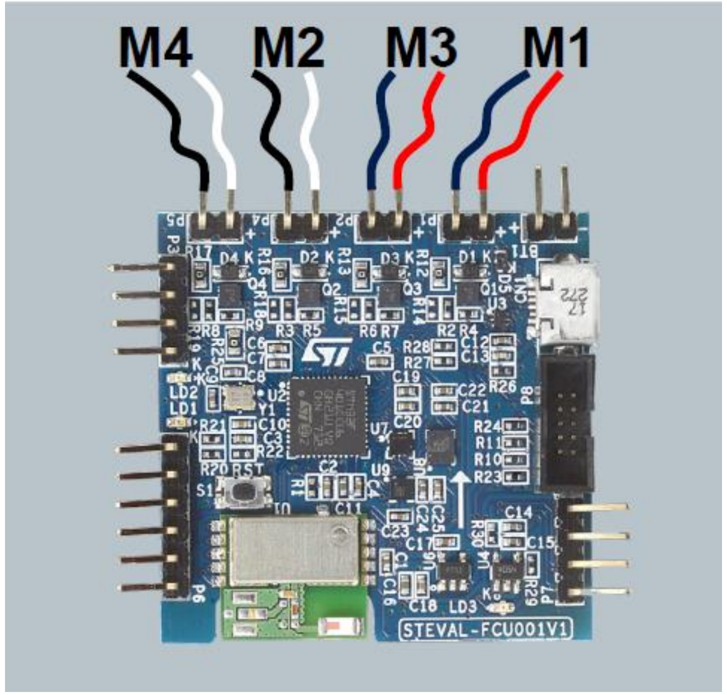
Embedded systems with drones – Hands-on lecture LAB2

Project-based Learning Center | ETH Zurich
Tommaso Polonelli tommaso.polonelli@pbl.ee.ethz.ch
Michele Magno michele.magno@pbl.ee.ethz.ch
Vlad Niculescu vladn@iis.ee.ethz.ch

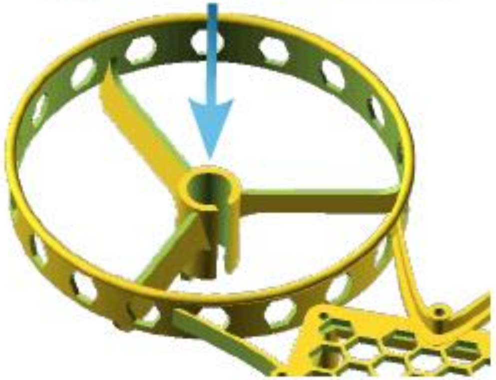
Evaluation board – Motor connection

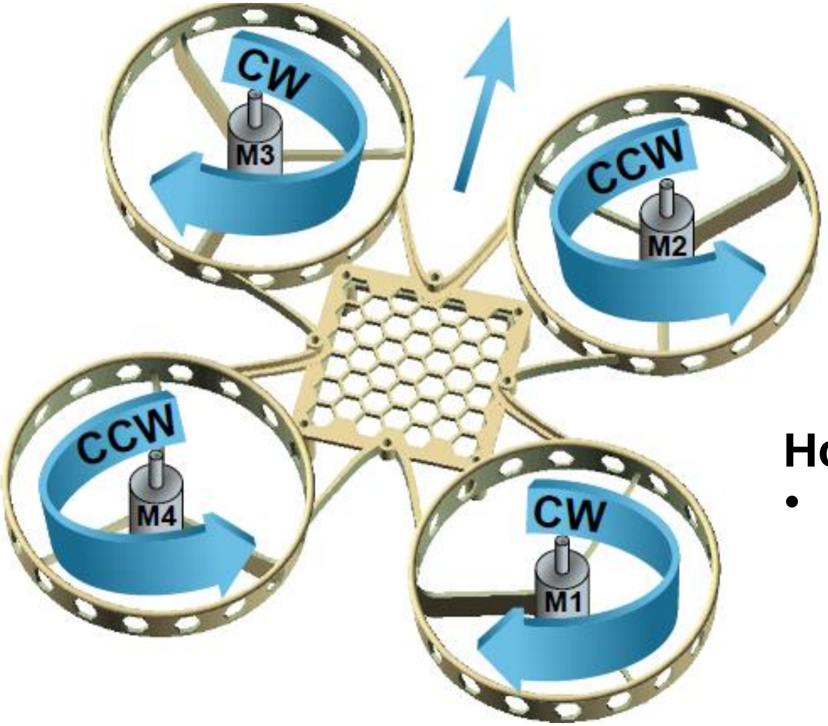


Motor supply connections on FCU board

- Place the four motors in the correct configuration with respect to the facing direction of the board:
  - The **clockwise** motors (M1 and M3) have the **red** (+) and **blue** (-) cables
  - The **counterclockwise** motors (M2 and M4) have **white** (+) and **black** (-) cables.

Motor housing



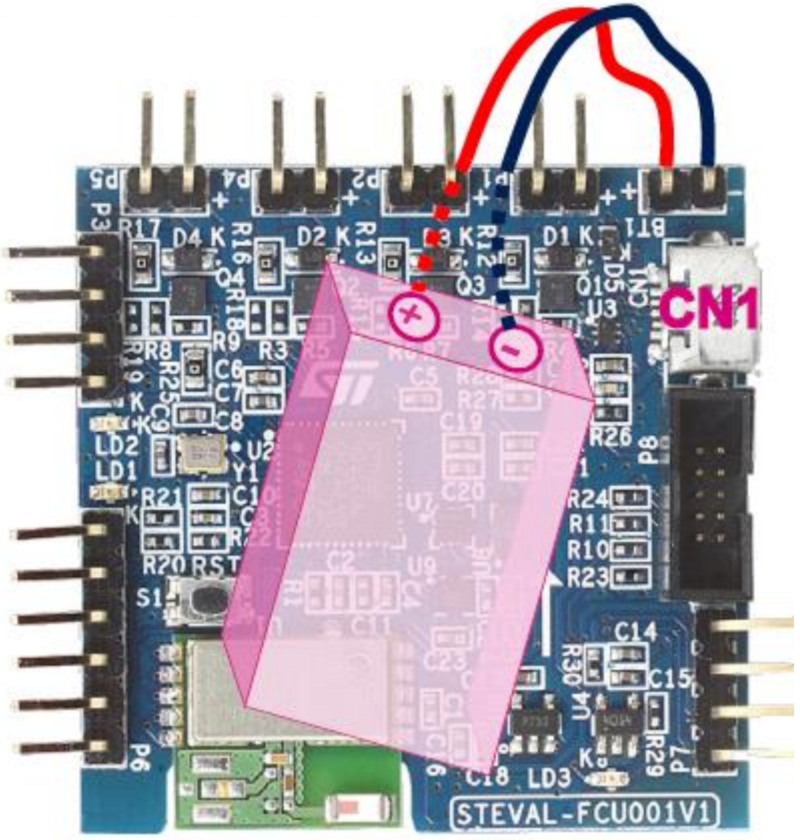


How to build your own minidrone:
- Document dm00563953.pdf on Polybox

Evaluation board – Battery connection

Motor supply connections on FCU board

- CN1 recharge the battery
- BT1 battery connector

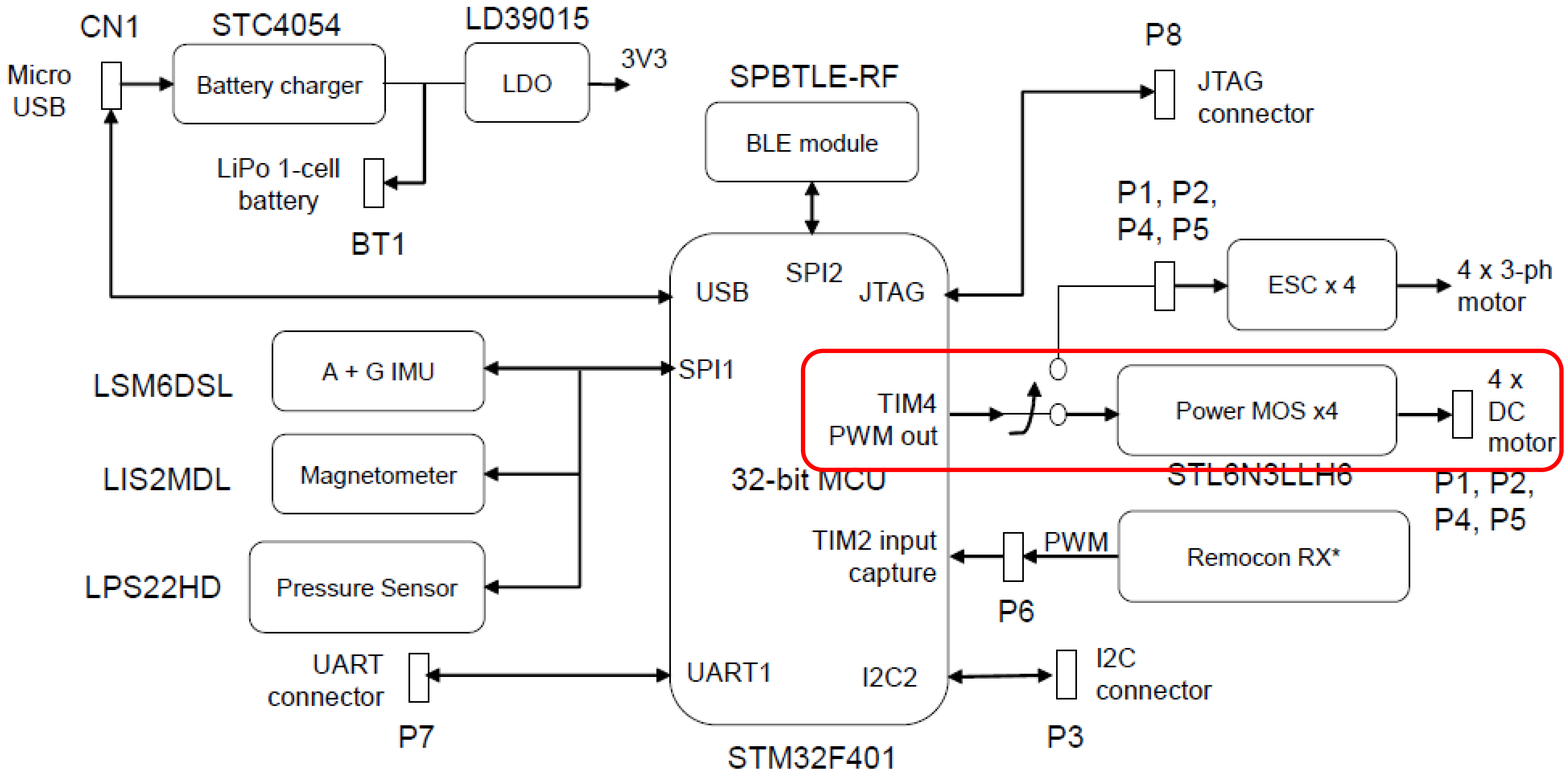


BE AWARE If you invert the battery polarity:

