

Športna ura

Poročilo za projektno nalogo pri predmetu Vhodno-izhodne
naprave

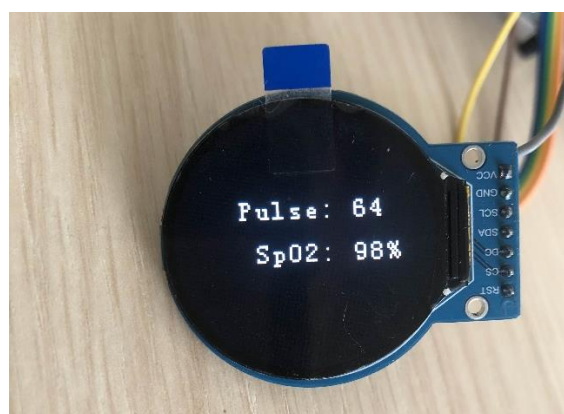
Avtor: Žan Reščič, 63220274; Matija Baša 63220014

Mentor: viš. pred. dr. Robert Rozman

Datum: 12.9.2024

UVOD

Za VIN projekt sva se odločila narediti prototip športne ure, katera s pomočjo pulznega oksimetra meri srčni utrip in kisik v krvi (SpO2) in to prikazuje na zaslon. Uporabila sva mikrokontroler STM32H750B-DK, ploščico Oximeter 5 Click, na kateri je senzor MAX30102, ki vsebuje integriran pulzni oksimeter in TFT LCD zaslon z GC9A01 gonilnikom. Projekt je bil spisan v STM32CubeIDE.



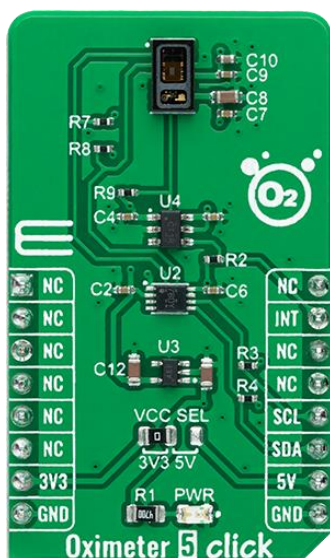
Slike meritev so le simbolične in ne dejansko izmerjene vrednosti z uporabo Oximeter 5 click.


OXIMETER 5 CLICK

PREDSTAVITEV SENZORJEV IN NAPRAV

Oximeter 5 click vsebuje senzor MAX30102, ki je integriran pulzni oksimeter in monitor srčnega utripa. Vključuje notranje LED diode, detektor svetlobe, optične elemente in nizko šumno elektroniko z zavrnitvijo svetlobe okolice. Deluje z enim samim napajalnikom 1,8 V, pridobljenim iz obeh napajalnih pinov mikroBUS-a za notranje LED diode. Komunicira prek standardnega vmesnika, združljivega z I2C.

Deluje tako, da ima MAX30102 integrirane Red in IR (Infra red) LED diode katere oddajajo svetlobo, ki se vpije oz. odbije od kože. Vsebuje tudi detektor svetlobe kateri lovi odbitke svetlobe od kože preko katere se nato izračuna SpO2 in srčni utrip. Kri, ki vsebuje več kisika absorbira več IR svetlobe in odbija več Red svetlobe. Kri, ki vsebuje manj kisika pa absorbira več Red svetlobe in odbija več IR svetlobe. Takšni tehniki pravimo fotopletizmografija (photoplethysmography- PPG).



