Embedded programing with STM Cube IDE

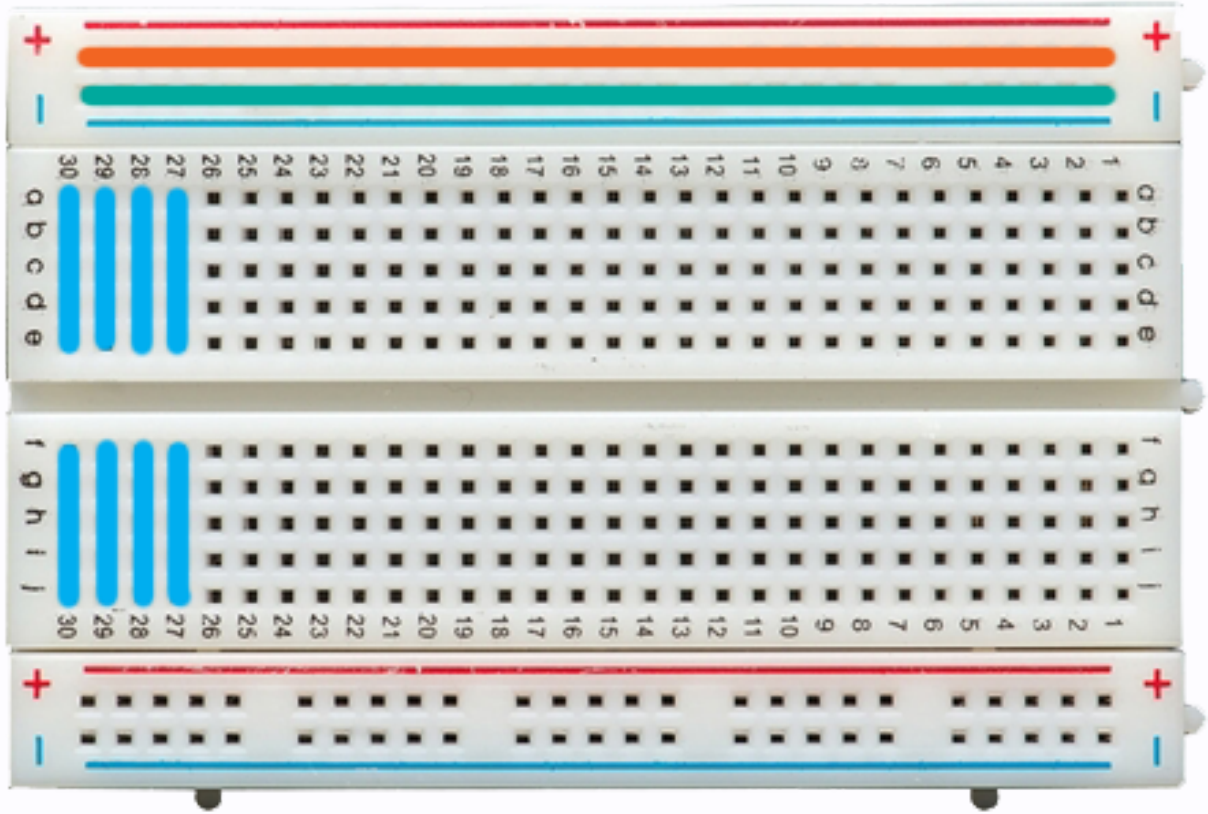
Ignas Brazauskas

What are microcontrollers?

- *A microcontroller (MCU for microcontroller unit, also MC, UC, or µC) is a small computer on a single VLSI integrated circuit (IC) chip.*
- Its a small computer for controlling devices!

Lets set it up

- Plop the MCU!



Initial setup

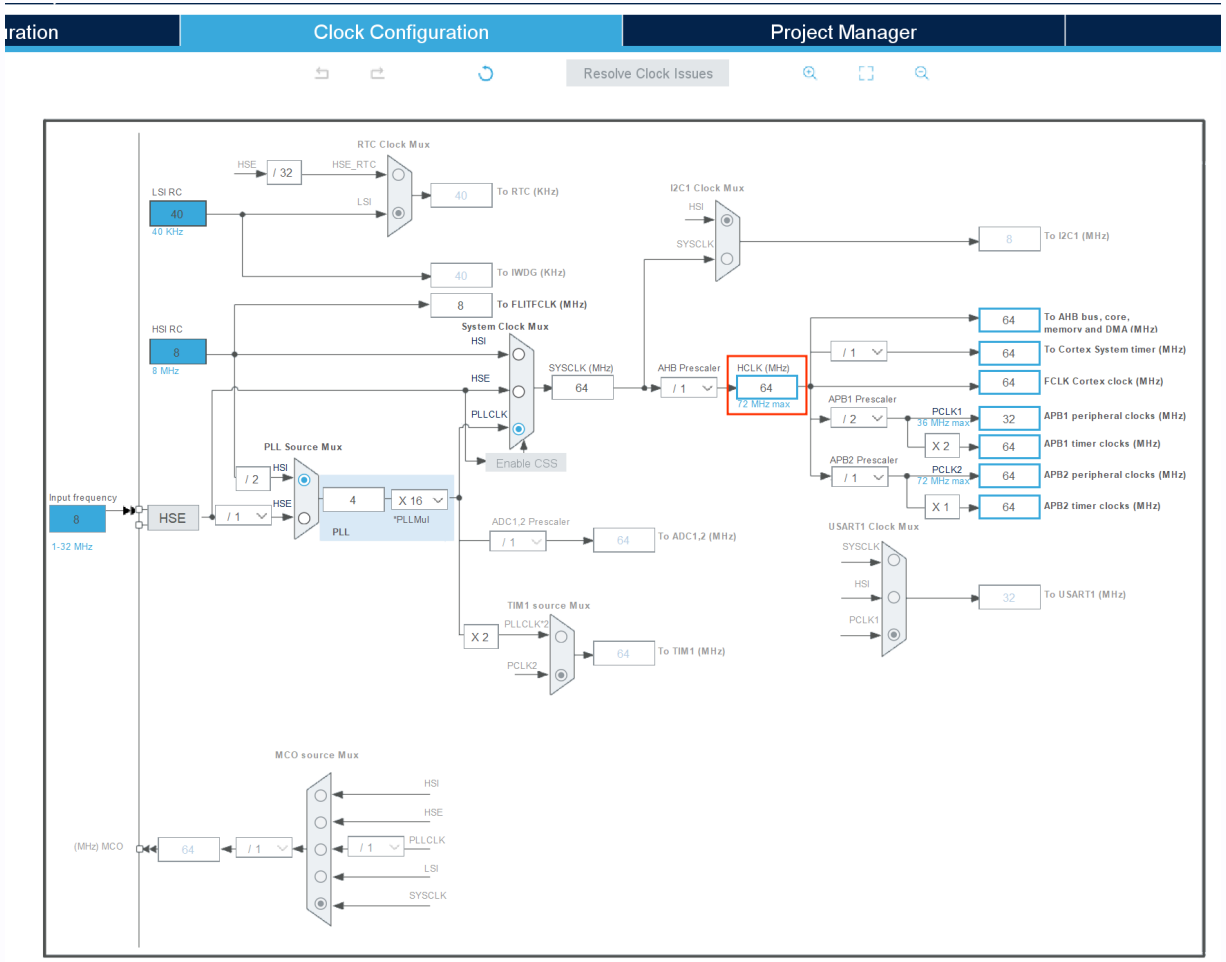
- In CubeIDE create a **new project** for **NUCLEO-F303K8**
- Recomendation:
  - Project Manager -> *Copy only*
  - Code generation -> *Generate peripheral as a pair of* `.c/.h` *files per peripheral*

Clocks

- Used as a trigger for logic gates
- *Keeps the circuits running in a rythm!*

We will adjust the clock frequency

- In **Clock configuration -> HCLK**
- We will use **64MHz**



It worked!

But its not doing anything?