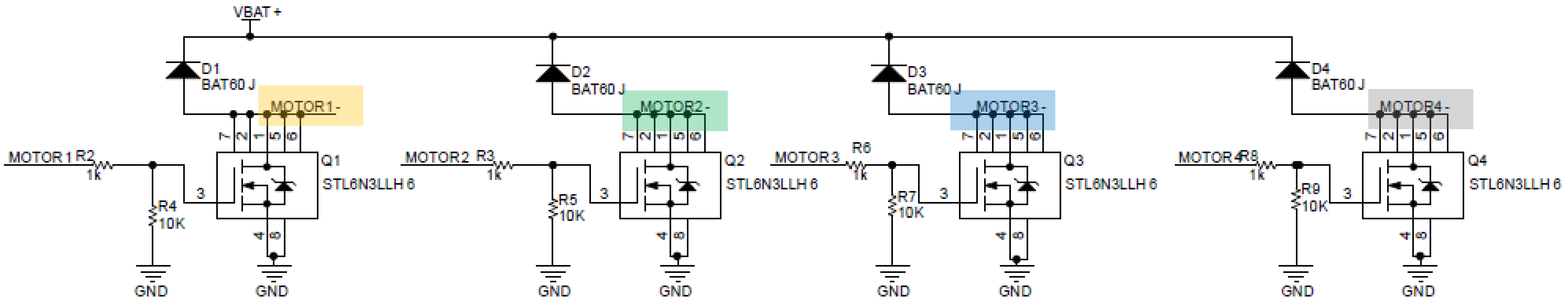
THERE IS NOT INVERSION OR OVERCURRENT PROTECTION

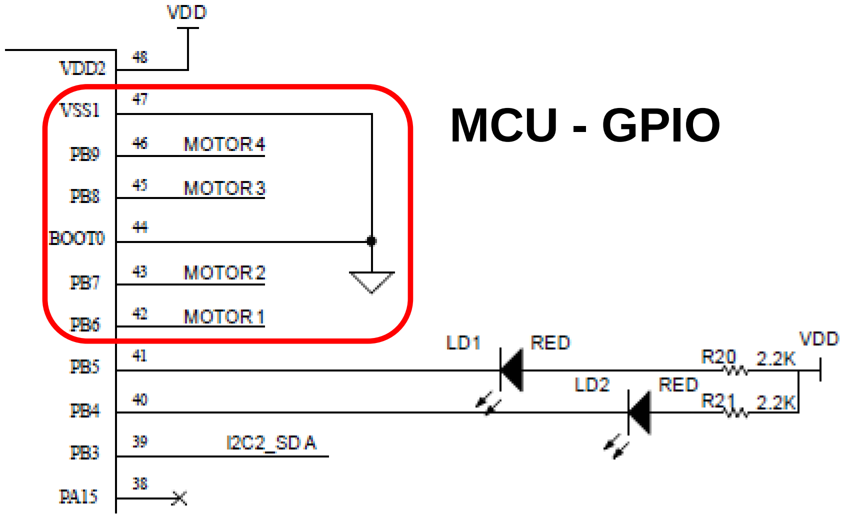
How to build your own minidrone:
- Document dm00563953.pdf on Polybox