Notes	Pin					Pin	Notes
	NC	1	AN	PWM	16	NC	
	NC	2	RST	INT	15	INT	Interrupt
	NC	3	CS	RX	14	NC	
	NC	4	SCK	TX	13	NC	
	NC	5	MISO	SCL	12	SCL	I2C Clock
	NC	6	MOSI	SDA	11	SDA	I2C Data
Power Supply	3.3V	7	3.3V	5V	10	5V	Power Supply
Ground	GND	8	GND	GND	9	GND	Ground

KOMUNIKACIJA

Z Oximeter 5 Click-om komuniciramo preko I2C (Inter-Integrated Circuit) protokola. Deluje tako, da uporablja dve žici za komunikacijo in sicer SCL (Serial Clock Line) za časovno usklajevanje in SDL (Serial Data Line) za prenos podatkov. Komunikacija poteka med master napravo in slave slave napravo. V našem primeru je master STM32H750B-KD in slave Oximeter 5 Click.

INICIALIZACIJA

Za inicializacijo uporabljamo metodo **oximeter5_init()**, katera preveri, če je naprava dosegljiva na svojem naslovu (0x57 oz. 0x AE, ko naslov shift-amo za 1 v levo). Če je izpiše, da je naprava pripravljena, drugače izpiše da ni.

```
err_t oximeter5_init ( void )
{
    retval = HAL_I2C_IsDeviceReady(&hi2c4, (OXIMETER5_SET_DEV_ADDR << 1), 3, 100);
    if (retval != HAL_OK)
    {
        snprintf(SendBuffer1, sizeof(SendBuffer1), "Device not ready\n\r");
        HAL_UART_Transmit(&huart3, SendBuffer1, strlen(SendBuffer1), 100);
        return OXIMETER5_ERROR;
    } else {
        snprintf(SendBuffer1, sizeof(SendBuffer1), "Device ready\n\r");
        HAL_UART_Transmit(&huart3, SendBuffer1, strlen(SendBuffer1), 100);
        return OXIMETER5_OK;
    }
}
```

V metodi **oximeter5_default_cfg()** resetiramo programsko opremo, registre in kazalce za fifo buffer, registre prekinitvev, amplitude LED diod. To naredimo tako, da z metodo **oximeter5_generic_write()** zapišemo na naslove želenih registrov določeno vrednost.

```
err_t oximeter5_default_cfg ( void )
{
    uint8_t tmp;

    err_t error_flag = oximeter5_sw_reset();
    HAL_Delay(1000);

    error_flag |= oximeter5_generic_read(OXIMETER5_REG_INTR_STATUS_1, &tmp, 1 );
    HAL_Delay(10);

    tmp = OXIMETER5_SET_INTR_EN_1_FULL_EN;
    tmp |= OXIMETER5_SET_INTR_EN_1_PPG_RDY_EN;
    error_flag |= oximeter5_generic_write(OXIMETER5_REG_INTR_ENABLE_1, &tmp, 1 );
    HAL_Delay(10);

    tmp = OXIMETER5_SET_INTR_EN_2_TEMP_DIS;
    error_flag |= oximeter5_generic_write(OXIMETER5_REG_INTR_ENABLE_2, &tmp, 1 );
    HAL_Delay(10);

    tmp = OXIMETER5_SET_FIFO_PTR_RESET;
    error_flag |= oximeter5_generic_write(OXIMETER5_REG_FIFO_WR_PTR, &tmp, 1 );
    HAL_Delay(10);

    tmp = OXIMETER5_SET_FIFO YYS_COUNTER_RESET;
    error_flag |= oximeter5_generic_write(OXIMETER5_REG_OVF_COUNTER, &tmp, 1 );
    HAL_Delay(10);

    tmp = OXIMETER5_SET_FIFO_PTR_RESET;
    error_flag |= oximeter5_generic_write(OXIMETER5_REG_FIFO_RD_PTR, &tmp, 1 );
    HAL_Delay(10);

    tmp = OXIMETER5_SET_FIFO_CFG_SMP_AVE_3;
    tmp |= OXIMETER5_SET_FIFO_CFG_DATA_SAMP_15;
    error_flag |= oximeter5_generic_write(OXIMETER5_REG_FIFO_CONFIG, &tmp, 1 );
    HAL_Delay(10);

    tmp = OXIMETER5_SET_CFG_MODE_SpO2;
    error_flag |= oximeter5_generic_write(OXIMETER5_REG_MODE_CONFIG, &tmp, 1 );
    HAL_Delay(10);

    tmp = OXIMETER5_SET_SPO2_CFG_ADC_RGE_4096;
    tmp |= OXIMETER5_SET_SPO2_CFG_SR_SEC_100;
    tmp |= OXIMETER5_SET_SPO2_CFG_LED_PW_18_bit;
    error_flag |= oximeter5_generic_write(OXIMETER5_REG_SPO2_CONFIG, &tmp, 1 );
    HAL_Delay(10);

    tmp = OXIMETER5_SET_LED_PULSE_AMPL_7_2_mA;
    error_flag |= oximeter5_generic_write(OXIMETER5_REG_LED1_PA, &tmp, 1 );
    HAL_Delay(10);

    tmp = OXIMETER5_SET_LED_PULSE_AMPL_7_2_mA;
    error_flag |= oximeter5_generic_write(OXIMETER5_REG_LED2_PA, &tmp, 1 );
    HAL_Delay(10);

    uint32_t ir, red;
    error_flag = oximeter5_read_sensor_data(&ir, &red );

    return error_flag;
}
```