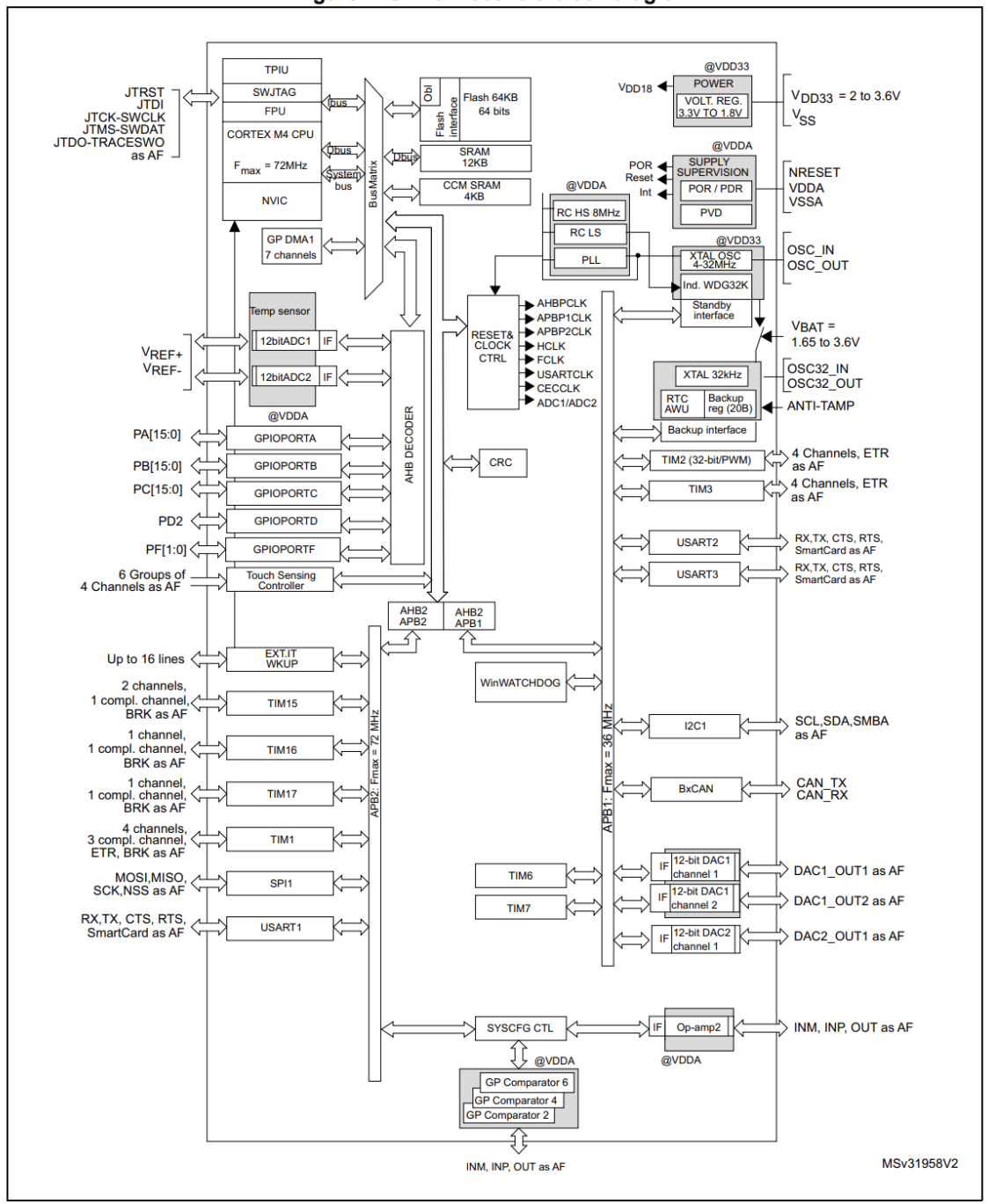
Lets make it do something!

The classic blink!

Microcontroller hardware

- STM32F303K8 dataheet

Figure 1. STM32F303x6/8 block diagram

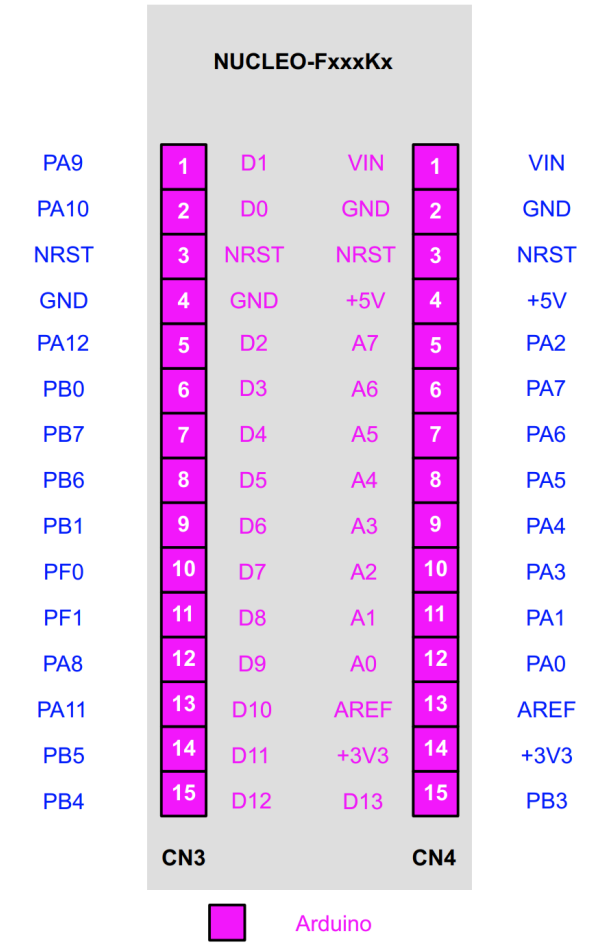


1. AF: alternate function on I/O pins.

Nucleo board pinout

- Nucleo-F303K8 manual
  - *Page 30*

Figure 7. NUCLEO-F031K6, NUCLEO-F042K6, NUCLEO-F303K8, and NUCLEO-F301K8 pin assignment



Nucleo board

User LED

- Nucleo-F303K8 manual
  - *Page 18*

6.5 LEDs

The tricolor LED (green, orange, red) LD1 (COM) provides information about ST-LINK communication status. LD1 default color is red. LD1 turns to green to indicate that the communication is in progress between the PC and the ST-LINK/V2-1, with the following setup:

- Slow blinking red/off: at power-on before USB initialization
- Fast blinking red/off: after the first correct communication between PC and ST-LINK/V2-1 (enumeration)
- Red on: when initialization between PC and ST-LINK/V2-1 is completed
- Green on: after a successful target communication initialization
- Blinking red/green: during communication with target
- Green on: communication finished and successful
- Orange on: communication failure

User LD3: the green LED is a user LED connected to Arduino Nano signal D13 corresponding to the STM32 I/O PB3 (pin 26). Refer to *Table 9*, *Table 10*, *Table 12*, *Table 13*, *Table 14*, *Table 15* and *Table 16* for concerned STM32:

- When the I/O is HIGH value, the LED is on
- When the I/O is LOW, the LED is off

PWR LD2: the red LED indicates that the STM32 part is powered and +5 V power is available.

Lets setup PB3 as an Output

How to toggle this pin?

```
HAL_GPIO_WritePin(LD3_GPIO_Port, LD3_Pin, 1);
```

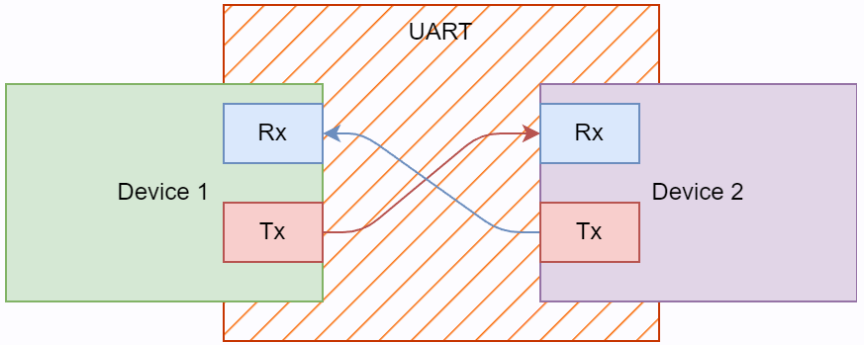
Lets get some information to the terminal!

*We will need to set up **UART**!*

Comunications:

UART

- *Universal Asynchronous Reciever Transmiter*
- Transfer data between **two devices**
- Must have the same configuration on two devices:
  - **Bitrate**
  - **Parity**
  - **Word lenght**



Virtual U(S)ART

- A **UART connection** provided by the **debug probe**
- Allows us to **send data to a computer!**

6.9 USART virtual communication

Thanks to SB2 and SB3, the USART interface of STM32 available on PA2 (TX) and PA15 (RX), can be connected to ST-LINK/V2-1. When USART is not used it is possible to use PA2 as Arduino Nano A7. Refer to *Table 7*.

Table 7. Virtual communication configuration

Bridge	State(1)	Description
SB2	OFF	PA2 is connected to CN4 pin 5 as Arduino Nano analog input A7 and disconnected from ST-LINK USART.
	ON	PA2 is connected to ST-LINK as virtual Com TX (default).
SB3	OFF	PA15 is not connected.
	ON	PA15 is connected to ST-LINK as virtual Com RX (default).