Evaluation board – STEVAL-FCU001V1

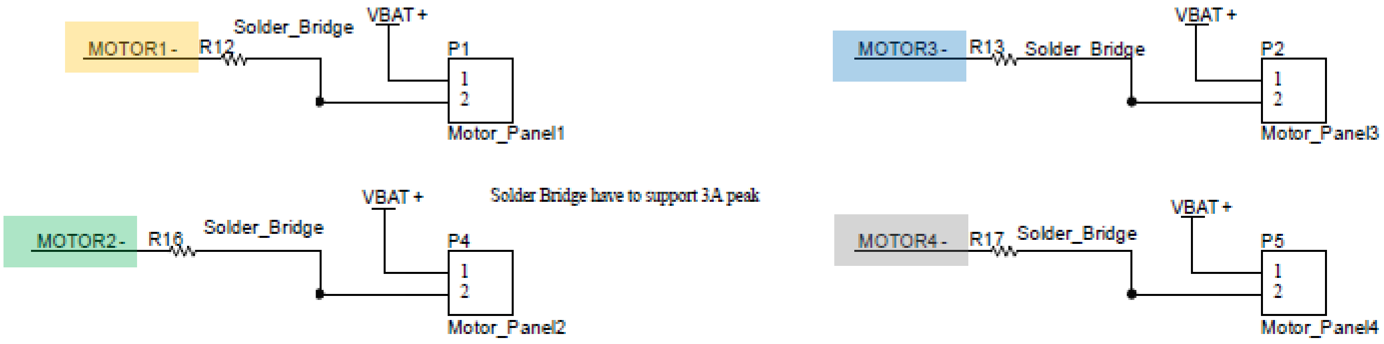


Evaluation board – STEVAL-FCU001V1

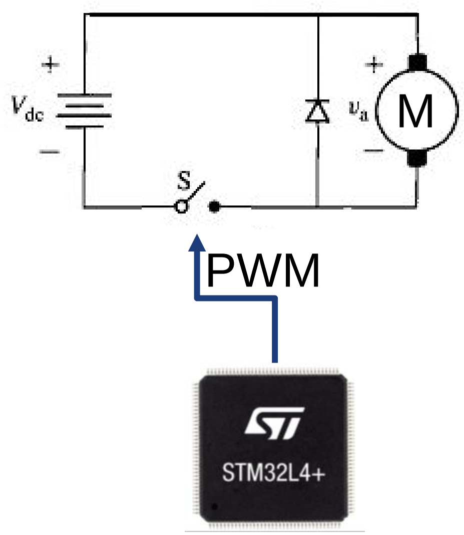




MCU - GPIO



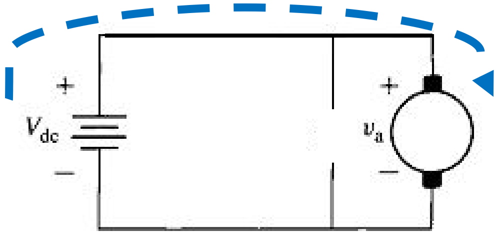
Evaluation board – STEVAL-FCU001V1

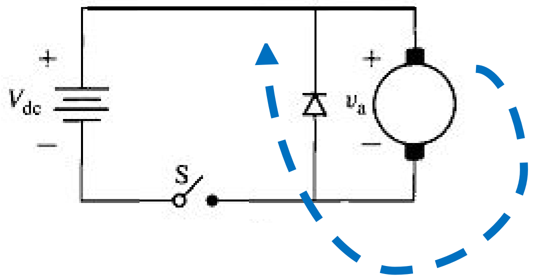


PWM

MCU

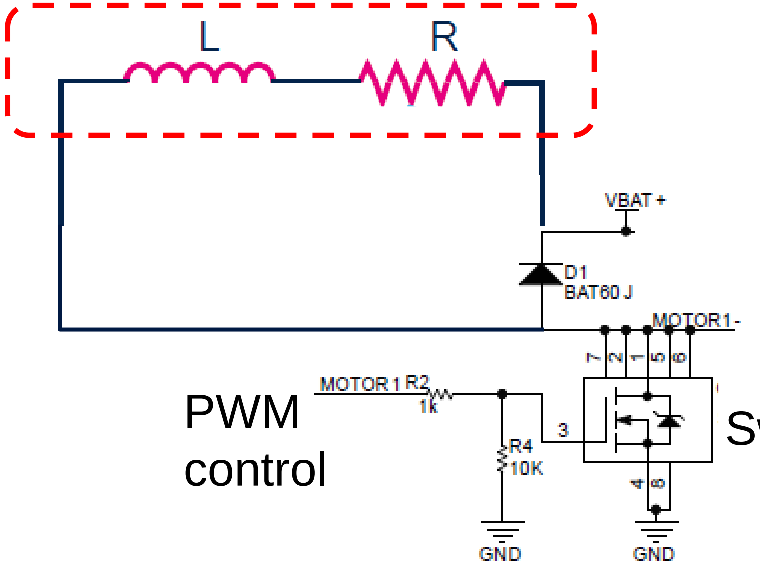
current





current

Motor



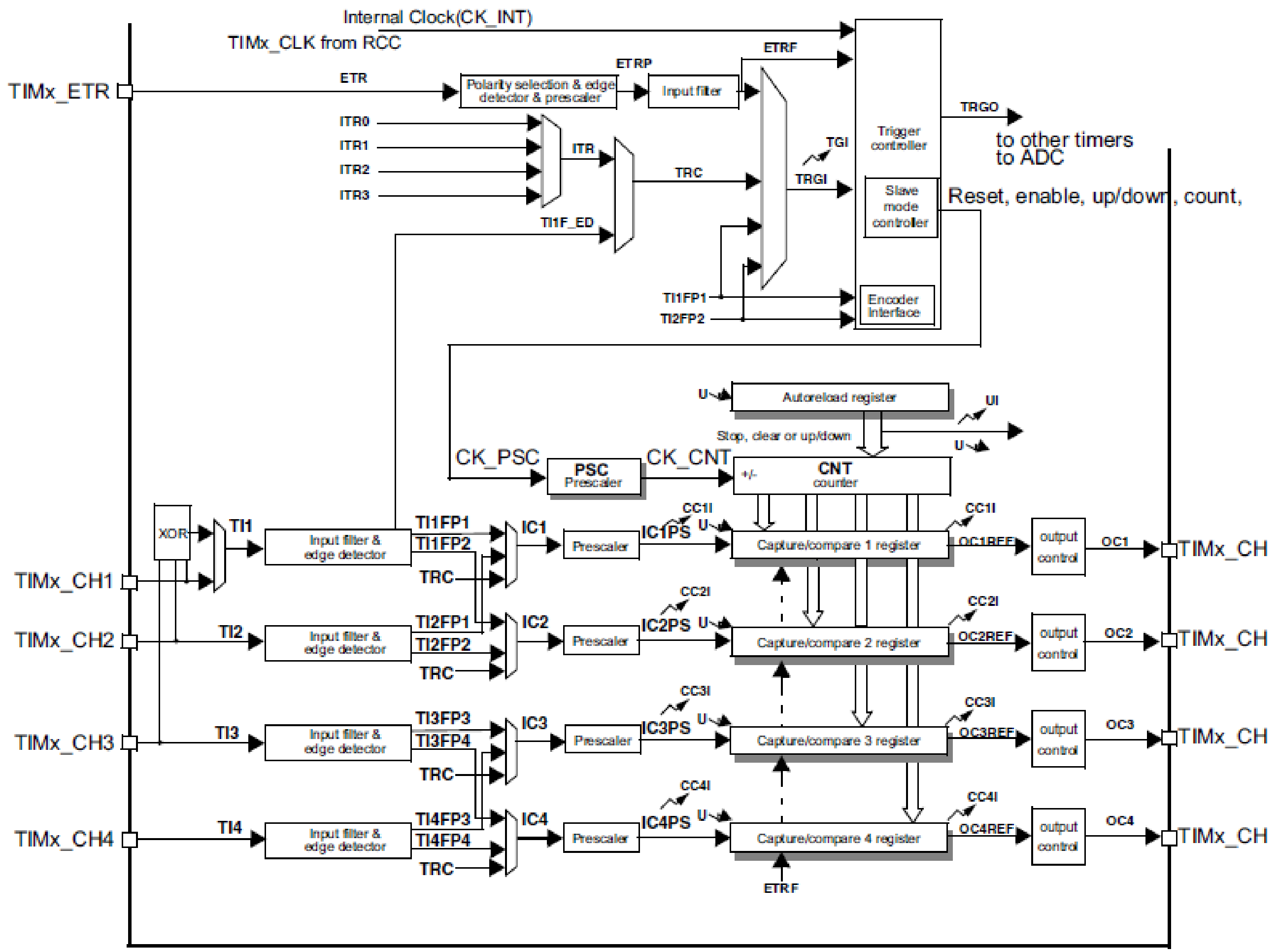
PWM control

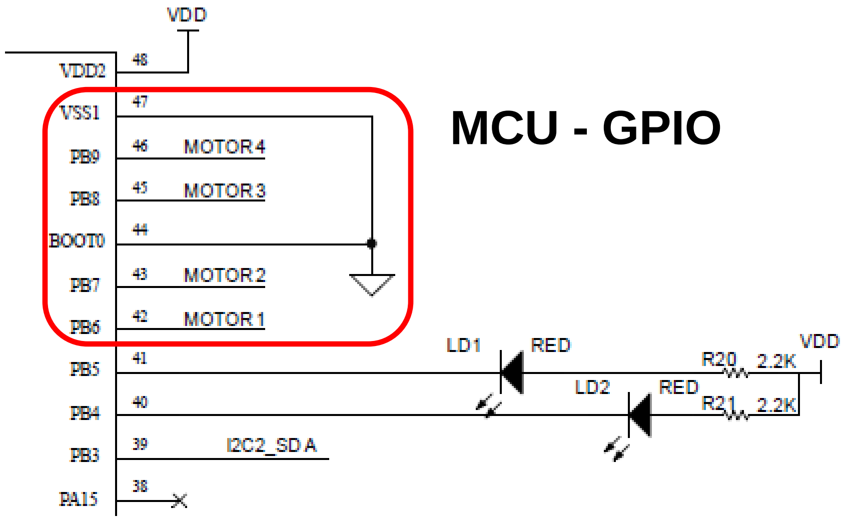
Switch

Electric implementation

TIMERS

- TIM4 block diagram





MCU - GPIO

Why Timer 4?

- Reference Manual (pag. 316)

39 General-purpose timers (TIM15/TIM16/TIM17)

39.1 TIM15/TIM16/TIM17 introduction

The TIM15/TIM16/TIM17 timers consist of a 16-bit auto-reload counter driven by a programmable prescaler.

They may be used for a variety of purposes, including measuring the pulse lengths of input signals (input capture) or generating output waveforms (output compare, PWM, complementary PWM with dead-time insertion).

Pulse lengths and waveform periods can be modulated from a few microseconds to several milliseconds using the timer prescaler and the RCC clock controller prescalers.

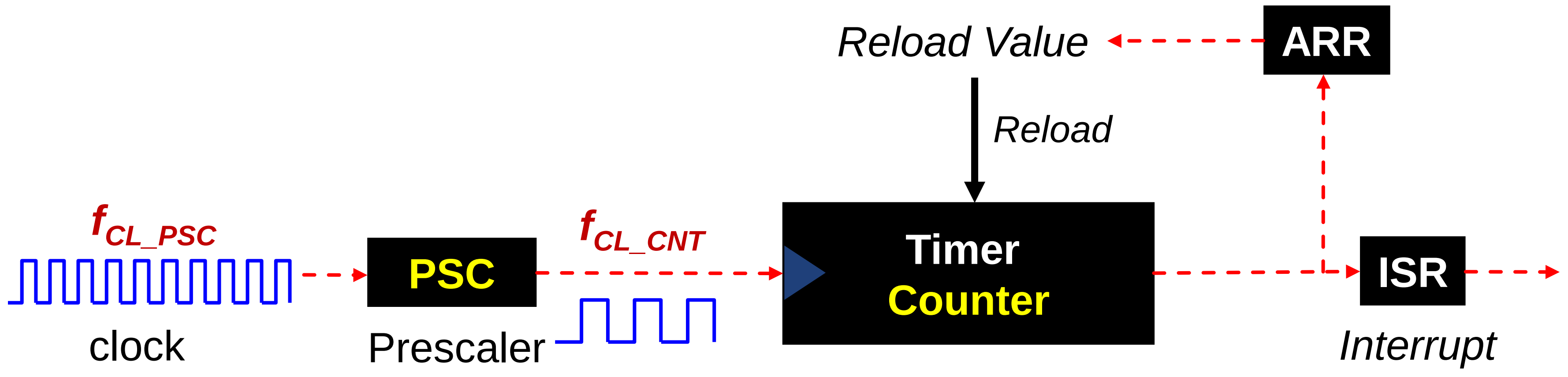
The TIM15/TIM16/TIM17 timers are completely independent, and do not share any resources. TIM15 can be synchronized as described in *Section 39.4.22: Timer synchronization (TIM15)*.

- 16-bit TIMER
- UP Counter
- 16-bit Prescaler
- PWM support

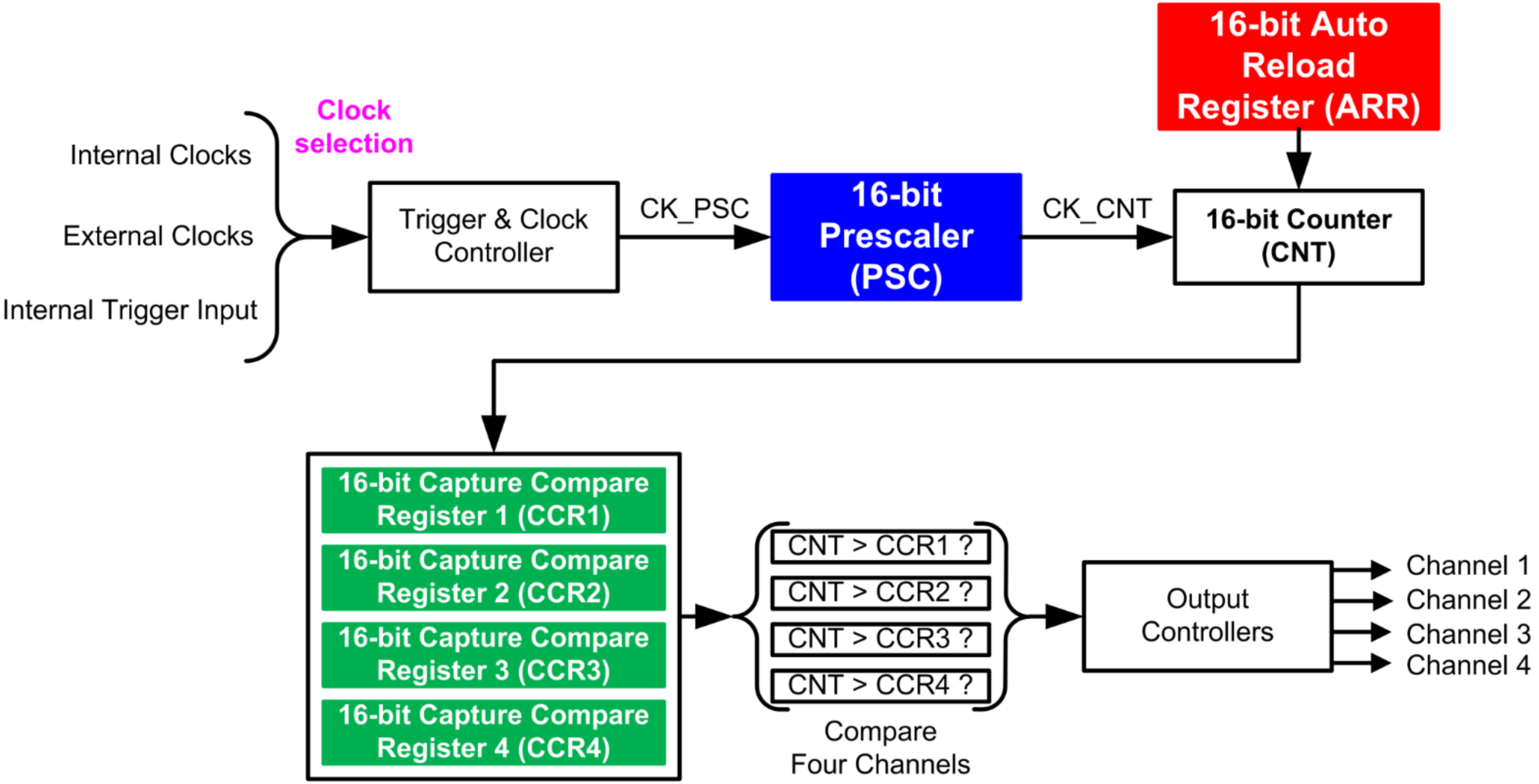
- Datasheet (pag. 42)

43	D4	59	93	B4	PB7	I/O	FT	-	I2C1_SDA, USART1_RX, TIM4_CH2, EVENTOUT	-
44	A5	60	94	A4	BOOT0	I	B	-	-	V_{PP}
45	B5	61	95	A3	PB8	I/O	FT	-	I2C1_SCL, TIM4_CH3, TIM10_CH1, SDIO_D4, EVENTOUT	-
46	C5	62	96	B3	PB9	I/O	FT	-	SPI2_NSS/I2S2_WS, I2C1_SDA, TIM4_CH4, TIM11_CH1, SDIO_D5, EVENTOUT	-
-	-	-	97	C3	PE0	I/O	FT	-	TIM4_ETR, EVENTOUT	-
-	-	-	98	A2	PE1	I/O	FT	-	EVENTOUT	-