BRANJE IN PISANJE

Branje in pisanje je implementirano z uporabo metod **oximeter5_generic_write()** in **oximeter5_generic_read()**. Write deluje tako, da kliče metodo **HAL_I2C_Master_Transmit()**, katera pošilja podatke slave-om. Prejme parametre hi2c4 ročico, naslov Oximetra, podatke za prenos, število podatkov, ki se bo preneslo in časovno omejitev.

```
err_t oximeter5_generic_write ( uint8_t reg, uint8_t *tx_buf, uint8_t tx_len )
{
    uint8_t data_buf[ 257 ] = { 0 };

    data_buf[ 0 ] = reg;

    for ( uint8_t cnt = 1; cnt <= tx_len; cnt++ )
    {
        data_buf[ cnt ] = tx_buf[ cnt - 1 ];
    }
    retval = HAL_I2C_Master_Transmit(&hi2c4, (OXIMETERS5_SET_DEV_ADDR << 1), data_buf, tx_len+1, 1000);
    if (retval != HAL_OK)
    {
        snprintf(SendBuffer1, sizeof(SendBuffer1), "Transmit failed\n\r");
        //HAL_UART_Transmit(&huart3, SendBuffer1, strlen(SendBuffer1), 100);
        return OXIMETERS5_ERROR;
    } else {
        snprintf(SendBuffer1, sizeof(SendBuffer1), "Transmit successful\n\r");
        //HAL_UART_Transmit(&huart3, SendBuffer1, strlen(SendBuffer1), 100);
        return OXIMETERS5_OK;
    }
    //return i2c_master_write( &ctx->i2c, data_buf, tx_len + 1 );
}
```

Read deluje podobno, le da prvo kliče write, da pove določenemu registru, da bo iz njega brala podatke in nato kliče metodo **HAL_I2C_Master_Receive()** katera prejme podatke. Parametri metode so enaki kot pri write, le da tukaj se podatki sprejemajo v write pa pošiljajo.

```
err_t oximeter5_generic_read ( uint8_t reg, uint8_t *rx_buf, uint8_t rx_len )
{
    if (oximeter5_generic_write(reg, rx_buf, rx_len) == OXIMETERS5_OK) {
        retval = HAL_I2C_Master_Receive(&hi2c4, (OXIMETERS5_SET_DEV_ADDR << 1),rx_buf, rx_len, 1000);
        if (retval != HAL_OK)
        {
            snprintf(SendBuffer1, sizeof(SendBuffer1), "Receive failed\n\r");
            HAL_UART_Transmit(&huart3, SendBuffer1, strlen(SendBuffer1), 100);
            return OXIMETERS5_ERROR;
        } else {
            snprintf(SendBuffer1, sizeof(SendBuffer1), "Receive successful\n\r");
            HAL_UART_Transmit(&huart3, SendBuffer1, strlen(SendBuffer1), 100);
            return OXIMETERS5_OK;
        }
    }
    snprintf(SendBuffer1, sizeof(SendBuffer1), "Write before read failed\n\r");
    HAL_UART_Transmit(&huart3, SendBuffer1, strlen(SendBuffer1), 100);
    return OXIMETERS5_ERROR;
    //return i2c_master_write_then_read( &ctx->i2c, &reg, 1, rx_buf, rx_len );
}
```