1. The default configuration is reported in bold style.

STM USART setup

- Bit rate: **115200**
- Word lenght: **8 Bits**
- Parity: **None**
- Stop bits: **1**

PuTTY setup

- Connection type: **Serial** and **Raw**
- Serial line: **COM4**
  - *Most likely, otherwise can be found in **Device manager***
- Speed: **115200**
- Terminal -> **Implicit CR in every LF**

Terminal-s setup

- Run it! `terminal-s`

Lets try to send something!

```
char helloText[] = "Hello!\n";
HAL_UART_Transmit(&huart2, helloText, sizeof(helloText), 1000);
```

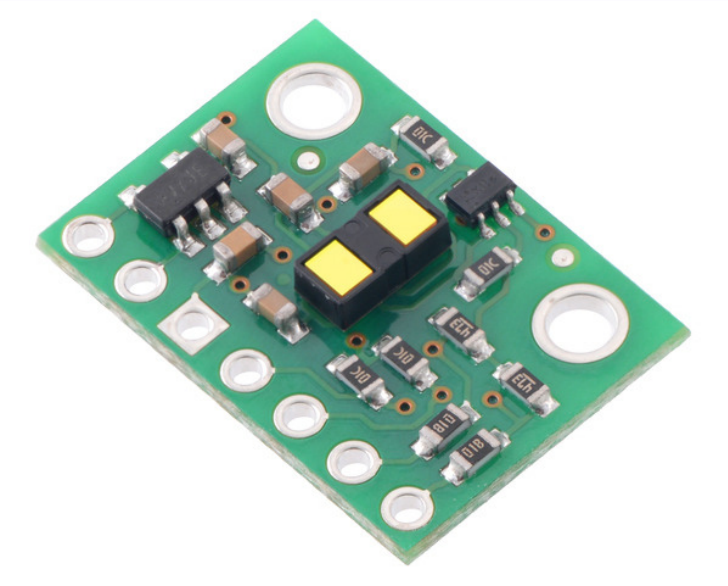
We can use this for status and debugging information!

But now lets use some hardware!

- *Get that distance!*

VL53L1X I2C TOF sensor

- 400cm sensing distance
- 3.3V operating voltage

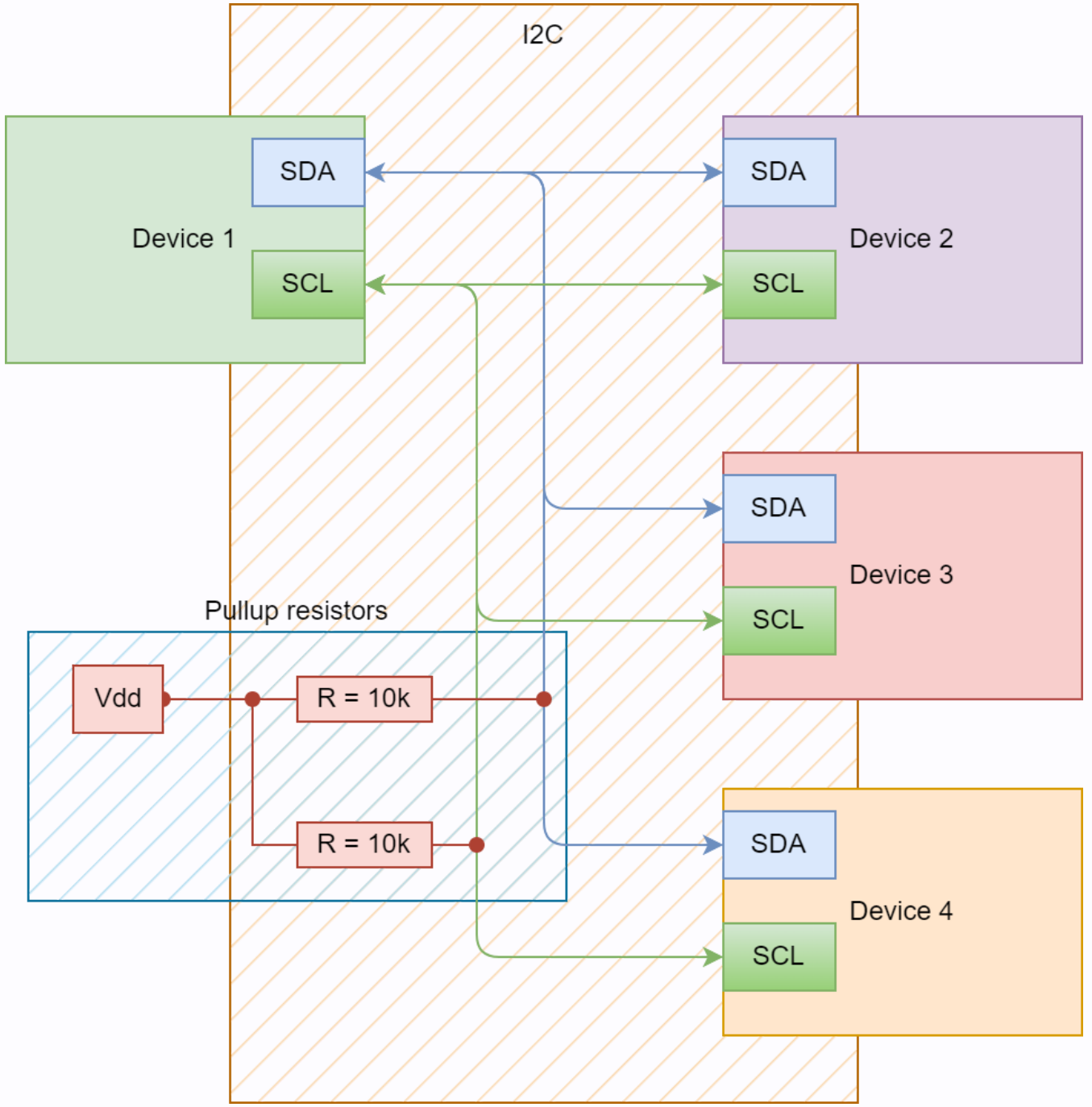


Comunications: I2C

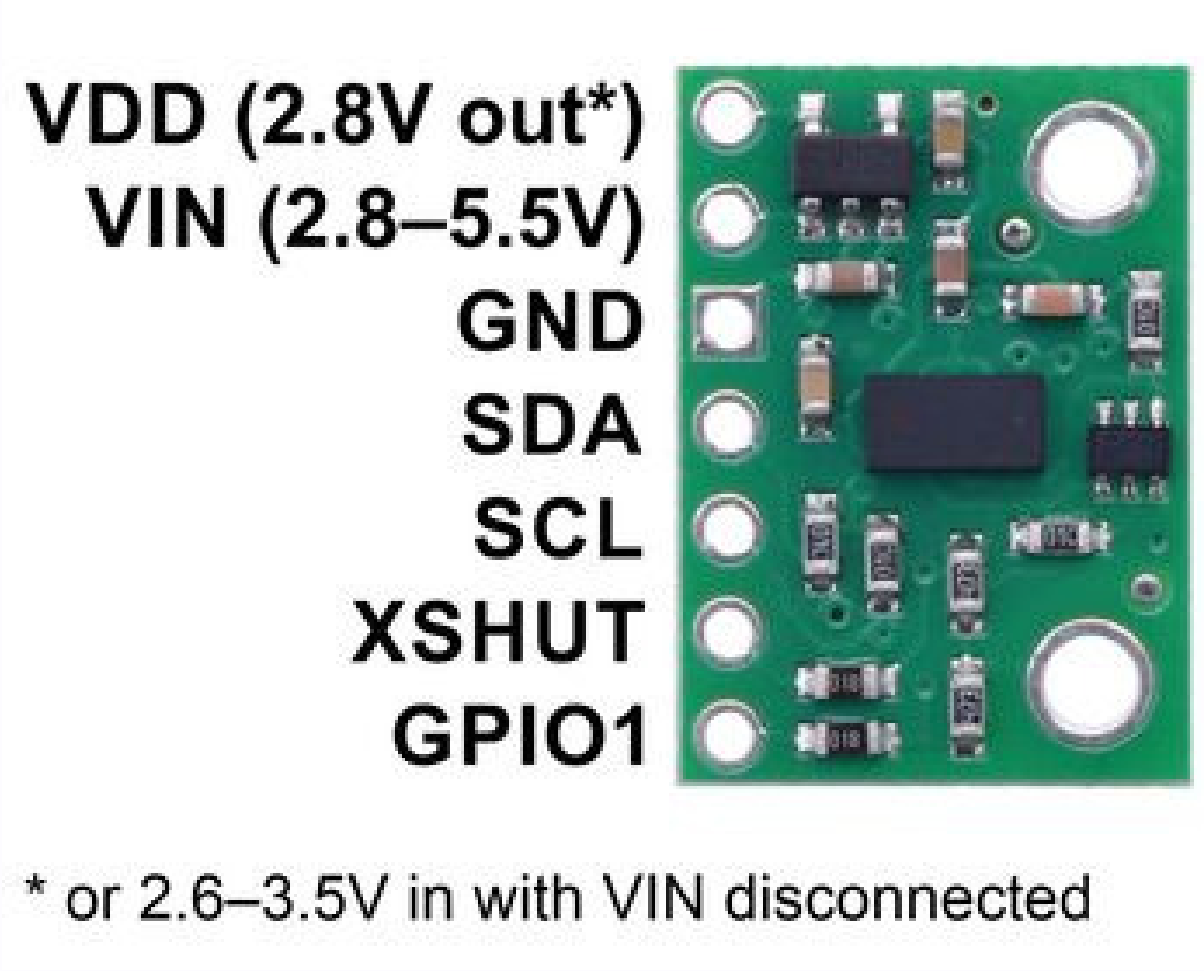
- *Inter-Integrated Circuit*
- Transfer data between **multiple devices**
- Must have the same configuration:
  - **Speed mode**: ***Standart***, ***Fast***, *Fast Plus*
  - **Address lenght**: **7bit**, 10bit
- *Double check the **device address**!*

