**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);

Obsah obrázku text, snímek obrazovky, software, číslo

Obsah generovaný pomocí AI může být nesprávný.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)

{ }

}

**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);

Obsah obrázku text, snímek obrazovky, software, číslo

Obsah generovaný pomocí AI může být nesprávný.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);

**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;

}

}