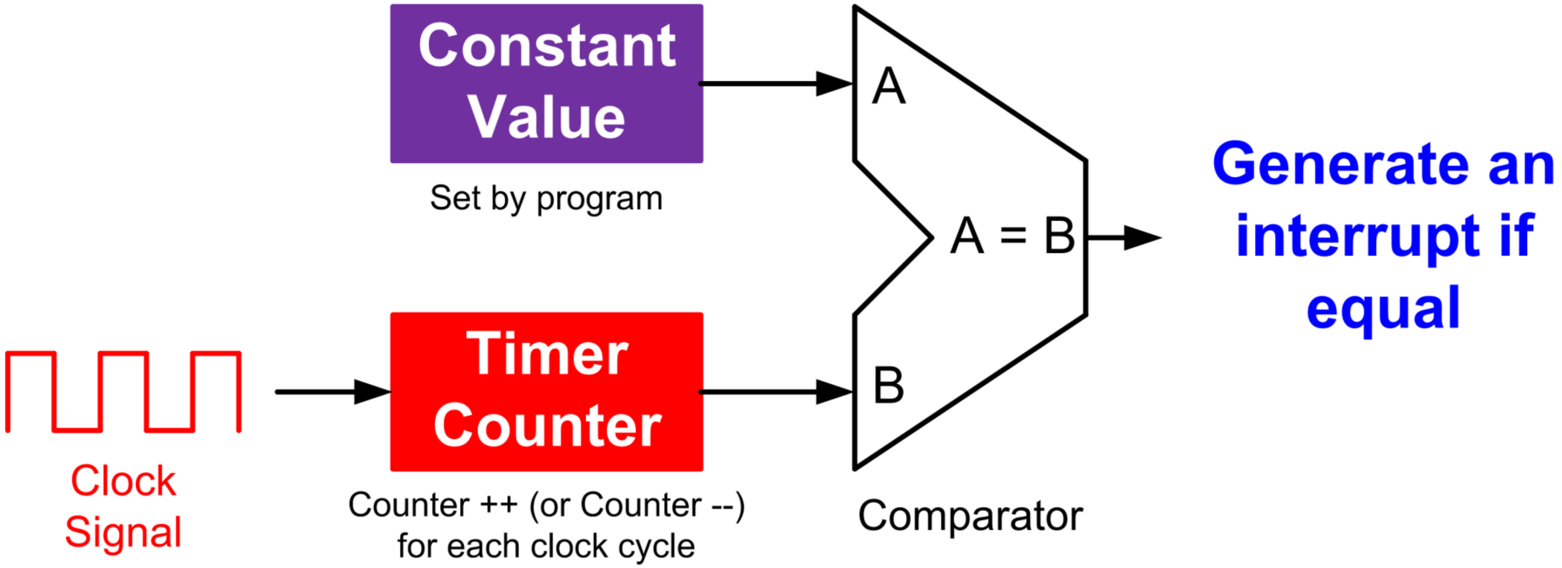
PWM Mode


Reload Value
ARR
Reload
f_{CL_PSC}
PSC
f_{CL_CNT}
Timer
Counter
ISR
clock
Prescaler
Interrupt

Multi-Channel Outputs

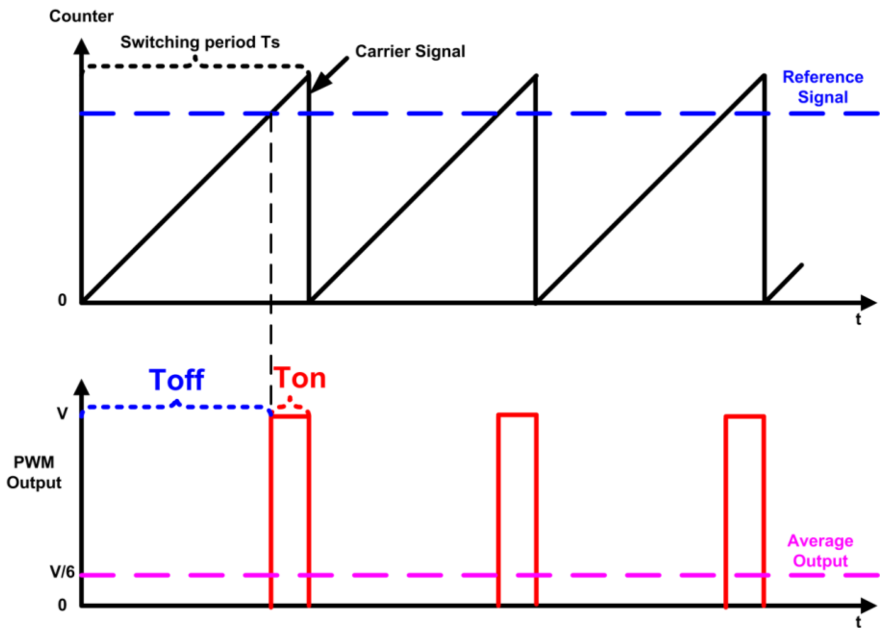


Output compare



Output Compare Mode (OCM)	Timer Output (OCREF)
000	Frozen
001	High if CNT == CCR
010	Low if CNT == CCR
011	Toggle if CNT == CCR
100	Forced low (always low)
101	Forced high (always high)

PWM Mode

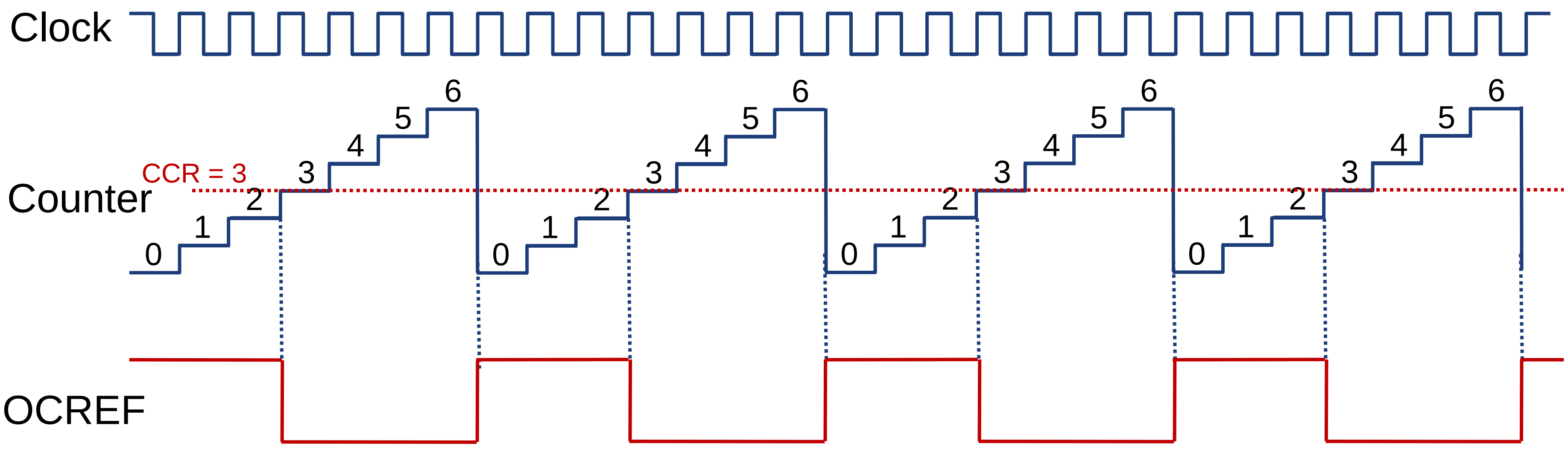


Mode	Counter < Reference	Counter ≥ Reference
PWM mode 1 (Low True)	Active	Inactive
PWM mode 2 (High True)	**Inactive**	**Active**

PWM Mode 1 (Low-True)

Mode 1

Timer Output = { High if counter < CCR ; Low if counter ≥ CCR }



Duty Cycle = $\frac{CCR}{ARR + 1}$

$= \frac{3}{7}$

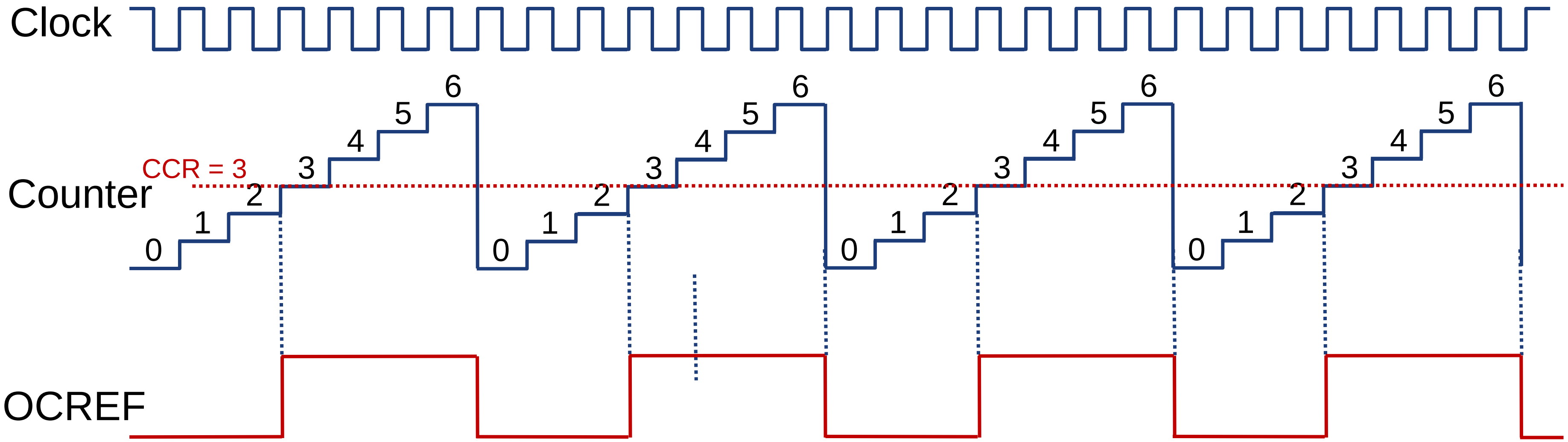
Period = (1 + ARR) * Clock Period

= 7 * Clock Period

PWM Mode 2 (High-True)

Mode 2

$$\text{Timer Output} = \begin{cases} \text{Low if counter} < \text{CCR} \\ \text{High if counter} \geq \text{CCR} \end{cases}$$