I2C

conection scheme



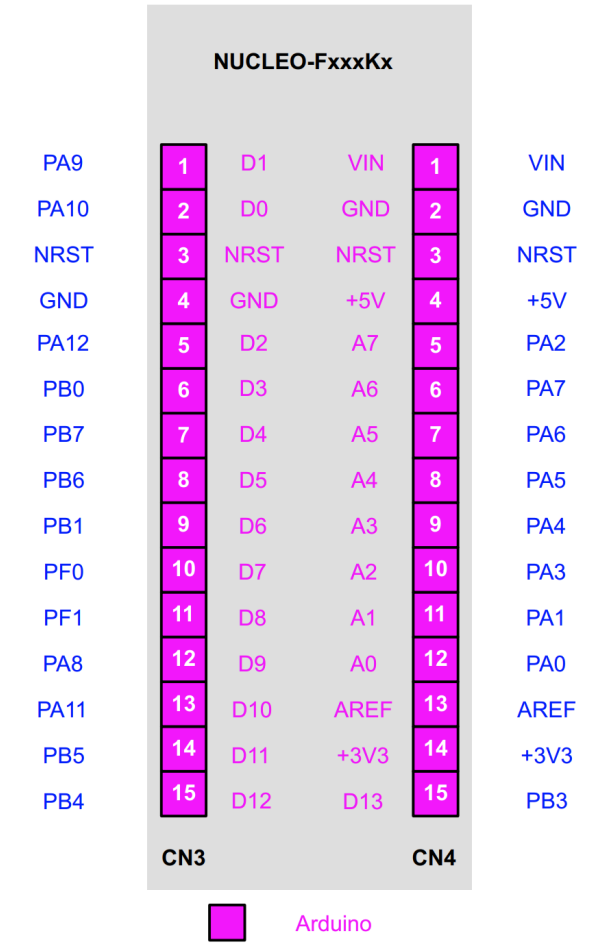
Connecting the VL53L1X



Nucleo board pinout

- Nucleo-F303K8 manual
  - *Page 30*

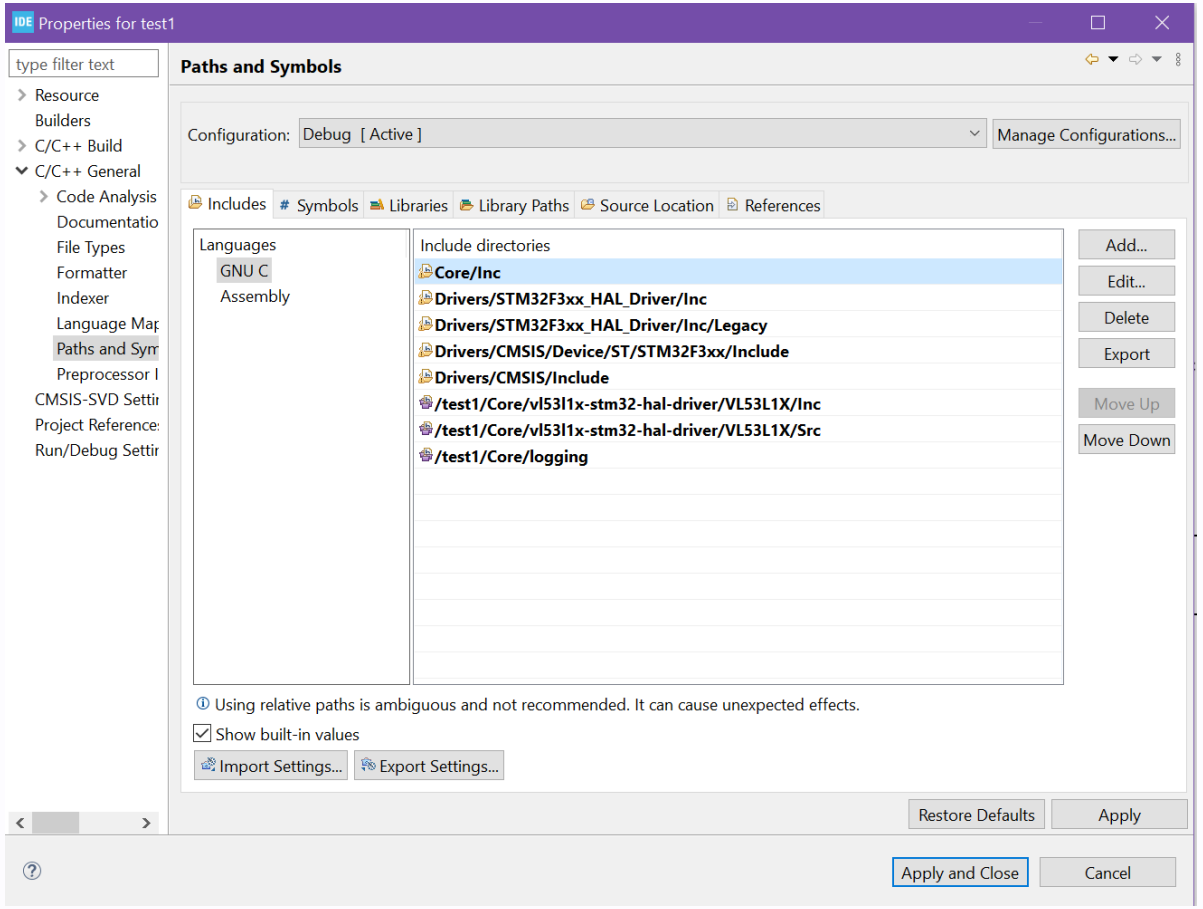
Figure 7. NUCLEO-F031K6, NUCLEO-F042K6, NUCLEO-F303K8, and NUCLEO-F301K8 pin assignment



Importing the VL53L1X library

- C/C++ General -> Paths and Symbols
  - *GNU C*: **Add**

```
/* USER CODE BEGIN Includes */
#include "VL53L1X.h"
```



Using the library

Outside `while(1)`

```
VL53L1X sensor0;
VL53L1X sensor1;

TOF_InitStruct(&sensor0, &hi2c1, 0x32, VL53L0X0_XHUT_GPIO_Port, VL53L0X0_XHUT_Pin);
TOF_InitStruct(&sensor1, &hi2c1, 0x34, VL53L0X1_XHUT_GPIO_Port, VL53L0X1_XHUT_Pin);

VL53L1X* sensors[] = {&sensor0, &sensor1};
TOF_BootMultipleSensors(sensors, 2);

uint16_t sensor0_Data = 0;
uint16_t sensor1_Data = 0;

/* ------------------------------------------------------------------------- */
// Inside while(1)
sensor0_Data = TOF_GetDistance(&sensor0);
HAL_Delay(5);
sensor1_Data = TOF_GetDistance(&sensor1);
```

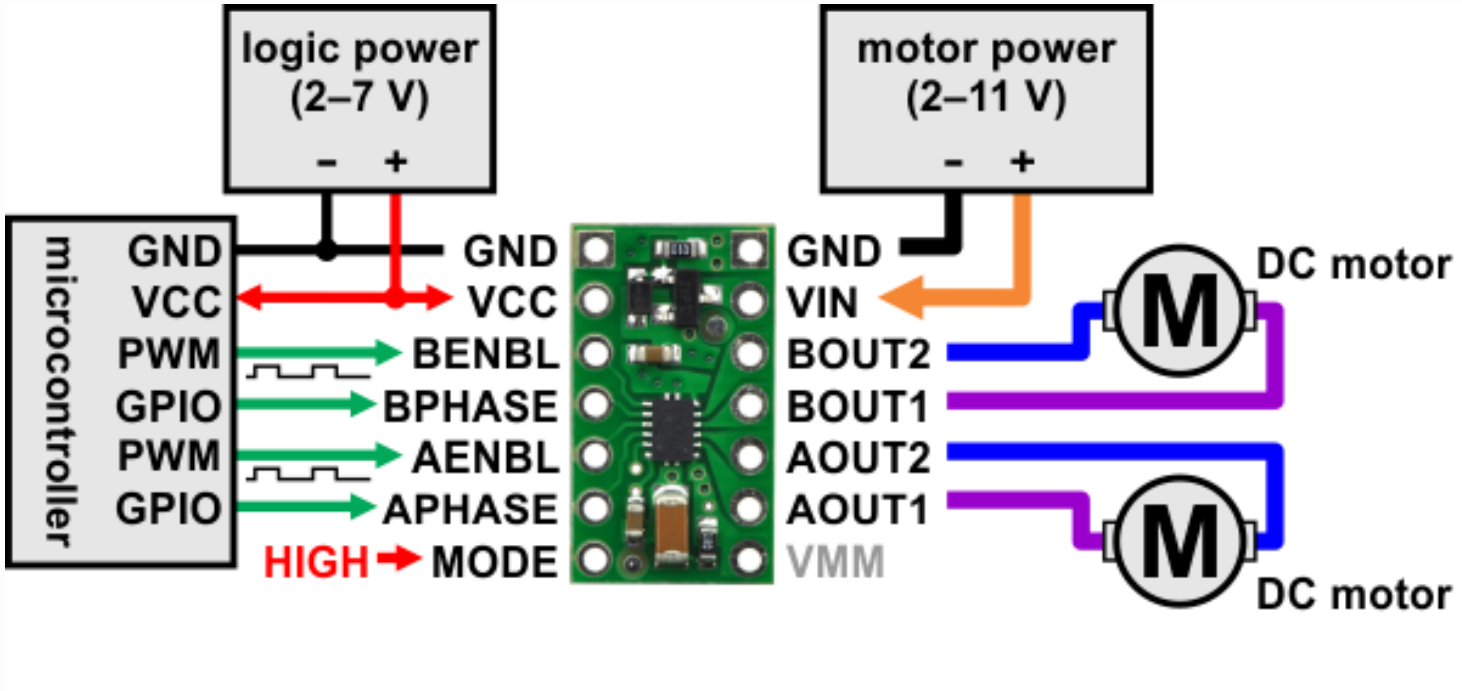
Controlling a motor!