MERJENJE SpO2 IN SRČNEGA UTRIPA

Za merjenje najprej kličemo metodo `oximeter5_read_sensor_data()`, katera iz senzorja bere z hitrostjo 25 smp/s (samples/seconds) štiri sekunde, tako da prejme 100 podatkov. Podatki se hranijo v FIFO bufferjih kar pomeni, da je hranjenim samo prvih 100 podatkov. Imamo dva bufferja in sicer RED buffer in IR buffer.

```
err_t oximeter5_read_sensor_data ( uint32_t *ir, uint32_t *red )
{
    uint8_t rx_buf[ 6 ];

    err_t error_flag = oximeter5_generic_read(OXIMETER5_REG_FIFO_DATA, rx_buf, 6 );

    *ir = rx_buf[ 0 ];
    *ir <<= 8;
    *ir |= rx_buf[ 1 ];
    *ir <<= 8;
    *ir |= rx_buf[ 2 ];
    *ir &= DATA_18_BIT;

    *red = rx_buf[ 3 ];
    *red <<= 8;
    *red |= rx_buf[ 4 ];
    *red <<= 8;
    *red |= rx_buf[ 5 ];
    *red &= DATA_18_BIT;

    return error_flag;
}
```

Ko imamo podatke v bufferjih jih lahko začnemo procesirati za SpO2 in srčni utrip. Za meritev SpO2 kličemo metodo `oximeter5_get_oxygen_saturation()`. Metoda podatke v bufferjih procesira s pristopom digitalnega procesiranja signalov (DPS). Odstraniti mora vrhove v signalu kateri so preveč blizu, odstraniti mora prenizke vrhove kateri bi pokvarili rezultat in jih sortirati (koda metode je na github repozitoriju, ker je preveč dolga).

```
err_t oximeter5_get_oxygen_saturation ( uint32_t *pun_ir_buffer, int32_t n_ir_buffer_length, uint32_t *pun_red_buffer, uint8_t *pn_spo2 )
{
    uint32_t un_ir_mean;
    int32_t n_i_ratio_count;
    int32_t n_exact_ir_valley_locs_count, n_middle_idx;
    int32_t n_th1, n_npk;
    int32_t an_ir_valley_locs[ 15 ];
    int32_t n_y_ac, n_x_ac;
    int32_t n_spo2_calc;
    int32_t n_y_dc_max, n_x_dc_max;
    int32_t n_y_dc_max_idx, n_x_dc_max_idx;
    int32_t an_ratio[ 5 ], n_ratio_average;
    int32_t n_num, n_denom ;
    int32_t an_x[ BUFFER_SIZE ];
    int32_t an_y[ BUFFER_SIZE ];
    err_t error_flag;

    un_ir_mean = 0;

    for ( int32_t n_cnt_k = 0; n_cnt_k < n_ir_buffer_length; n_cnt_k++ )
    {
        un_ir_mean += pun_ir_buffer[ n_cnt_k ];
    }
}
```

