GPIO HAL

```
HAL_GPIO_WritePin(XXX_GPIO_Port, XXX_Pin, GPIO_PIN_(RE)SET);

HAL_GPIO_ReadPin (XXX_GPIO_Port, XXX_Pin);

HAL_GPIO_TogglePin(XXX_GPIO_Port, XXX_Pin);

HAL_Delay(1000);

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) // Enable interrupt EXTI in NVIC

{

        if (GPIO_Pin == B1_Pin)

        { }

}
```

HAL/SysTick

```
#define DELAY_1s 1000

 static uint32_t lastTick = 0;

if (HAL_GetTick() - lastTick >= DELAY_1s)

{

lastTick = HAL_GetTick();

}
```

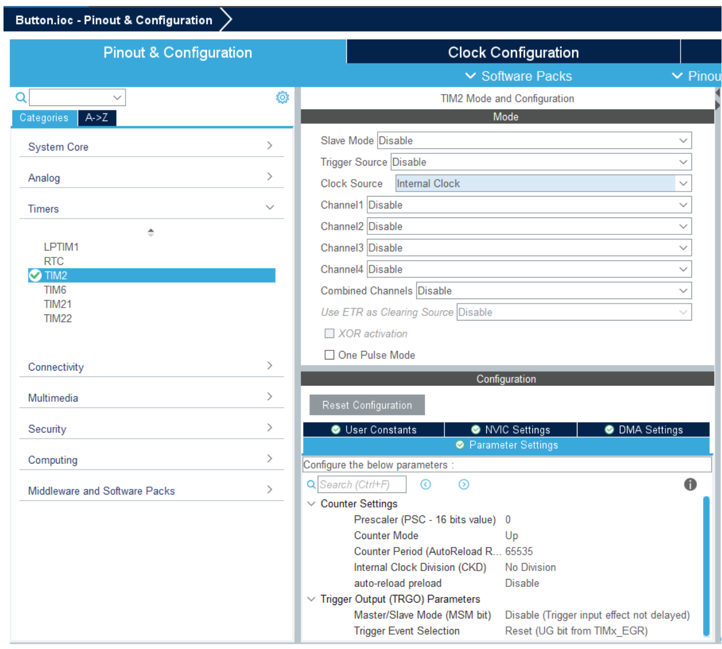
Timers

In main while without interupt

```
HAL_TIM_Base_Start(&htim2);

if (__HAL_TIM_GET_FLAG(&htim2, TIM_FLAG_UPDATE) != RESET)

{

 __HAL_TIM_CLEAR_FLAG(&htim2, TIM_FLAG_UPDATE);

HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);

}
```

As interupt

```
HAL_TIM_Base_Start_IT(&htim2);

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef
*htim)

{

 if (htim->Instance == TIM2)

 { }

}
```



TIM2 Mode and Configuration

Mode

Slave Mode	Disable
Trigger Source	Disable
Clock Source	Internal Clock
Channel1	Disable
Channel2	Disable
Channel3	Disable
Channel4	Disable
Combined Channels	Disable
Use ETR as Clearing Source	Disable

☐ *XOR activation*

☐ One Pulse Mode

Configuration

Configure the below parameters :

Counter Settings	
Prescaler (PSC - 16 bits value)	0
Counter Mode	Up
Counter Period (AutoReload R...	65535
Internal Clock Division (CKD)	No Division
auto-reload preload	Disable
Trigger Output (TRGO) Parameters	
Master/Slave Mode (MSM bit)	Disable (Trigger input effect not delayed)
Trigger Event Selection	Reset (UG bit from TIMx_EGR)

ADC convertors

Project-Properties-C/C++Build-Settings-Linker Misc-other flags: -u _printf_float

```
float adc_value = 0.0f;

float volt_value = 0.0f;

HAL_ADC_Start(&hadc);

if (HAL_ADC_PollForConversion(&hadc, 10) == HAL_OK)

{

uint32_t value = HAL_ADC_GetValue(&hadc);

}
```

DAC convertors

```
HAL_DAC_Start(&hdac, DAC_CHANNEL_1);

HAL_DAC_SetValue(&hdac, DAC_CHANNEL_1, DAC_ALIGN_12B_R, 4095);
```