We will use an H-bridge driver

DRV8835

H-bridge driver

- Allows to controll a **high power load**!



Generating PWM

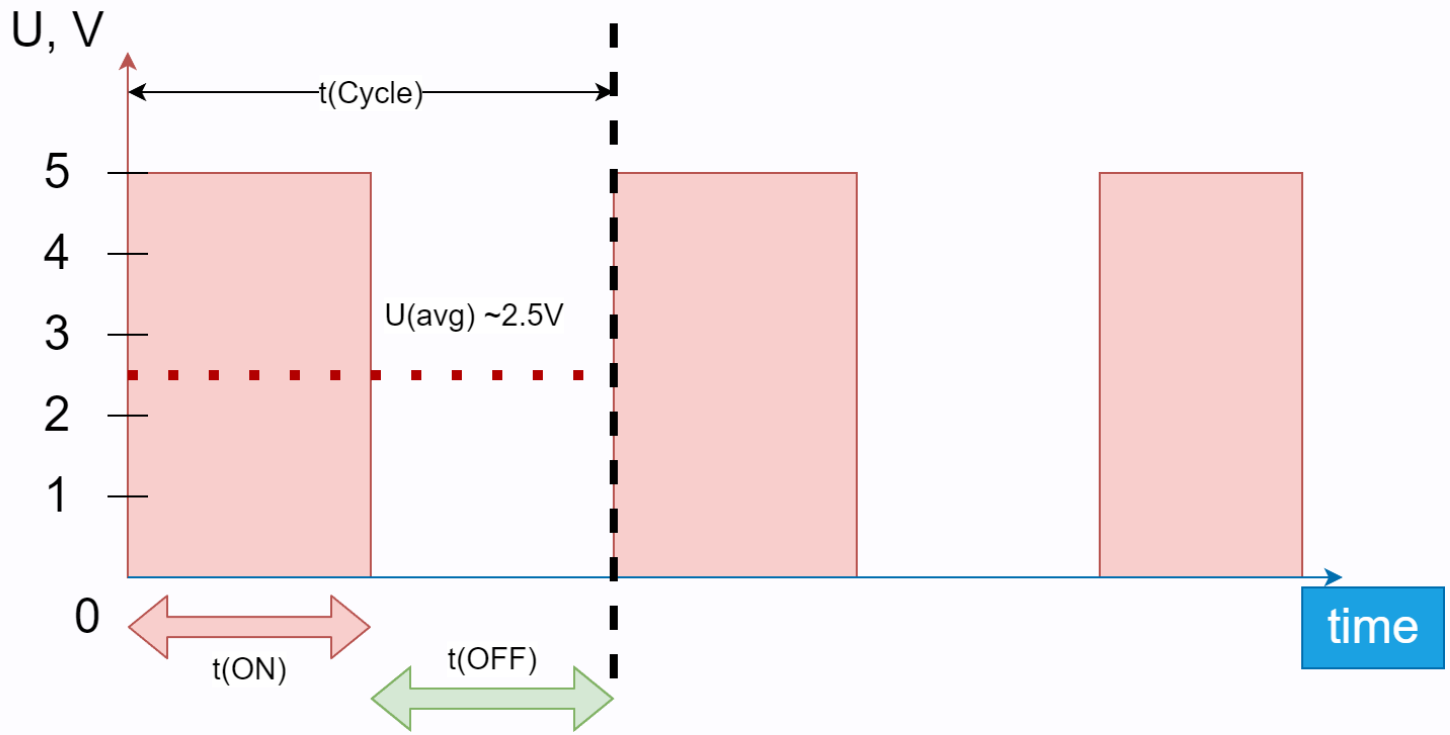
A what now?

PWM: Intro

- Pulse Width Modulation allows us to generate a **lower *average* voltage**
  - $U_{avg} = D * U_{max}$
- This is done by **quickly switching the power** for a **certain period of time**
- $D, \% = \frac{t_{ON}}{t_{ON} + t_{OFF}}$