Za meritev srčnega utripa uporabljamo metodo **oximeter5_get_heart_rate()**. Metoda najprej podatke v bufferjih pripravi za procesiranje in nato jih s pristopom digitalnega procesiranja signalov pripravi v sprejemljive za izračun srčnega utripa (koda metode je na github repozitoriju, ker je preveč dolga).

```
err_t oximeter5_get_heart_rate ( uint32_t *pun_ir_buffer, int32_t n_ir_buffer_length, uint32_t *pun_red_buffer, int32_t *pn_heart_rate )
{
    uint32_t un_ir_mean;
    int32_t n_th1, n_npk;
    int32_t an_ir_valley_locs[ 15 ];
    int32_t n_peak_interval_sum;
    int32_t an_x[ BUFFER_SIZE ];
    err_t error_flag;

    // calculates DC mean and subtract DC from ir
    un_ir_mean = 0;
    for ( int32_t n_cnt_k = 0; n_cnt_k < n_ir_buffer_length; n_cnt_k++ )
    {
        un_ir_mean += pun_ir_buffer[ n_cnt_k ];
    }

    un_ir_mean = un_ir_mean / n_ir_buffer_length;

    // remove DC and invert signal so that we can use peak detector as valley detector
    for ( int32_t n_cnt_k = 0; n_cnt_k < n_ir_buffer_length; n_cnt_k++ )
    {
        an_x[ n_cnt_k ] = -1 * ( pun_ir_buffer[ n_cnt_k ] - un_ir_mean );
    }

    // 4 pt Moving Average
    for( int32_t n_cnt_k = 0; n_cnt_k < BUFFER_SIZE - MA4_SIZE; n_cnt_k++ )
    {
        an_x[ n_cnt_k ]=( an_x[ n_cnt_k ] + an_x[ n_cnt_k + 1 ] + an_x[ n_cnt_k + 2 ] + an_x[ n_cnt_k + 3 ] ) / 4;
    }

    // calculate threshold
    n_th1 = 0;
    for ( int32_t n_cnt_k = 0; n_cnt_k < BUFFER_SIZE; n_cnt_k++ )
    {
        n_th1 += an_x[ n_cnt_k ];
    }
}
```


POMOŽNE METODE

Metode, katere smo uporabili za procesiranje surovega signala so **dev_find_peaks()**, katere najde prave vrhove v signalu. Pomožni metodi, ki jih uporabi sta **dev_peaks_above_min_height()** in **dev_remove_close_peaks()**. Prva odstrani vrhove, kateri so prenizki za pravilen izračun tako SpO2, kot srčnega utripa.

```
static void dev_peaks_above_min_height ( int32_t *pn_locs, int32_t *n_npk,  int32_t  *pn_x, uint8_t n_size, int32_t n_min_height )
{
    uint8_t n_width;
    uint8_t n_cnt = 1;

    *n_npk = 0;

    while ( n_cnt < ( n_size - 1 ) )
    {
        if ( pn_x[ n_cnt ] > n_min_height && pn_x[ n_cnt ] > pn_x[ n_cnt - 1 ] )
        {
            n_width = 1;

            while ( n_cnt + n_width < n_size && pn_x[ n_cnt ] == pn_x[ n_cnt + n_width ] )
            {
                n_width++;
            }

            if ( pn_x[ n_cnt ] > pn_x[ n_cnt + n_width ] && ( *n_npk ) < 15 )
            {
                pn_locs[( *n_npk )++ ] = n_cnt;
                n_cnt += n_width + 1;
            }
            else
            {
                n_cnt += n_width;
            }
        }
        else
        {
            n_cnt++;
        }
    }
}
```

Drugo metodo, katero uporabi odstrani vrhove signala, kateri so si med seboj preblizu.

```
static void dev_remove_close_peaks ( int32_t *pn_locs, int32_t *pn_npk, int32_t *pn_x, int32_t n_min_distance )
{
    int32_t n_old_npk, n_dist;

    dev_sort_indices_descend( pn_x, pn_locs, *pn_npk );

    for ( int32_t n_cnt_i = -1; n_cnt_i < *pn_npk; n_cnt_i++ )
    {
        n_old_npk = *pn_npk;
        *pn_npk = n_cnt_i + 1;

        for ( int32_t n_cnt_j = n_cnt_i + 1; n_cnt_j < n_old_npk; n_cnt_j++ )
        {
            n_dist = pn_locs[ n_cnt_j ] - ( n_cnt_i == -1 ? -1 : pn_locs[ n_cnt_i ] );

            if ( n_dist > n_min_distance || n_dist < -n_min_distance )
            {
                pn_locs[ (*pn_npk)++ ] = pn_locs[ n_cnt_j ];
            }
        }
    }

    dev_sort_ascend( pn_locs, *pn_npk );
}
```