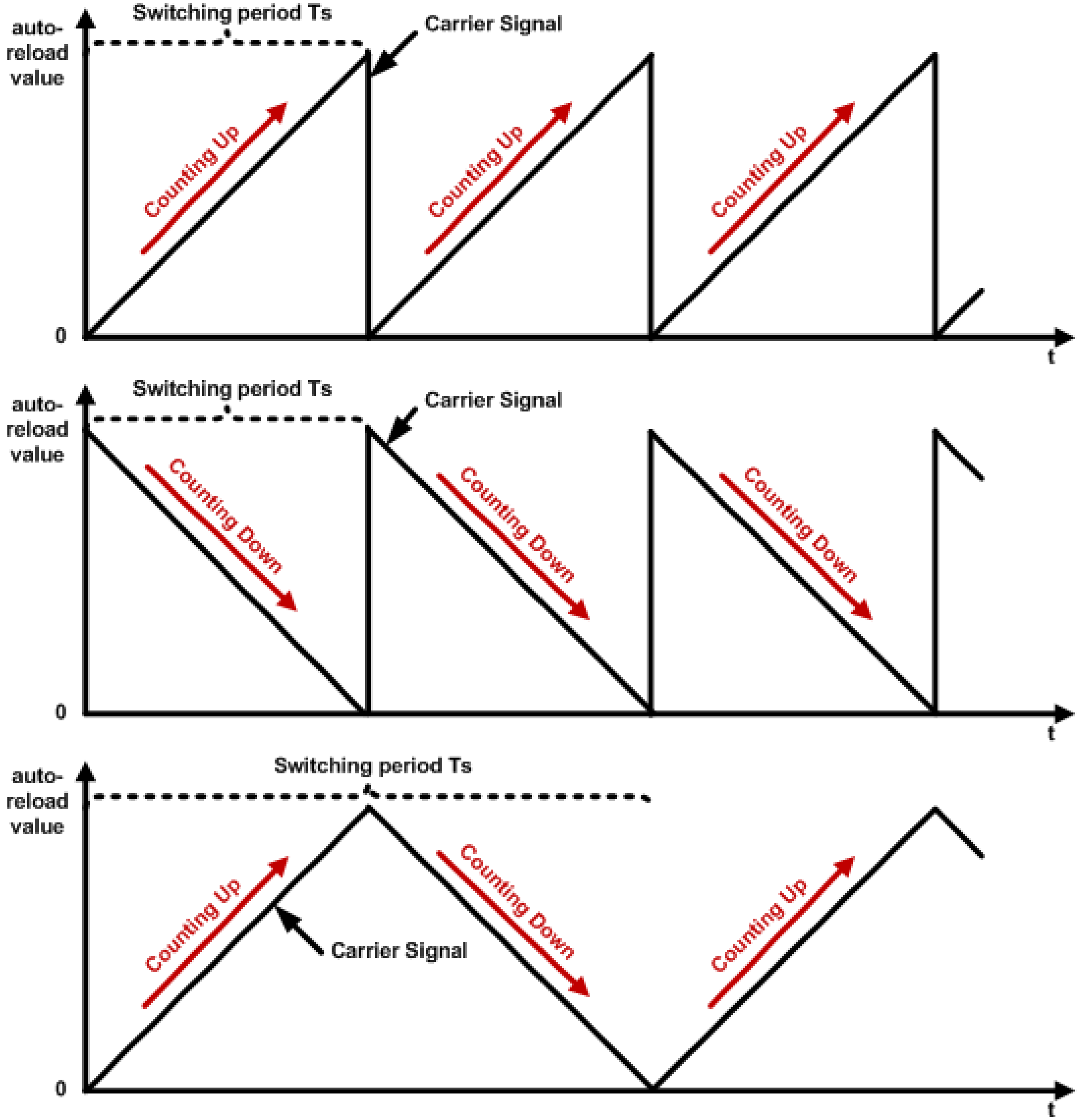


$$\text{Duty Cycle} = 1 - \frac{\text{CCR}}{\text{ARR} + 1} = \frac{4}{7}$$

$$\text{Period} = (1 + \text{ARR}) * \text{Clock Period} = 7 * \text{Clock Period}$$

ETH zürich

Counting up, down, center



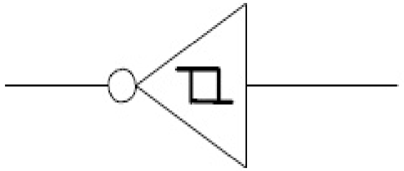
GPIO

GPIO: Intuition

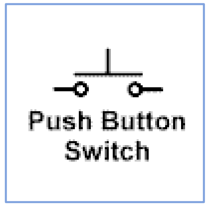
Take a bit from the physical world and put it in software (or vice versa).

- **Input**
  - Buttons

Somewhere in a register of our MCU

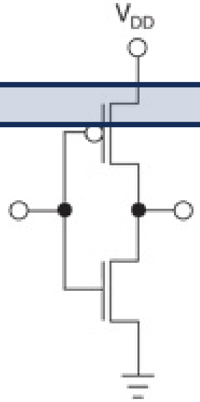


Voltage logic level to be read

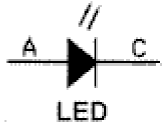


- **Output**
  - LEDs

Somewhere in a register of our MCU



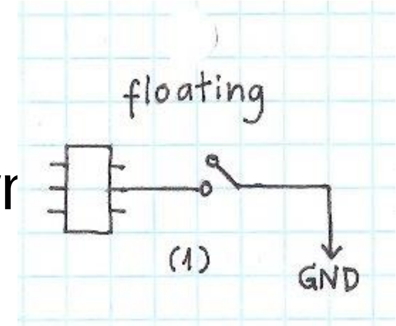
Voltage logic level to be set

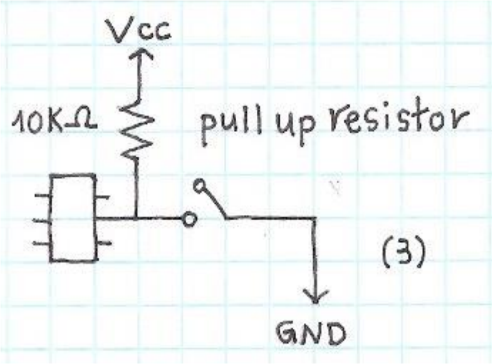


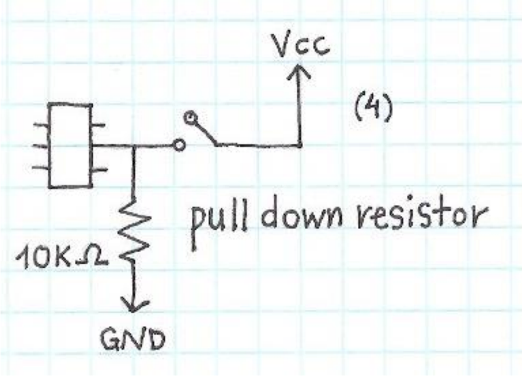
le

GPIO: operating modes

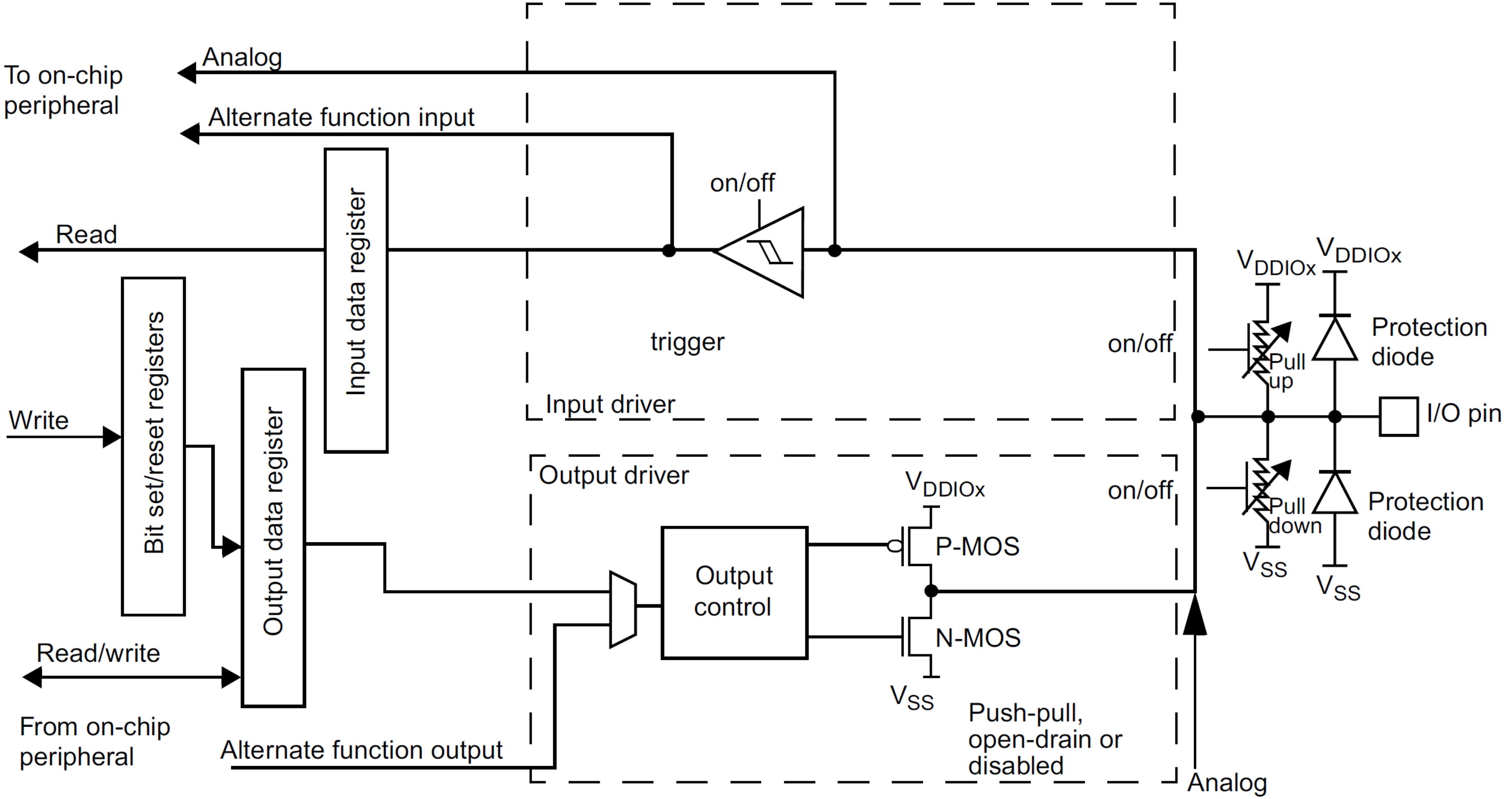
- Input mode
  - Floating
  - Input wit pull-up/down
  - Analog input mode
- Output mode
  - Push-pull, open drain
- Configurable output up to 80 MHz