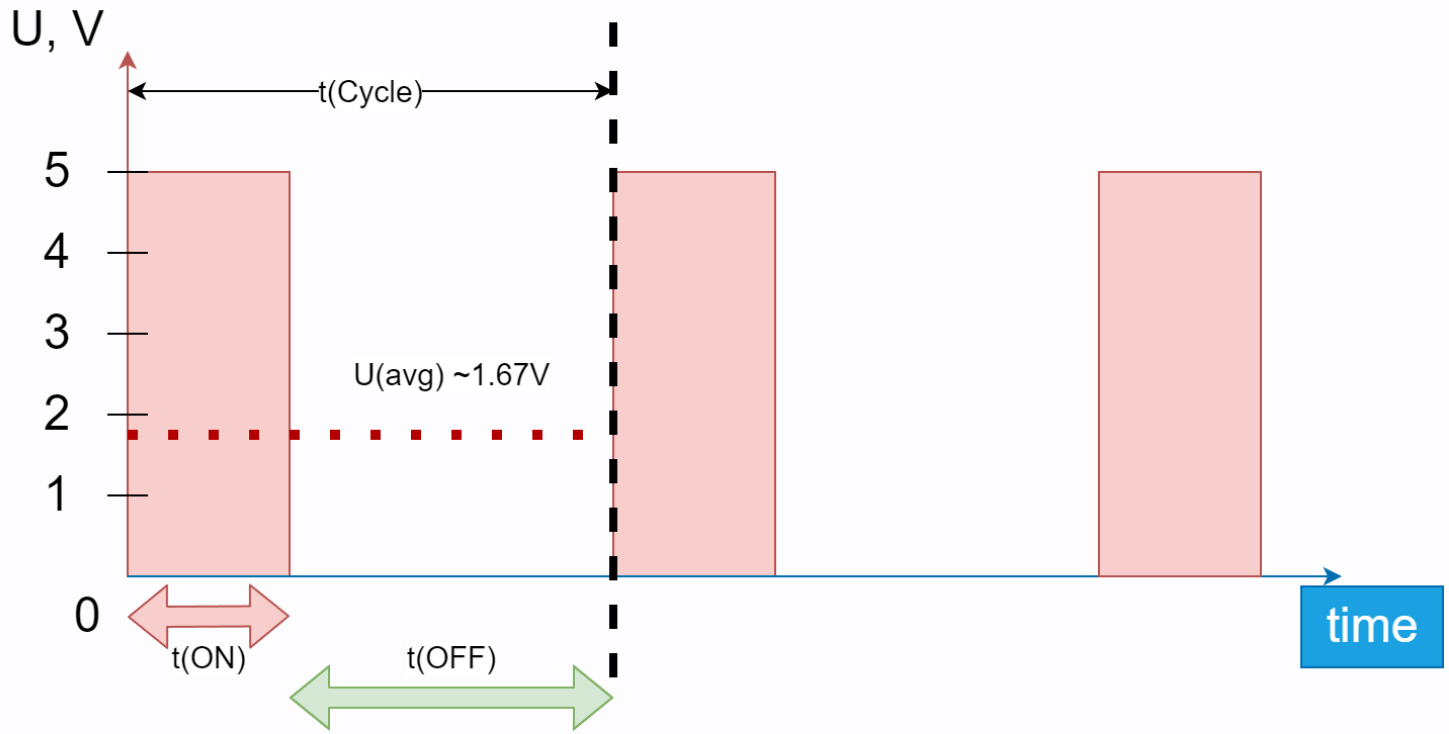
PWM Example

50% Duty



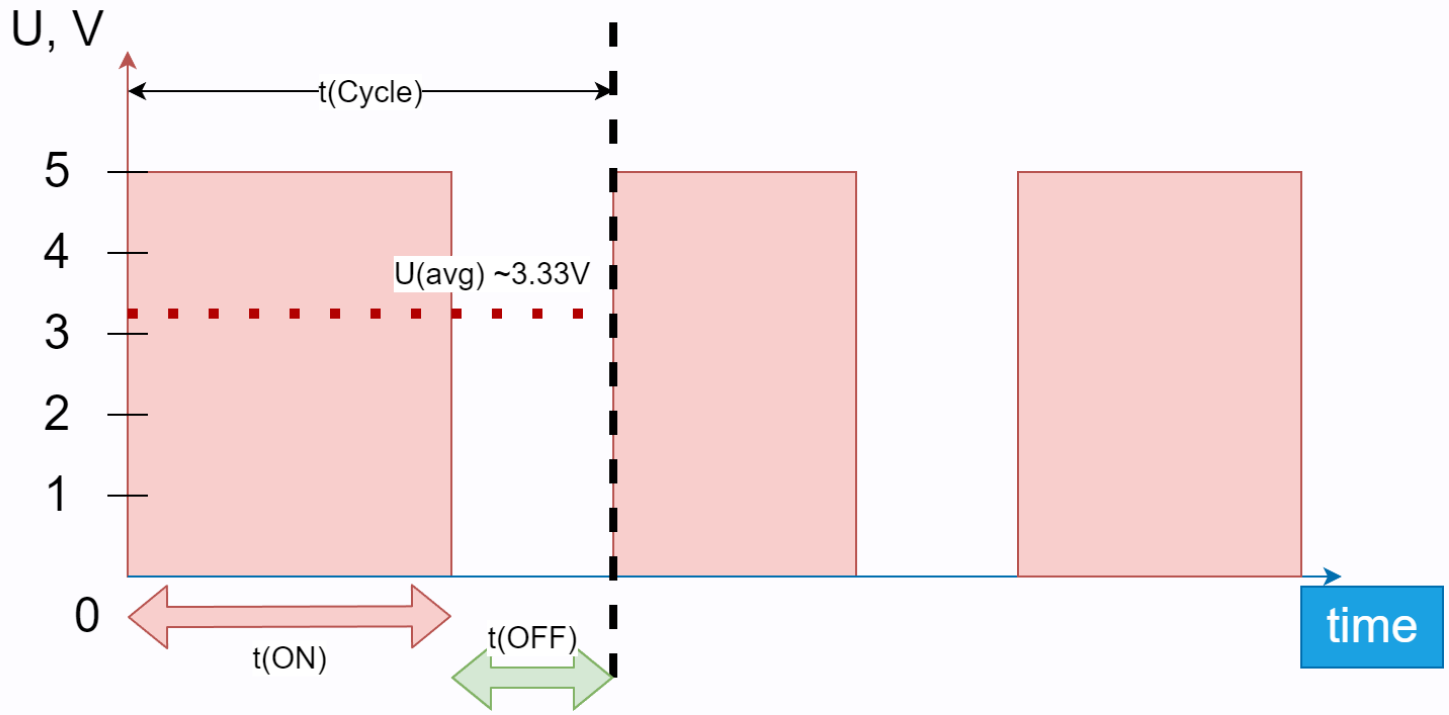
PWM Example

33% Duty



PWM Example

66% Duty



PWM parameters

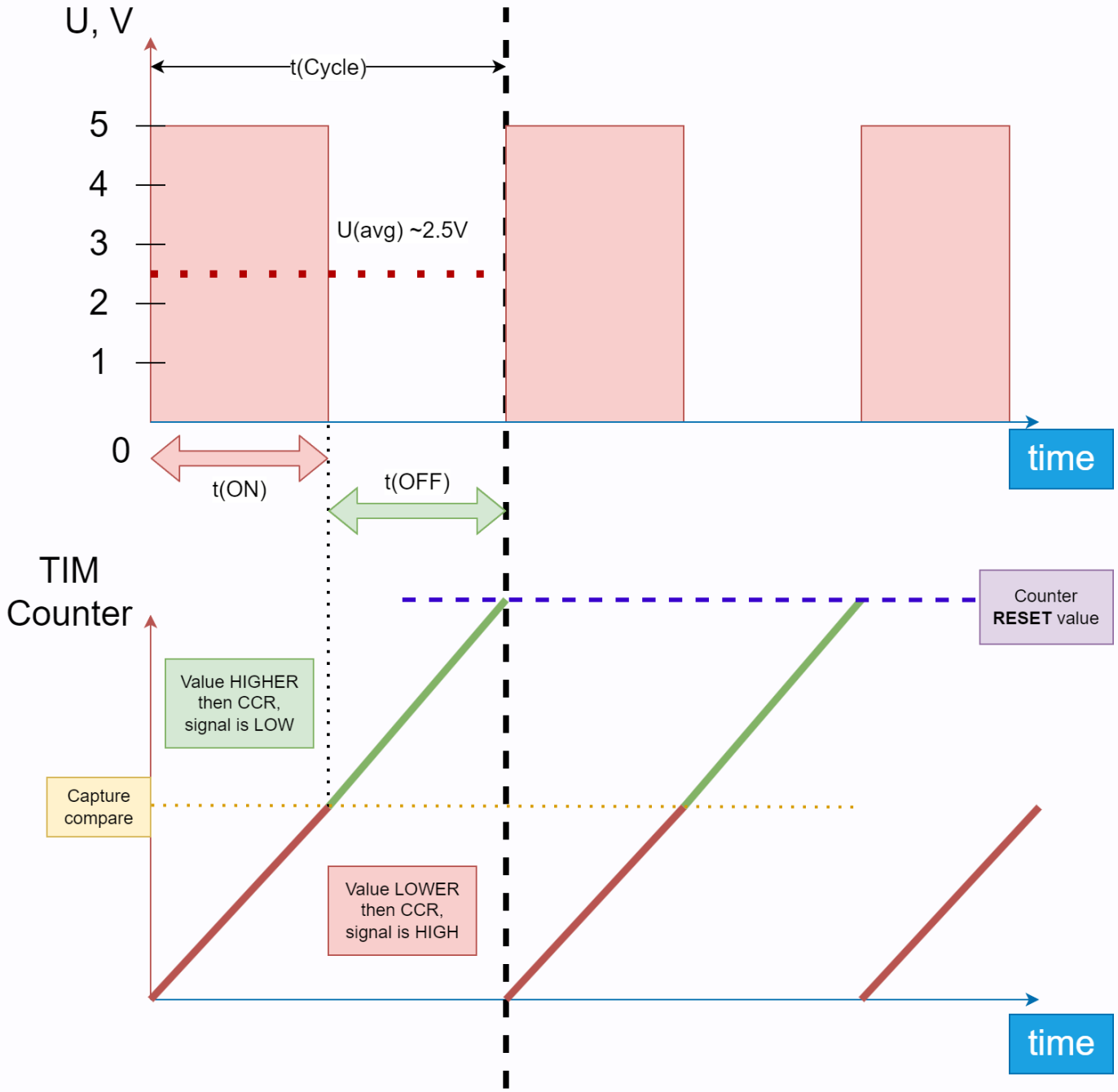
- f - frequency, Hz
- D - Duty cycle, %

How is PWM generated?