Uporabljene so še sortirne metode, katere uredijo podatke naraščajoče oz. padajoče.

```
static void dev_sort_indices_descend ( int32_t *pn_x, int32_t *pn_indx, int32_t n_size )
{
    int32_t n_temp;

    for ( int32_t n_cnt_i = 1; n_cnt_i < n_size; n_cnt_i++ )
    {
        n_temp = pn_indx[ n_cnt_i ];

        int32_t n_cnt_j;
        for ( n_cnt_j = n_cnt_i; n_cnt_j > 0 && pn_x[ n_temp ] > pn_x[ pn_indx[ n_cnt_j - 1 ] ]; n_cnt_j-- )
        {
            pn_indx[ n_cnt_j ] = pn_indx[ n_cnt_j - 1 ];
        }

        pn_indx[ n_cnt_j ] = n_temp;
    }
}

static void dev_sort_ascend ( int32_t *pn_x, int32_t n_size )
{
    int32_t n_temp;

    for ( int32_t n_cnt_i = 1; n_cnt_i < n_size; n_cnt_i++ )
    {
        n_temp = pn_x[ n_cnt_i ];

        int32_t n_cnt_j;
        for ( n_cnt_j = n_cnt_i; n_cnt_j > 0 && n_temp < pn_x[ n_cnt_j - 1 ]; n_cnt_j-- )
        {
            pn_x[ n_cnt_j ] = pn_x[ n_cnt_j - 1 ];
        }

        pn_x[ n_cnt_j ] = n_temp;
    }
}
```

LCD ZASLON

OPIS NAPRAVE

Naprava, na katero bomo izpisovali rezultate je okrogel LCD zaslon, ki ima naslednje specifikacije:

- Velikost 1.28 palčni oziroma 3.25cm
- Ločljivost: 240x240
- IC: GC9A01

POVEZAVA

Naprava ima 6 priključitvenih pinov in sicer:

- VCC, povežemo ga na napajanje 3.3V
- GND, povežemo ga na ozemljitveni pin na naši napravi
- SCL, povežemo ga na urin signal
- SDA, povežemo na master out slave in
- DC, povežemo na GPIO output
- CS, povežemo na GPIO output
- RST, povežemo na GPIO output

INICIALIZACIJA NA STM32H7

Za upravljanje zaslona sem uporabil STM32H7 board. Za komunikacijo bomo uporabili SPI. Zaslon na STM23H7 povežemo na naslednji način za privzeto inicializiran SPI2:

- VCC -> 3v3 pin na H7
- GND -> GND pin na H7
- SCL -> PD3 pin na H7 (SCK/D13)
- SDA -> PB15 pin na H7 (PWM/MOSI/D11)
- DC -> katerikoli prosti GPIO pin, jaz sem izbral PE6
- CS -> katerikoli prosti GPIO pin, jaz sem izbral PI8
- RST -> katerikoli prosti GPIO pin, jaz sem izbral PE3

Ko sem imel napravo uspešno povezano z kabli sem moral še projekt inicializirati v .ioc datoteki v CubeIDE. Najprej sem inicializiral PGIO pine potrebne za komunikacijo. V pinout sem inicializiral 3 GPIO pine kot GPIO_Output in sicer DC, CS, RST. Poimenoval sem jih LCD_DC_R, LCD_CS_R, LCD_RST_R. Poimenovanje sicer ni tako pomembno mi je pa bilo lažje potem v kodi da sem vedel kateri pin je kateri. Nato sem pod Connectivity zavihkom izbral SPI2. Tukaj sem najprej spremenil data size iz 4b na 8b. Nato sem moral še zmanjšati boud rate, saj je bil privzeto zelo previsok. Zmanjšal sem ga na 12Mbits/s. Kot zadnje sem inicializiral še DMA. Dodal sem SPI2_RX kot SPI2_TX. DMA sicer nebi bil obvezen za upravljanje naprave, ampak bi bila komunikacija bistveno prepočasna za uporabo zaslona.

