger Conversion ...: Trigger detection on the rising edge
- Rank: 1

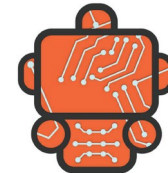


# Analog-to-Digital Converter

In adc.c:

```
23  /* USER CODE BEGIN 0 */
24  volatile uint32_t adcValue;
25  volatile uint8_t adcFlag;
26  /* USER CODE END 0 */

104 /* USER CODE BEGIN 1 */
105
106 void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc) {
107     if(hadc->Instance == ADC1){
108         adcValue = HAL_ADC_GetValue(&hadc1);
109         adcFlag = 1;
110     }
111 }
112
113 /* USER CODE END 1 */
```

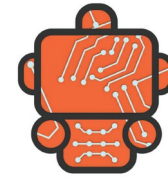


# Analog-to-Digital Converter

---

In adc.h:

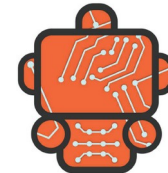
```
35  /* USER CODE BEGIN Private defines */
36  extern volatile uint32_t adcValue;
37  extern volatile uint8_t adcFlag;
38  /* USER CODE END Private defines */
```



# Analog-to-Digital Converter

In main.c:

```
103 HAL_TIM_Base_Start_IT(&htim6);
104 HAL_ADC_Start_IT(&hadc1);
105 /* USER CODE END 2 */
106
107 /* Infinite loop */
108 /* USER CODE BEGIN WHILE */
109 while (1) {
110     double temp;
111     if (adcFlag) {
112         adcFlag = 0;
113         temp = (((double) adcValue * 3300 / 4095) - 760.0) / 2.5 + 25;
114         printf("adcValue: %u\r\n", adcValue);
115         printf("Temperature: %0.2lf %cC\r\n", temp, 0xBA);
116     }
117     /* USER CODE END WHILE */
```

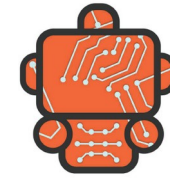


# Digital-to-Analog Converter

---

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.

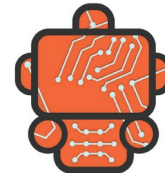
STM32F767ZI microcontrollers provide only a single dual channel DAC. The documentation is in RM0410 at page 486. The API for DAC is at page 198 in UM1905.



# Digital-to-Analog Converter

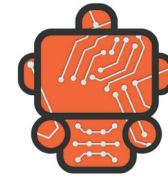
Example 5: Put the value obtained by the ADC into an analog output.

The screenshot shows the STM32CubeMX software interface for configuring the DAC. On the left, the 'Categories' pane shows 'System Core', 'Analog', 'Timers', 'Connectivity', and 'Multimedia'. Under 'Analog', 'ADC1', 'ADC2', 'ADC3', and 'DAC' are listed. 'DAC' is selected and highlighted in blue. On the right, the 'DAC Mode and Configuration' window is open. It has two main sections: 'Mode' and 'Configuration'. In the 'Mode' section, 'OUT1 Configuration' is checked, while 'OUT2 Configuration' and 'External Trigger' are unchecked. In the 'Configuration' section, there is a 'Reset Configuration' button and five tabs: 'NVIC Settings', 'DMA Settings', 'GPIO Settings', 'Parameter Settings' (which is selected), and 'User Constants'. Below the tabs, a text box says 'Configure the below parameters :'. Under this, 'DAC Out1 Settings' is expanded, showing 'Output Buffer' set to 'Disable' and 'Trigger' set to 'None'. A search bar at the top left of the configuration pane contains the text 'Search (Ctrl+F)'.



# Digital-to-Analog Converter

```
110  /* USER CODE BEGIN WHILE */
111  while (1) {
112      double temp;
113      if (adcFlag) {
114          adcFlag = 0;
115          temp = (((double) adcValue * 3300 / 4095) - 760.0) / 2.5 + 25;
116          printf("adcValue: %u\r\n", adcValue);
117          printf("Temperature: %0.2lf %cC\r\n", temp, 0xBA);
118          if (HAL_DAC_GetState(&hdac) != HAL_DAC_STATE_READY) {
119              HAL_DAC_Stop(&hdac, DAC1_CHANNEL_1);
120          }
121          if (HAL_DAC_Start(&hdac, DAC1_CHANNEL_1) == HAL_OK) {
122              HAL_DAC_SetValue(&hdac, DAC1_CHANNEL_1, DAC_ALIGN_12B_R, adcValue);
123          }
124      }
125      /* 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>