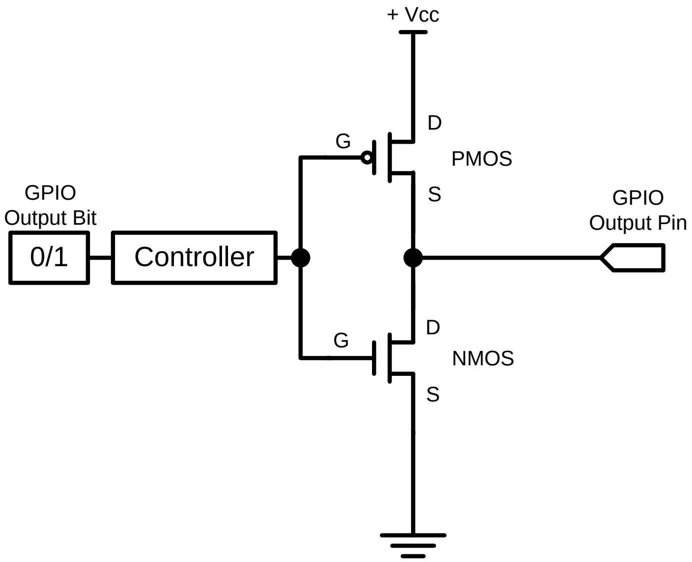


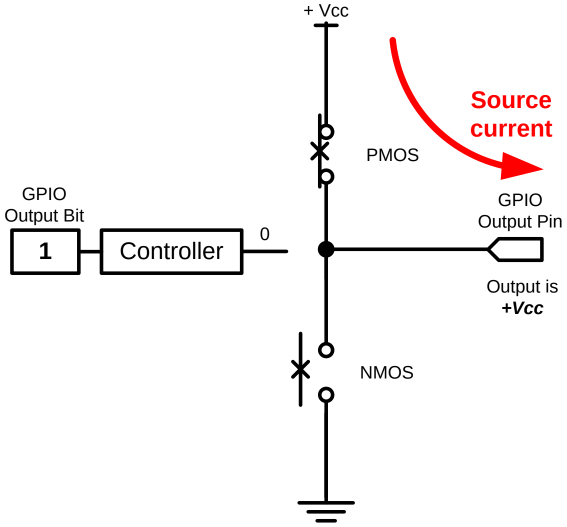


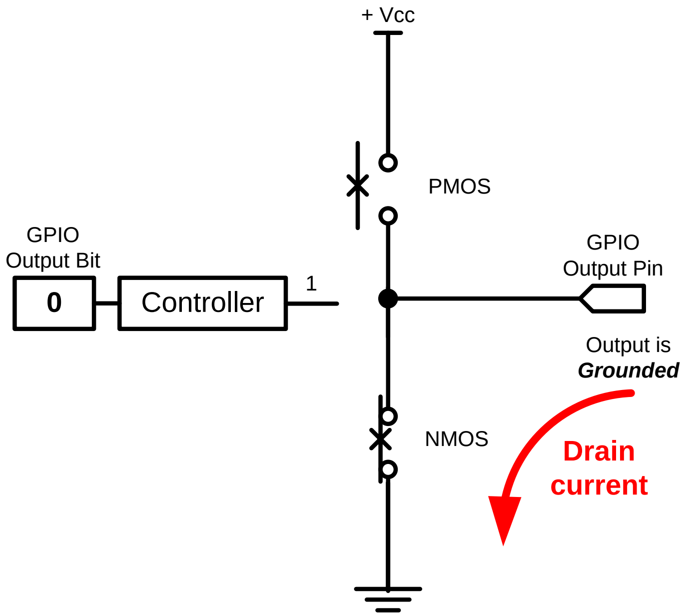
General Purpose Input/Output (GPIO)



GPIO Output: Push-Pull





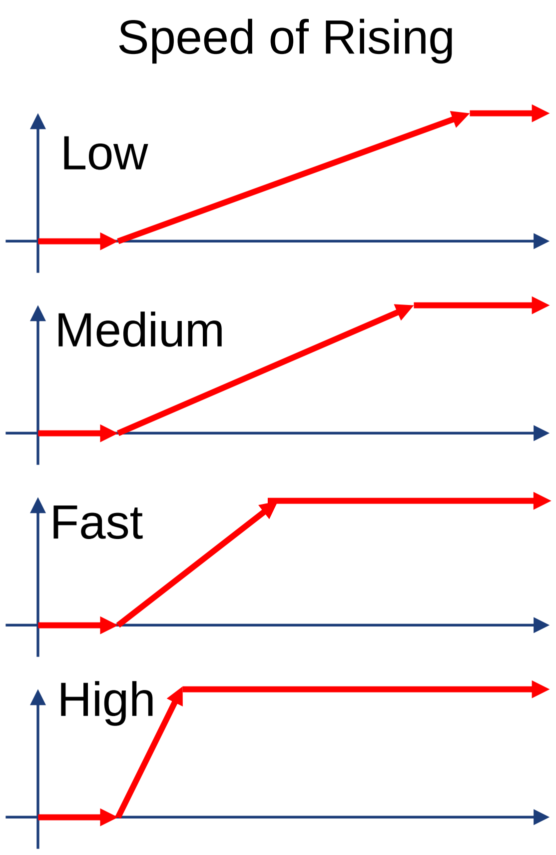


GPIO Output = 1
Source current to external circuit

GPIO Output = 0
Drain current from external circuit

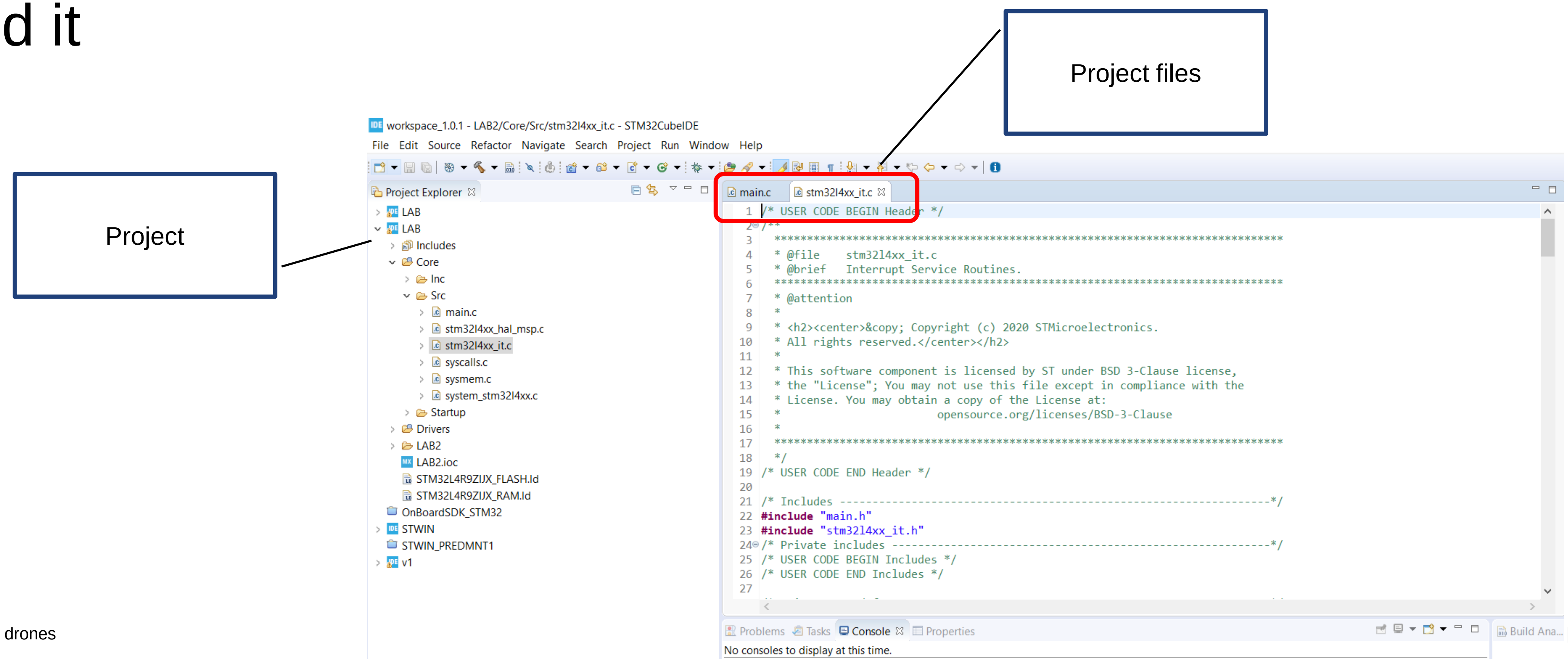
GPIO Output Speed

- Output Speed:
  - Speed of rising and falling
  - Four speeds: Low, Medium, Fast, High
- Tradeoff
  - Higher GPIO speed increases EMI noise and power consumption
  - Configure based on peripheral speed
    - Low speed for toggling LEDs
    - High speed for SPI

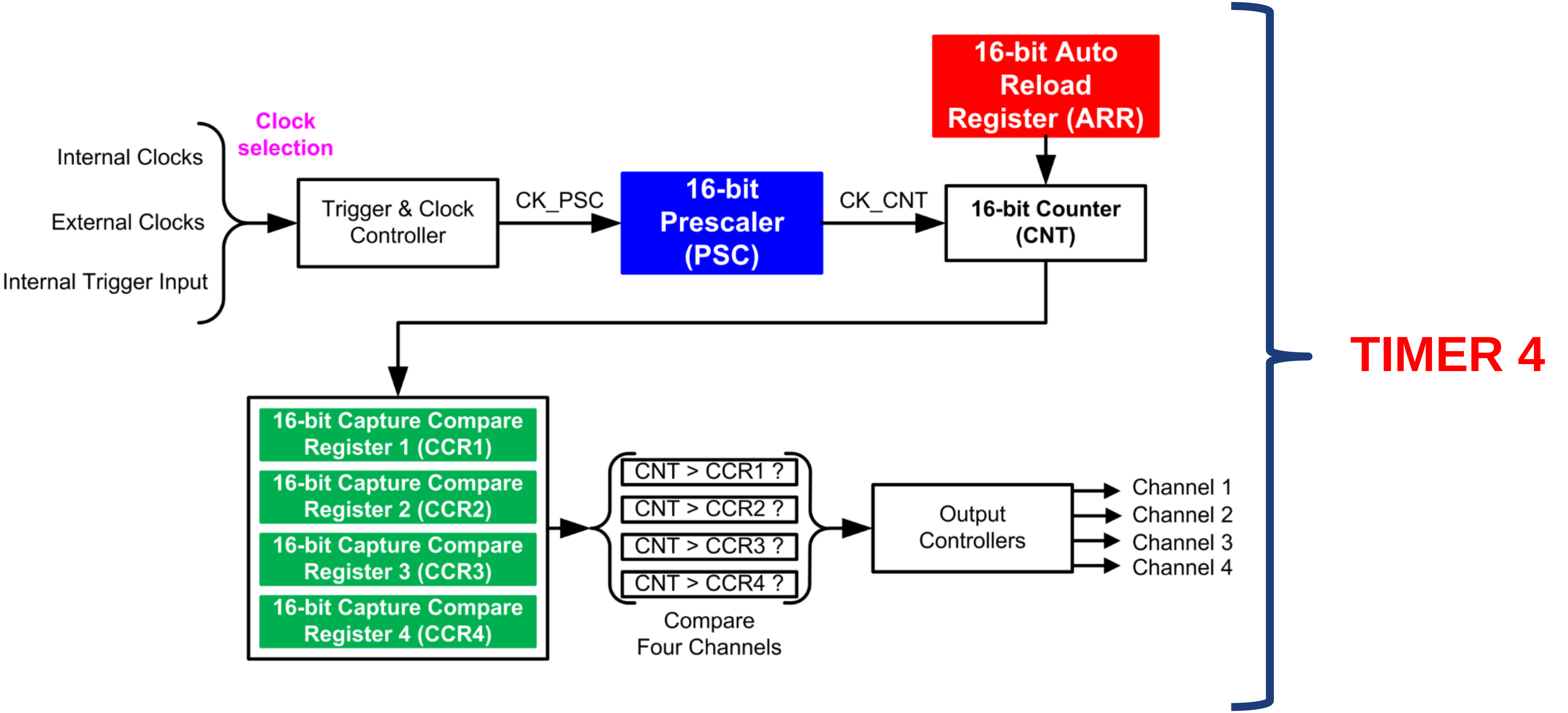


Preparation for LAB3

- Import in CubeIDE LAB2
- Open STM32CubeMX and open LAB2 config. file
- Build it