- *It all comes down to the timer!*

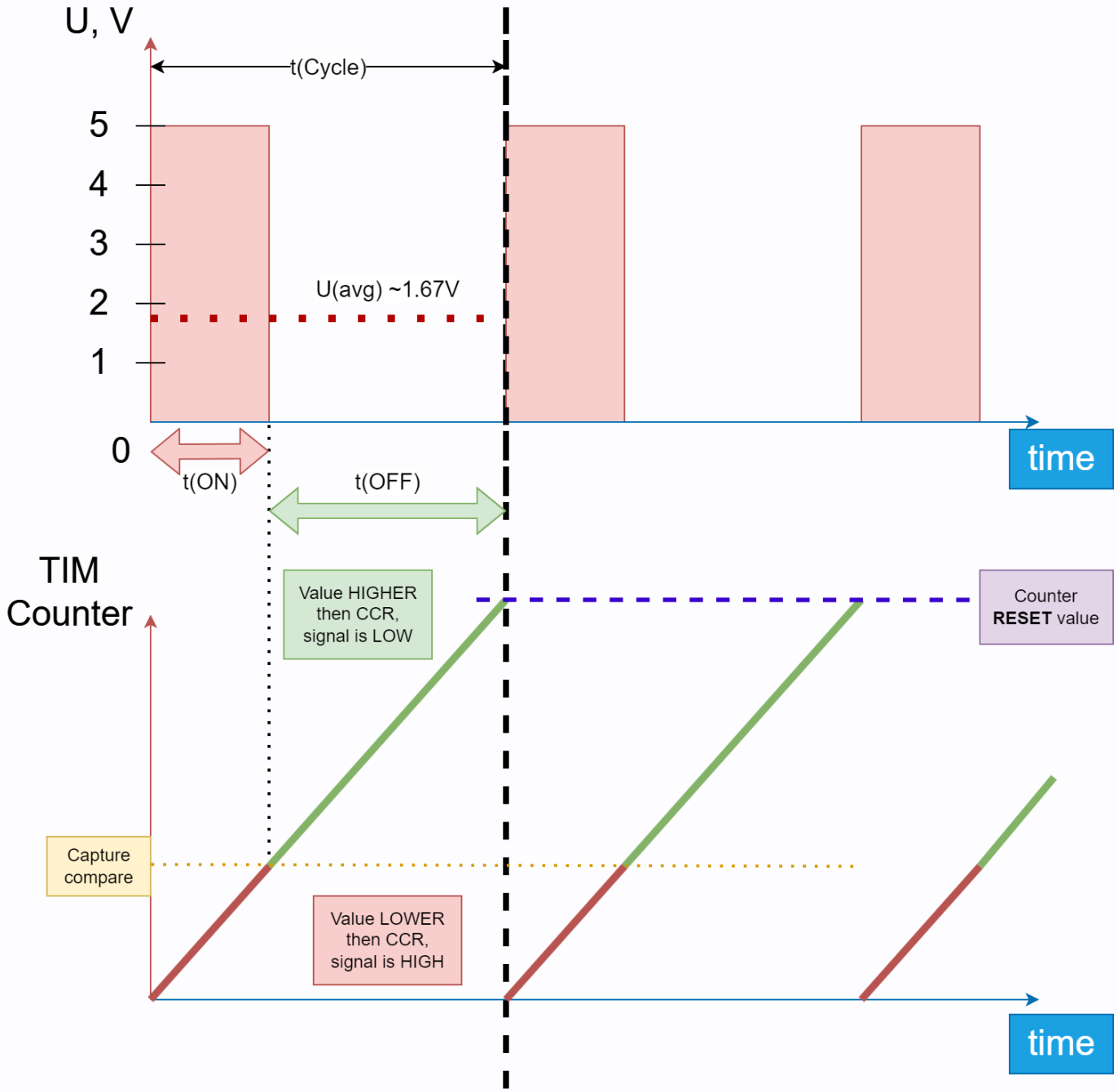
PWM Example

50% Duty



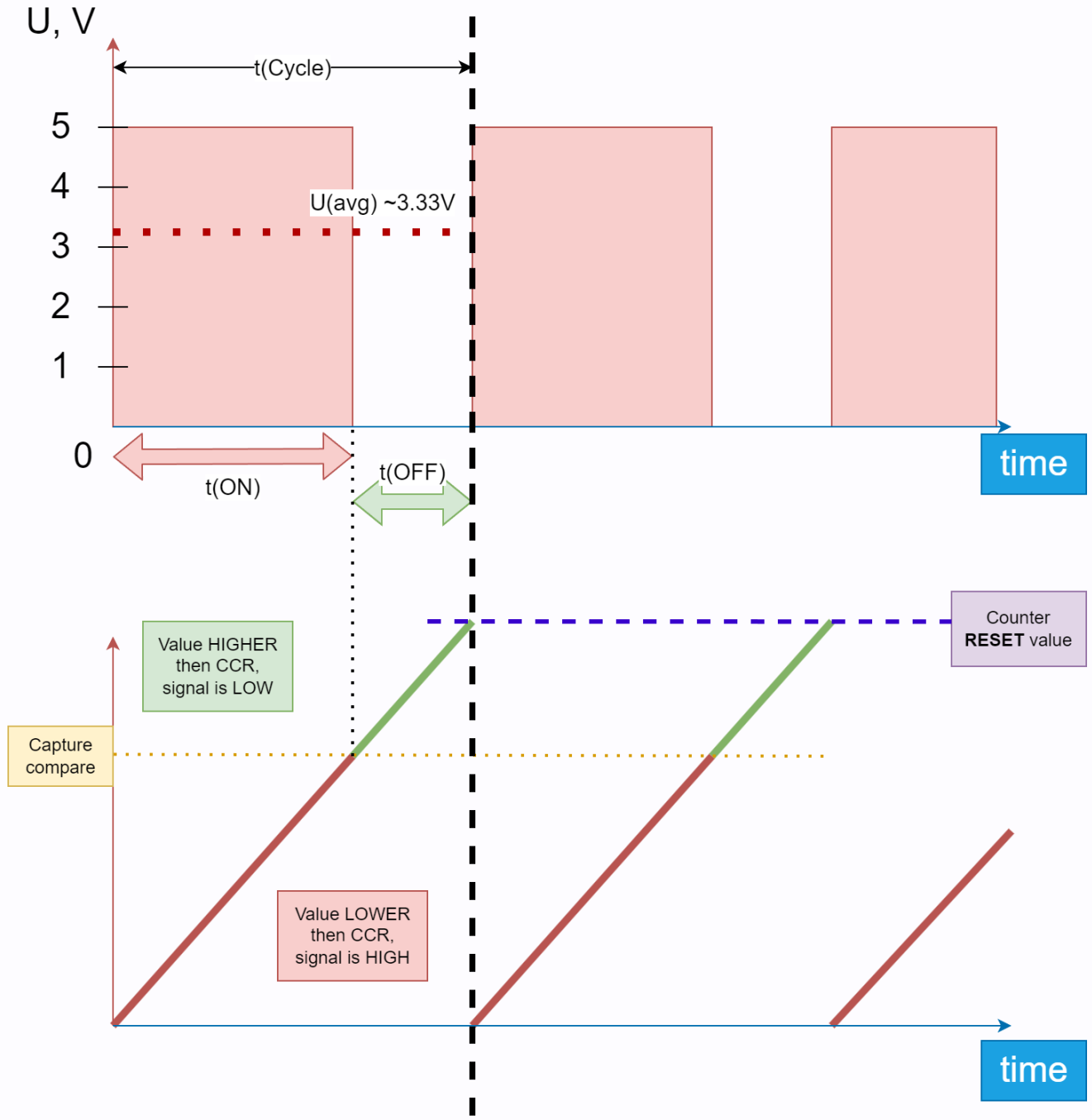
PWM Example

33% Duty



PWM Example

66% Duty



What frequency to use?

- Idealy it should be in a **motors datasheet**, but quick seaching online I found this:
  - Is there an ideal PWM frequency for DC brush motors?

```
Stay in the 5-20 kHz range and you probably will be safe.
If you go too much lower, the motor current ripple (and torque ripple)
may be noticeable, but you can experiment with this.
Too much higher and you will be heating up your switches.
You may also want to go towards the higher end to get out of the audible range.
```

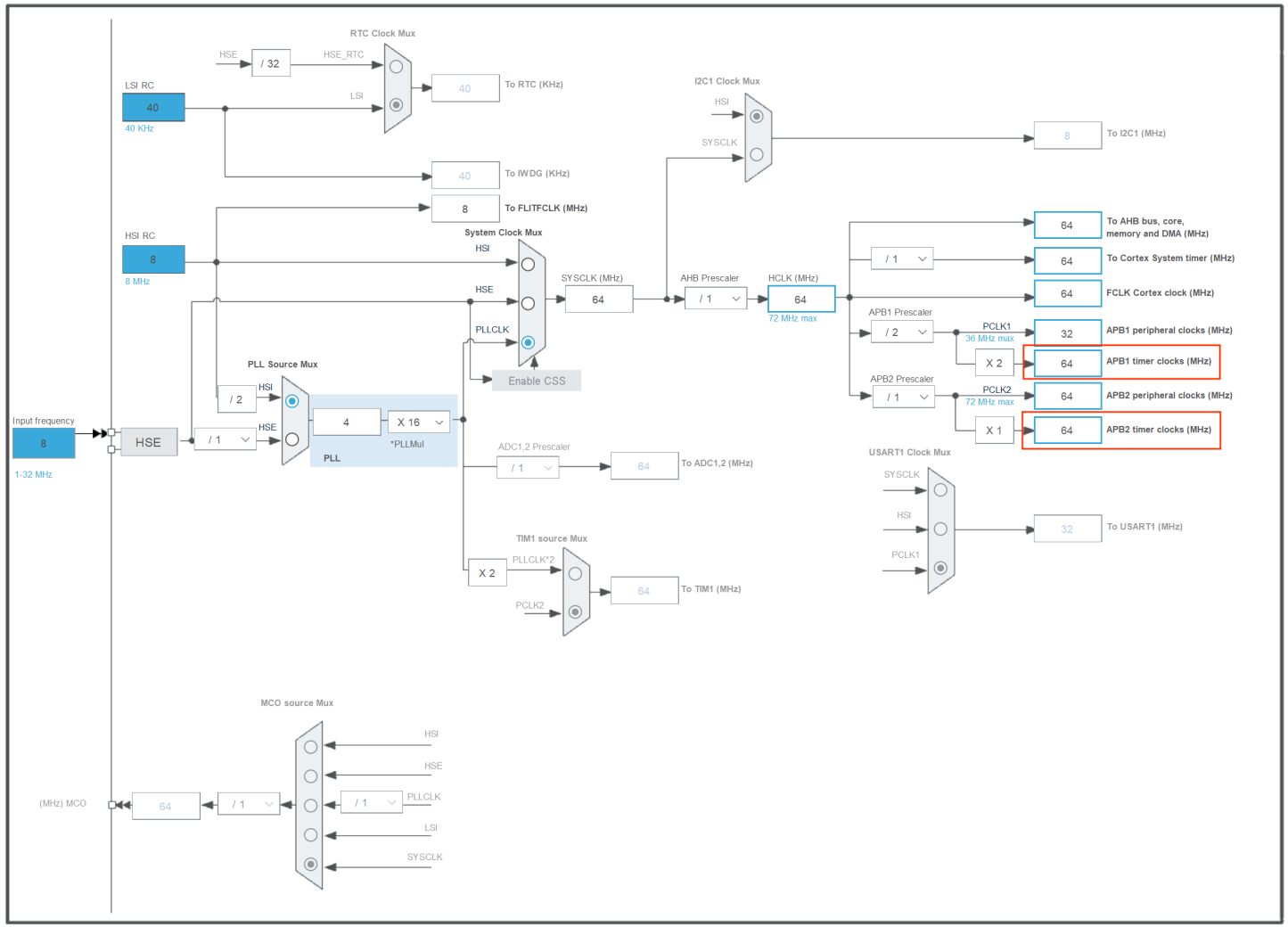
- So lets use **10kHz**

How to find the counter value?