IZPISOVANJE NA ZASLON

Za izpisovanje sem si pomagal z minimalnim driverjem za GC9A01. Za začetek je potrebno dodati pine, ki smo jih inicializirali v .ioc datoteki. To naredimo na naslednji način:

```
#define LCD_RST_1 HAL_GPIO_WritePin(LCD_RST_R_GPIO_Port,LCD_RST_R_Pin,GPIO_PIN_SET)
#define LCD_RST_0 HAL_GPIO_WritePin(LCD_RST_R_GPIO_Port,LCD_RST_R_Pin,GPIO_PIN_RESET)

#define LCD_CS_1 HAL_GPIO_WritePin(LCD_CS_R_GPIO_Port,LCD_CS_R_Pin,GPIO_PIN_SET)
#define LCD_CS_0 HAL_GPIO_WritePin(LCD_CS_R_GPIO_Port,LCD_CS_R_Pin,GPIO_PIN_RESET)

#define LCD_DC_1 HAL_GPIO_WritePin(LCD_DC_R_GPIO_Port,LCD_DC_R_Pin,GPIO_PIN_SET)
#define LCD_DC_0 HAL_GPIO_WritePin(LCD_DC_R_GPIO_Port,LCD_DC_R_Pin,GPIO_PIN_RESET)
```

Tukaj imamo nizka in visoka stanja za vse 3 pine. Imena so tukaj ista kot v pinout.

Za pisanje uporabimo naslednjo funkcijo:

```
HAL_StatusTypeDef SPI_Write(uint8_t* pbuff, uint16_t size){

    HAL_StatusTypeDef status = HAL_SPI_Transmit_DMA(&hspi2, pbuff, size);

    while(HAL_SPI_GetState(&hspi2) != HAL_SPI_STATE_READY){;}
    return status;

}
```

Za zapis uporabimo HAL_SPI_Transmit_DMA() jas je tako najhitrejše. Zanko na koncu potrebujemo zato, da vemo kdaj je SPI pripravljen za naslednji zapis.

Za izbiro naslovnega okna oziroma izbire na kateri del zaslona bomo risali uporabimo naslednjo funkcijo:

```
void LCD_SetPos(unsigned int Xstart,unsigned int Ystart,unsigned int Xend,unsigned int Yend)
{
    Write_Cmd(0x2a);
    Write_Cmd_Data(Xstart>>8);
    Write_Cmd_Data(Xstart);
    Write_Cmd_Data(Xend>>8);
    Write_Cmd_Data(Xend);

    Write_Cmd(0x2b);
    Write_Cmd_Data(Ystart>>8);
    Write_Cmd_Data(Ystart);
    Write_Cmd_Data(Yend>>8);
    Write_Cmd_Data(Yend);

    Write_Cmd(0x2c);
}
```

Write_Cmd(0x2a) in Write_Cmd(0x2b), ki jim potem sledijo podatki za začetek in konec nam x in y koordinate okna. Nato pa imamo še Write_Cmd(0x2c), ki nam pove, da bomo od sedaj naprej pisali pixle na zaslon, se pravi vsi podatki, ki jih bomo poslali bodo interpretirani kot podatki za prižiganje pixlov na območju, ki smo ga definirali.