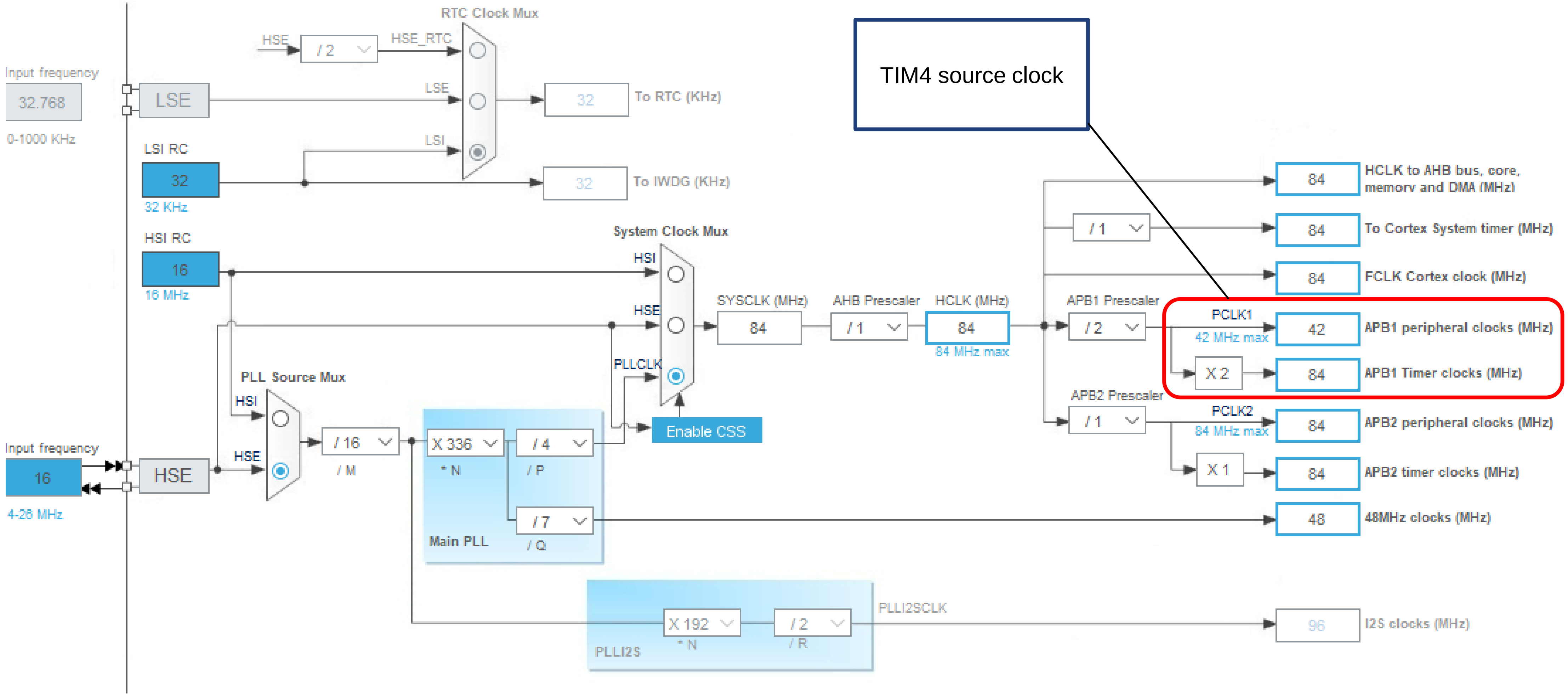
PWM Motor Control on our STM32F401



Reference Manual pag. 316

Clock Tree

- Clock tree (Reference Manual pag. 94)



Frequency and Times

Example, a **16 bit Timer**

$f_{SystemClock}=f_{timer}=84MHz$

we have a tick in every,

$T_{timer}=1/f_{PCLK2}=1/f_{timer}=1/84MHz=12\ ns$

With a **16 bit Timer** it means,

$ticks_{max}=(2^{16}-1)=65535ticks$

So the timer will overflow every,

$t_{overflow}=ticks_{max}\times T_{timer}=65535\times 12\ ns = 780\ \mu s$

Timer Prescaler

We want to count a clock of 10KHz (0.1 ms tick), how we set the timer clock and prescaler?

CK_CNT = TIM_CK / PrescalerValue

CK_CNT = the clock being counted; we want it to 10KHz (this is your choice)

TIM_CK = input clock for TIMx

PrescalerValue = divider of the PCLK clock, we have to set it to obtain the desired CK_CNT:

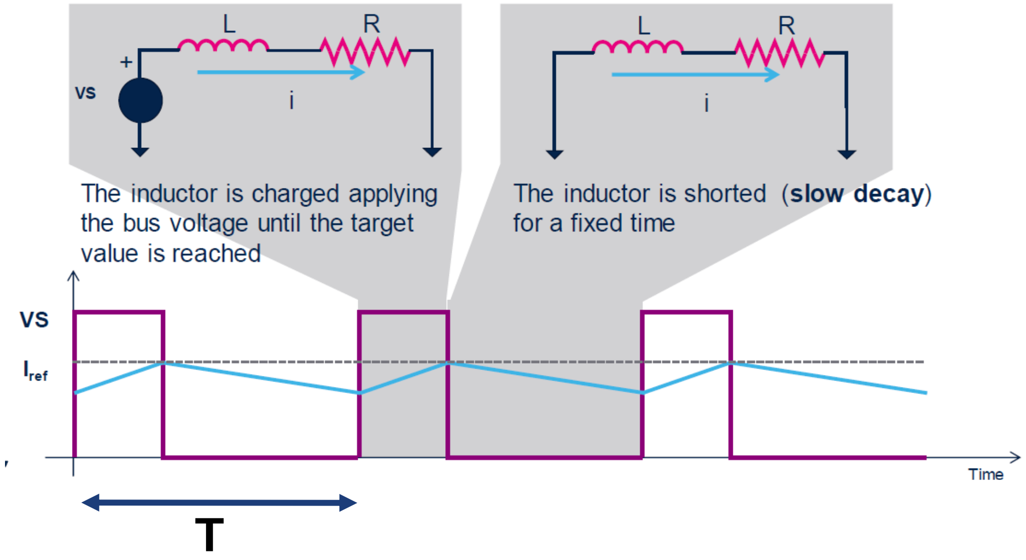
PrescalerValue = ((PCLKx**) / CK_CNT) – 1**
PrescalerValue = ((84 MHz**) / 10KHz) – 1 = 8400**

Timer Period

Programming the **prescaler** we set a precise clock frequency, now we need to define the Timer Period value, which is calculated from the period **T** that we need:

TIM_Period = (CK_CNT * T)-1

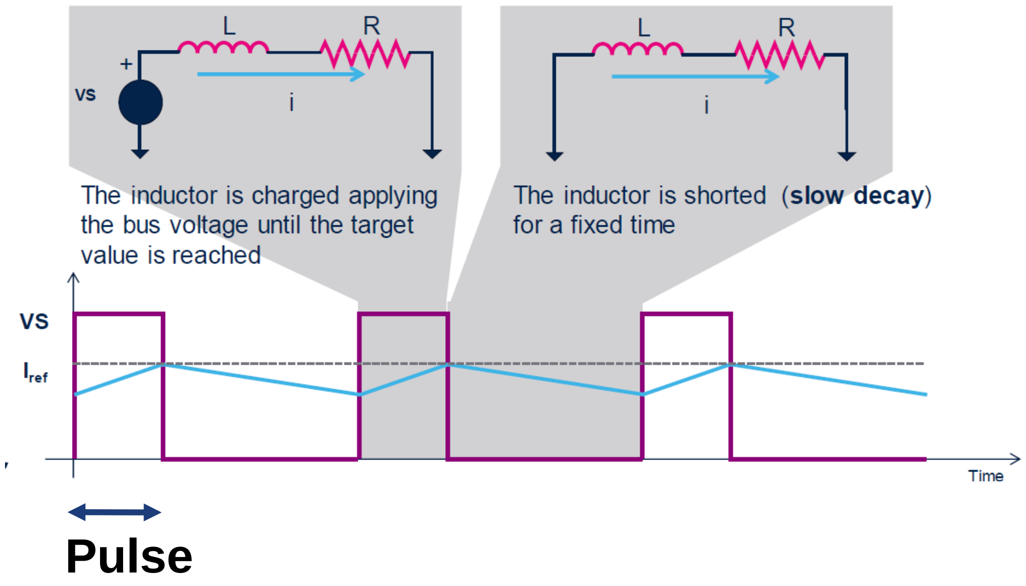
Integer value always < (2^{16}-1) (16-bit counter)