- $N_{counter} = \frac{f_{TIM\ clock}}{f_{wanted}}$
  - $f_{wanted} = 10kHz$
  - $f_{TIM\ clock}$?
- We will use **TIM2**

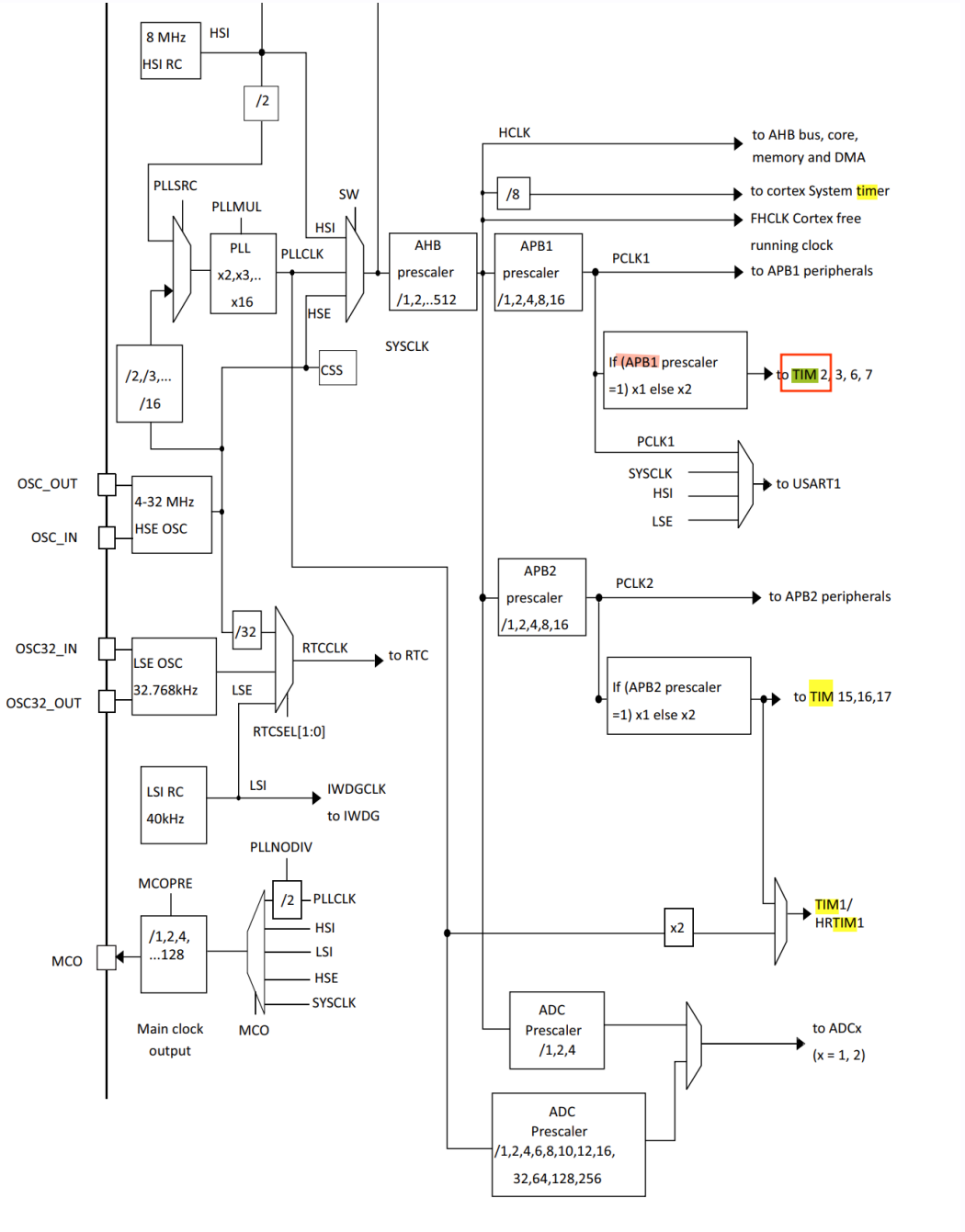
What do we need?

- **Timer clock frequency**
- *We can see this in the* ***Clock configurator***



APB1 or APB2?

- STM32F303K8 dataheet
  - *Page 19*
- We want to use **TIM2**, and its connected to **APB1**



How to find the counter value?

- $N_{counter} = \frac{f_{TIM\ clock}}{f_{wanted}}$
- In our case, we want 10KHz:
  - $N_{counter} = \frac{64MHz}{10kHz} = 6400$

PWM CubeIDE setup

- We will use **TIM2**
  - Clock source: **Internal Clock**
  - Channel1: **PWM Generation CH1**
- Parameter settings:
  - Counter period: **6400 -1** *we calculated this*
  - Auto-reload preload: **Enable**
- *We can also change the output pin name!*

Lets generate some PWM!

```
TIM2->CCR1 = 0;
HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
```

How to change the duty cycle?

- $CCR1 = N_{counter} * D$
- Example, for $D = 50\%$,
  - $CCR1 = 6400 * 0.5 = 3200$

```
TIM2->CCR1 = 3200; // 50%
```

Using STM HAL

The Hardware Abstraction Layer

STM HAL

- Information can be found in STM's website
- Or in the header files:
  Drivers/STM32xxxx_HAL_Driver/Inc
  - Uart driver example:

```
/** @addtogroup UART_Exported_Functions_Group2 IO operation functions
  * @{
  */

/* IO operation functions *****************************************************/
HAL_StatusTypeDef HAL_UART_Transmit(UART_HandleTypeDef *huart, const uint8_t *pData, uint16_t Size, uint32_t Timeout);
HAL_StatusTypeDef HAL_UART_Receive(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size, uint32_t Timeout);
HAL_StatusTypeDef HAL_UART_Transmit_IT(UART_HandleTypeDef *huart, const uint8_t *pData, uint16_t Size);
HAL_StatusTypeDef HAL_UART_Receive_IT(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size);
HAL_StatusTypeDef HAL_UART_Transmit_DMA(UART_HandleTypeDef *huart, const uint8_t *pData, uint16_t Size);
HAL_StatusTypeDef HAL_UART_Receive_DMA(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size);
HAL_StatusTypeDef HAL_UART_DMAPause(UART_HandleTypeDef *huart);
HAL_StatusTypeDef HAL_UART_DMAResume(UART_HandleTypeDef *huart);
HAL_StatusTypeDef HAL_UART_DMAStop(UART_HandleTypeDef *huart);
```

HAL UART example

```
/**
  * @brief Send an amount of data in blocking mode.
  * @note   When UART parity is not enabled (PCE = 0), and Word Length is configured to 9 bits (M1-M0 = 01),
  *         the sent data is handled as a set of u16. In this case, Size must indicate the number
  *         of u16 provided through pData.
  * @param huart   UART handle.
  * @param pData   Pointer to data buffer (u8 or u16 data elements).
  * @param Size    Amount of data elements (u8 or u16) to be sent.
  * @param Timeout Timeout duration.
  * @retval HAL status
  */
HAL_StatusTypeDef HAL_UART_Transmit(UART_HandleTypeDef *huart, const uint8_t *pData, uint16_t Size, uint32_t Timeout)
{
  const uint8_t  *pdata8bits;
  const uint16_t *pdata16bits;
  uint32_t tickstart;
  // etc...
```

HAL I2C example

```
/**
  * @brief  Transmits in master mode an amount of data in blocking mode.
  * @param  hi2c Pointer to a I2C_HandleTypeDef structure that contains
  *               the configuration information for the specified I2C.
  * @param  DevAddress Target device address: The device 7 bits address value
  *         in datasheet must be shifted to the left before calling the interface
  * @param  pData Pointer to data buffer
  * @param  Size Amount of data to be sent
  * @param  Timeout Timeout duration
  * @retval HAL status
  */
HAL_StatusTypeDef HAL_I2C_Master_Transmit(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint8_t *pData,
                                          uint16_t Size, uint32_t Timeout)
{
  uint32_t tickstart;

  if (hi2c->State == HAL_I2C_STATE_READY)
  {
  // etc...
```

Want to use a HAL Function?

- `HAL_[Peripheral]_[Function]`