Za dejansko prižiganje pixlov pa uporabimo naslednje funkcije:

```
void Write_Data_U16(unsigned int y)
{
    unsigned char m,n;
    m=y>>8;
    n=y;
    Write_Data(m,n);
}

void Write_Data(unsigned char DH,unsigned char DL)
{
    LCD_CS_0;
    LCD_DC_1;

    SPI_Write(&DH, 1);
    SPI_Write(&DL, 1);

    LCD_CS_1;
}
```

Funkcija Write_Data_U16() nam razdeli 16bitno integer število na dva 8bitnadela in jih pošlje v funkcijo Write_Data(). Tukaj najprej chip select nastavimo na nizko stanje in LCD_DC na visoko kar določi, da bomo pošiljali podatke in ne ukaze. Nate z prejšnje implementirano funkcijo pošljemo oba dela in postavimo CS na visoko stanje.

Sedaj če bi želeli celoten zaslon spremeniti v eno barvo bi uporabili naslednjo kodo:

```
void ClearScreen(unsigned int bColor)
{
    unsigned int i,j;
    LCD_SetPos(0,0,GC9A01_TFTWIDTH-1,GC9A01_TFTHEIGHT-1); //240x240
    for (i=0;i<GC9A01_TFTWIDTH;i++)
    {
        for (j=0;j<GC9A01_TFTHEIGHT;j++)
            Write_Data_U16(bColor);
    }
}
```

Parameter, ki ga funkcija dobi je barva, ki je v formatu RGB565 (5bit rdeča, 6bit zelena in 5bit modra). Ta funkcija sicer deluje, je pa počasna, saj moramo vsak pixel posebej nastaviti.

To bi pohitрили z DMA in sicer da namesto vsakega pixla posebej pošljemo cel buffer.

```
void Write_Bytes(unsigned char * pbuff, unsigned short size)
{
    LCD_CS_0;
    LCD_DC_1;

    SPI_Write(pbuff, size);

    LCD_CS_1;
}
```

To funkcijo pa lahko kličemo na naslednji način:

```
void ClearWindow(unsigned int startX, unsigned int startY, unsigned int endX, unsigned int endY, unsigned int bColor)
{
    unsigned int i;

    unsigned char hb = (bColor & 0xFFFF) >> 8;
    unsigned char lb = bColor & 0xFF;
    unsigned short tempColor = lb * 256 + hb;

    unsigned int totalSize = (endX - startX) * (endY - startY) * 2;
    unsigned int bufSize = 512;

    unsigned int loopNum = (totalSize - (totalSize % bufSize)) / bufSize;
    unsigned int modNum = totalSize % bufSize;

    unsigned short tempBuf[bufSize];
    unsigned char * ptempBuf;

    for(i=0; i<bufSize; i++){
        tempBuf[i] = tempColor;
    }

    LCD_SetPos(startX, startY, endX-1, endY-1);

    ptempBuf = (unsigned char *)tempBuf;
    for(i=0; i<loopNum; i++){
        Write_Bytes(ptempBuf, bufSize);
    }
    Write_Bytes(ptempBuf, modNum);
}
```

Ta funkcija bistveno pohitri delovanje, saj ne pošiljamo vseh 240*240 pixlov, vendar pošiljamo po kosih velikih 512 bajtov. Na tak način lahko na zaslon izpisujemo kar želimo, le da je ta način bistveno hitrejši.

Ko imamo vse pripravljeno, lahko te funkcije kličemo na naslednji način:

```
GC9A01_Initial();  
ClearScreen2(0x0000);
```

Prva funkcija inicializira zaslon, da je pripravljen za komunikacijo, druga pa je funkcija, ki kot zgoraj nastavi cel zaslon na eno barvo, v tem primeru bela.

GITHUB

Na naslednji povezavi se nahaja naše celoten projekt, ki vključuje datoteke oximeter5custom.c, ki vključuje vso kodo povezano z Oximeter 5 click. In gc9a01.c, ki vključuje vso kodo povezano z LCD zaslonom.

https://github.com/ZanRescic/VIN_Oximeter_5_Click