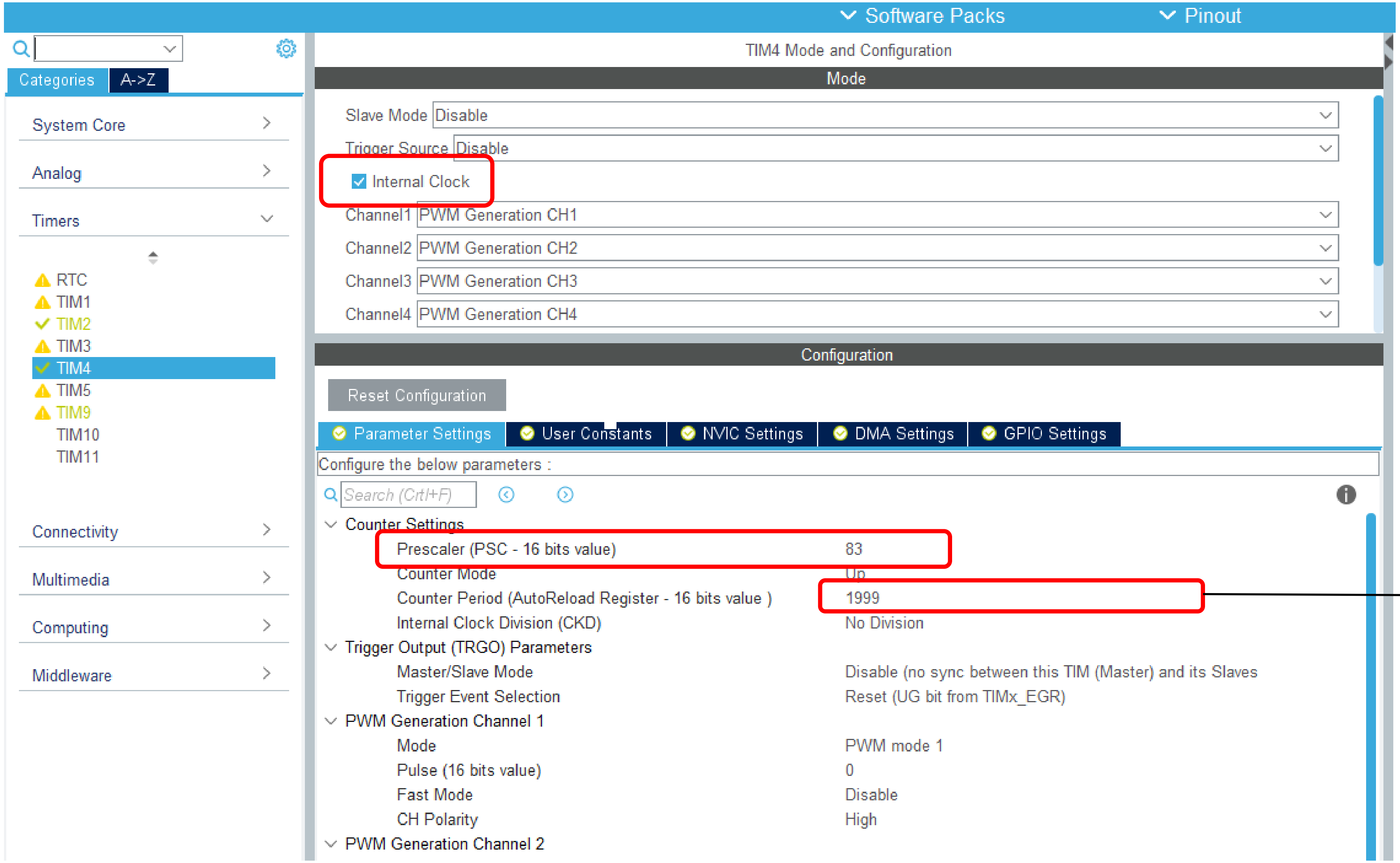
Timer Pulse (in PWM 1)

Programming the **Pulse** register define the PWM duty cycle (DC), exactly what we need to control our motor:

Pulse = TIM_Period * DC



TIM4 – Cube MX



Timer Period, maximum

TIM4 – Settings for our DRONE-KIT

TIM4, a **16 bit Timer**

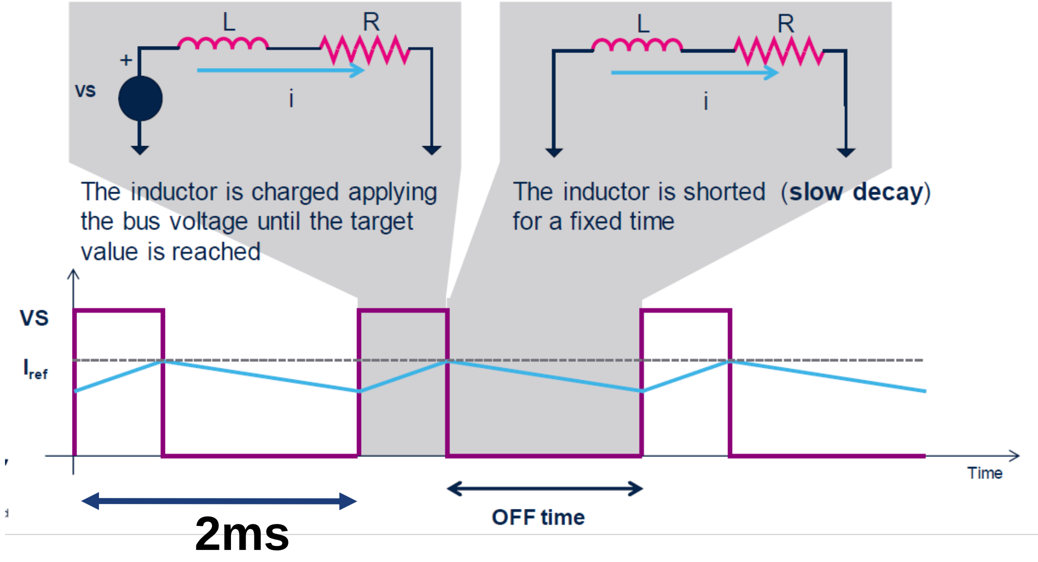
$f_{SystemClock}=f_{timer}=84MHz$

tick in every,

$T_{timer}=1/f_{PCLK2}=1/f_{timer}=1/84MHz=12\ ns$

CK_CNT = TIM_CK / PrescalerValue = **1 MHz**

T = (TIM_Period + 1) / CK_CNT **= 2 ms (500 Hz)**



TIM4 – Cube IDE

```
/**
  * @brief  Starts the PWM signal generation.
  * @param  htim: pointer to a TIM_HandleTypeDef structure that contains
  *                the configuration information for TIM module.
  * @param  Channel: TIM Channels to be enabled.
  *          This parameter can be one of the following values:
  *            @arg TIM_CHANNEL_1: TIM Channel 1 selected
  *            @arg TIM_CHANNEL_2: TIM Channel 2 selected
  *            @arg TIM_CHANNEL_3: TIM Channel 3 selected
  *            @arg TIM_CHANNEL_4: TIM Channel 4 selected
  * @retval HAL status
  */
```

```
HAL_TIM_PWM_Start(&htim4,TIM_CHANNEL_1);
```

```
void MX_TIM4_Init(void)
{

  TIM_ClockConfigTypeDef sClockSourceConfig;
  TIM_MasterConfigTypeDef sMasterConfig;
  TIM_OC_InitTypeDef sConfigOC;

  htim4.Instance = TIM4;
  htim4.Init.Prescaler = 84;
  htim4.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim4.Init.Period = 1999;

  htim4.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  HAL_TIM_Base_Init(&htim4);

  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  HAL_TIM_ConfigClockSource(&htim4, &sClockSourceConfig);

  HAL_TIM_PWM_Init(&htim4);

  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  HAL_TIMEx_MasterConfigSynchronization(&htim4, &sMasterConfig);

  sConfigOC.OCMode = TIM_OCMODE_PWM1;
  sConfigOC.Pulse = 0;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  HAL_TIM_PWM_ConfigChannel(&htim4, &sConfigOC, TIM_CHANNEL_1);
```

Code Template (Polybox – LAB2 – main.c)

```c
/* Initialize TIM4 for Motors PWM Output*/
HAL_TIM_PWM_Start(&htim4,TIM_CHANNEL_1);
/****************************************/
/* insert here motor control for M2, M3 and M4*/

/* set motor PWM to 0 DC */
set_motor_pwm_zero(&motor_pwm);
```

Start TIM4 and enable CH 1

Set all the CHANNEL Pulse to 0 motors off

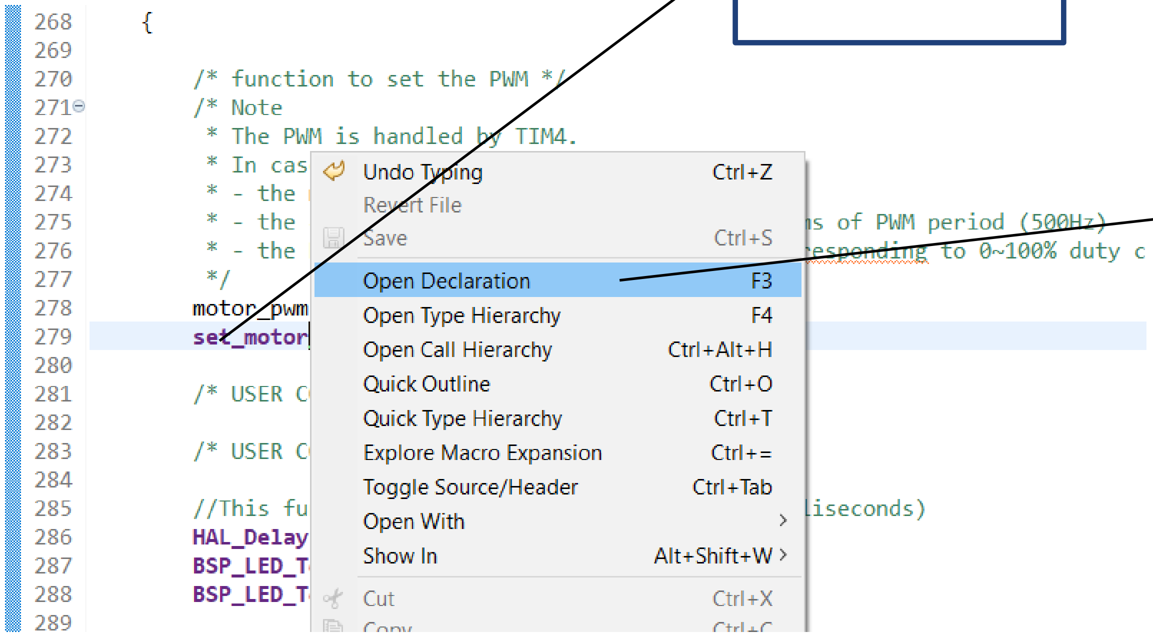
```c
while (1)
{

    /* function to set the PWM */
    /* Note
     * The PWM is handled by TIM4.
     * In case of DC motor configuration:
     * - the master clock for TIM4 is 1MHz
     * - the counter counts up to 2000, result in 2ms of PWM period (500Hz)
     * - the PWM pulse width data can to 0~1999, coresponding to 0~100% duty cycle
    */
    motor_pwm.motor1_pwm = 99;
    set_motor_pwm(&motor_pwm);
```

MOTOR 1 Pulse value
NOTE: this is the Pulse value not the DC

Set the PWM for all motors

STM32CubeIDe: Tips

Right Click



Open Declaration
automatic function finder in project files

LAB2: Exercise overview

Exercise	Assignment	Concept
Exercise 1	- Run the LAB2 template, Motor 1 should spin slowly	Start program, Debug
Exercise 2	- Enable all the motors ad the same speed, and at different fixed speed (random choiche)	PWM, TIM
Exercise 3	- Sweep the DC between 0 to 100% in loop with 1 sec period and a step of 10% (10 steps in total)	PWM, TIM, C programming
Questions:	- To generate 1 Hz square wave with a duty cycle of 50%, how should we set up the timer? - What is the smallest PWM frequency that can be generated? - What is the shortest ON pulse that can be generated?	Programming and debugging

LAB2: PRO - Exercises

Exercise	Assignment	Concept
Exercise 1	- Receive commands from the terminal Letters: S: enable the motors O: disable the motors P: increase the rpm by 10% M: decrease the rpm by 10%	PWM, TIM
Exercise 2	At start-up: generate a welcome theme song using the motors	PWM, TIM