UART communication with DMA

Project-Properties-C/C++Build-Settings-Linker Misc-other flags: -u _printf_float

DMA: USART2_RX Circular mode

```
#include <string.h>

#include <stdio.h>

#define RX_BUFFER_LEN 64

#define CMD_BUFFER_LEN 256

static uint8_t uart_rx_buf[RX_BUFFER_LEN];

static volatile uint16_t uart_rx_read_ptr = 0;

#define uart_rx_write_ptr (RX_BUFFER_LEN - hdma_usart2_rx.Instance->CNDTR)

char msg[30];

static void uart_process_command(char *cmd) {

        char *token;

        token = strtok(cmd, " ");

        if (strcasecmp(token, "GetData") == 0) {

                sprintf(msg, "%.3f\r\n", 3.3);

                HAL_UART_Transmit(&huart2, (uint8_t*) msg, strlen(msg), HAL_MAX_DELAY);

        }

}

static void uart_byte_available(uint8_t c) {

        static uint16_t cnt;

        static char data[CMD_BUFFER_LEN];
```

```
            if (cnt < CMD_BUFFER_LEN && c >= 32 && c <= 126)
                        data[cnt++] = c;
            if ((c == '\n' || c == '\r') && cnt > 0) {
                        data[cnt] = '\0';
                        uart_process_command(data);
                        cnt = 0;
            }
}
HAL_UART_Receive_DMA(&huart2, uart_rx_buf, RX_BUFFER_LEN);
while (uart_rx_read_ptr != uart_rx_write_ptr) {
                                    uint8_t b = uart_rx_buf[uart_rx_read_ptr];
                                    if (++uart_rx_read_ptr >= RX_BUFFER_LEN)
                                                uart_rx_read_ptr = 0; // increase read pointer
                                    uart_byte_available(b); // process every received byte with the RX state machine
                        }
```

Python serial communication

```
import serial
PORT = "COM4"
BAUDRATE = 115200
ser = serial.Serial(PORT, BAUDRATE, timeout=1)
message = "GetData\r\n"
ser.write(message.encode())

line = ser.readline().decode(errors="ignore").strip()
adc_val = float(line)
print(f"Přijatá hodnota ADC: {adc_val} V")
ser.close()
```

SPI communication

```
uint8_t txData[] = "Hello"; uint8_t rxData[6];
HAL_SPI_Transmit(&hspi1, XXX, sizeof(XXX), HAL_MAX_DELAY)
HAL_SPI_Receive(&hspi1, XXX, sizeof(XXX), HAL_MAX_DELAY)
HAL_SPI_TransmitReceive (&hspi1, XXX, XXX, sizeof(XXX), HAL_MAX_DELAY)
if (strcmp((char*)rxData, "Hello") == 0) { }
```

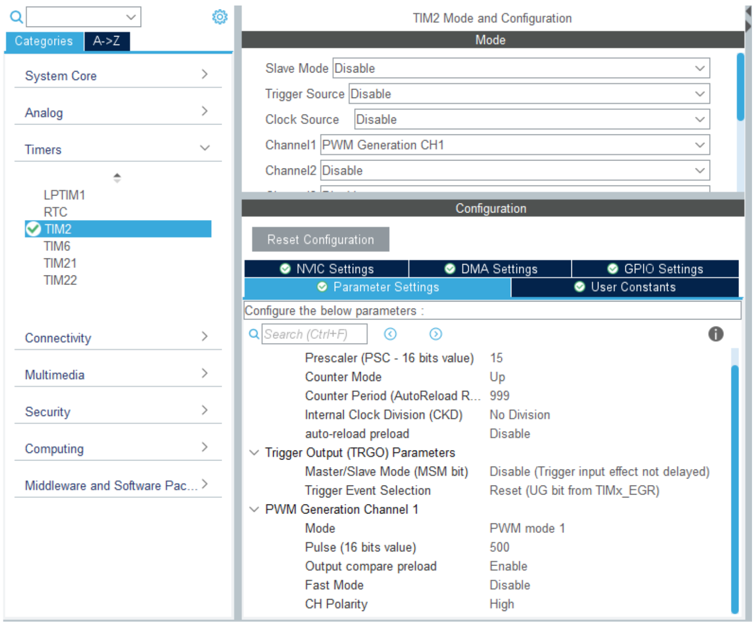
```
uint8_t txData = 0x0B; uint8_t rxData;

HAL_SPI_Transmit(&hspi1, &XXX, 1, HAL_MAX_DELAY)

HAL_SPI_Receive(&hspi1, &XXX, 1, HAL_MAX_DELAY)

HAL_SPI_TransmitReceive (&hspi1, &XXX, &XXX, 1, HAL_MAX_DELAY)

if (rxData[0] == 0x00) { }


void HAL_SPI_(Tx)(RX)(TxRx)CpltCallback(SPI_HandleTypeDef * hspi){ }
```

I2C communication with EEPROM

```
HAL_I2C_Master_Transmit (&hi2c1, Address, Data, Size, HAL_MAX_DELAY);

HAL_I2C_Master_Recieve (&hi2c1, Address, Data, Size, HAL_MAX_DELAY);

HAL_I2C_Slave_Transmit (&hi2c1, Data, Size, HAL_MAX_DELAY);

HAL_I2C_Slave_Recieve (&hi2c1, Data, Size, HAL_MAX_DELAY);

#define EEPROM_ADDR  0xA0

HAL_I2C_Mem_Write (&hi2c1, EEPROM_ADDR, MemAddress, I2C_MEMADD_SIZE_16BIT, Data, Size, HAL_MAX_DELAY);

HAL_I2C_Mem_Read (&hi2c1, EEPROM_ADDR, MemAddress, I2C_MEMADD_SIZE_16BIT, Data, Size, HAL_MAX_DELAY);

while (HAL_I2C_IsDeviceReady(&hi2c1, EEPROM_ADDR, 300, 1000) == HAL_TIMEOUT) {}
```

PWM

```
HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);

__HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, 250);
```



TIM2 Mode and Configuration

Mode

Slave Mode	Disable
Trigger Source	Disable
Clock Source	Disable
Channel1	PWM Generation CH1
Channel2	Disable

Configuration

Configure the below parameters :

Prescaler (PSC - 16 bits value)	15
Counter Mode	Up
Counter Period (AutoReload R...	999
Internal Clock Division (CKD)	No Division
auto-reload preload	Disable
Trigger Output (TRGO) Parameters	
Master/Slave Mode (MSM bit)	Disable (Trigger input effect not delayed)
Trigger Event Selection	Reset (UG bit from TIMx_EGR)
PWM Generation Channel 1	
Mode	PWM mode 1
Pulse (16 bits value)	500
Output compare preload	Enable
Fast Mode	Disable
CH Polarity	High

8-seg LED display

```
#include <stdio.h>

#include <math.h>

static const uint32_t reg_values[10] = {};

void sct_led(uint8_t address, uint8_t data) {

  uint8_t tx[2] = {address, data};

  HAL_GPIO_WritePin(CS_PORT, CS_PIN, GPIO_PIN_RESET);

  HAL_SPI_Transmit(&hspi1, tx, 2, HAL_MAX_DELAY);

  HAL_GPIO_WritePin(CS_PORT, CS_PIN, GPIO_PIN_SET);

}

void sct_value(uint32_t value) {

        uint8_t reg = 0;

        for (uint8_t i = 0; i < 8; i++) {

                reg = reg_values[(value / (int)pow(10, i)) % 10];
```

```
            sct_led(i+1, reg);
        }
}
```

Rotary encoder

TIM: Combined Channels – Encoder mode (Polarity: CH1 – Falling Edge, CH2 – Rising Edge, Counter Period)

```
HAL_TIM_Encoder_Start(&htim2, htim2.Channel);
__HAL_TIM_GET_COUNTER(&htim2);
```

Matrix keyboard

Row1-4: GPIO Output Open Drain

Col1-4: GPIO Input Pull-up

```
static volatile int key = -1;
static const uint32_t code[5] = {7,9,3,2,12};
uint32_t position = 0;
if (key != -1) {
  if (key == code[position] && position == 0){
  position = 1;
  }
}
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
        static int row = 0;
        static const int keyboard[4][4] = {
                        { 1, 2, 3, 21 },
                        { 4, 5, 6, 22 },
                        { 7, 8, 9, 23 },
                        { 11, 0, 12, 24 },
        };
        if (key == -1) {
                if (HAL_GPIO_ReadPin(Col1_GPIO_Port, Col1_Pin) == GPIO_PIN_RESET)
                        key = keyboard[row][0];
                if (HAL_GPIO_ReadPin(Col2_GPIO_Port, Col2_Pin) == GPIO_PIN_RESET)
                        key = keyboard[row][1];
                if (HAL_GPIO_ReadPin(Col3_GPIO_Port, Col3_Pin) == GPIO_PIN_RESET)
                        key = keyboard[row][2];
                if (HAL_GPIO_ReadPin(Col4_GPIO_Port, Col4_Pin) == GPIO_PIN_RESET)
```

```
                    key = keyboard[row][3];
        }
        HAL_GPIO_WritePin(Row1_GPIO_Port, Row1_Pin, GPIO_PIN_SET);
        HAL_GPIO_WritePin(Row2_GPIO_Port, Row2_Pin, GPIO_PIN_SET);
        HAL_GPIO_WritePin(Row3_GPIO_Port, Row3_Pin, GPIO_PIN_SET);
        HAL_GPIO_WritePin(Row4_GPIO_Port, Row4_Pin, GPIO_PIN_SET);
        switch (row) {
        case 0:
              row = 1;
              HAL_GPIO_WritePin(Row2_GPIO_Port, Row2_Pin, GPIO_PIN_RESET);
              break;
        case 1:
              row = 2;
              HAL_GPIO_WritePin(Row3_GPIO_Port, Row3_Pin, GPIO_PIN_RESET);
              break;
        case 2:
              row = 3;
              HAL_GPIO_WritePin(Row4_GPIO_Port, Row4_Pin, GPIO_PIN_RESET);
              break;
        case 3:
              row = 0;
              HAL_GPIO_WritePin(Row1_GPIO_Port, Row1_Pin, GPIO_PIN_RESET);
              break;
        }
}
```