Univerza *v Ljubljani*
Fakulteta *za računalništvo*
in informatiko

Zračna miška

Poročilo za projekt pri predmetu vhodno-izhodne naprave

Avtor: Bernard Kuchler, 63200160

Mentor: viš. pred. dr. Robert Rozman

Kazalo

Povzetek

Naslov: Zračna miška

Avtor: Bernard Kuchler

Pri predmetu Vhodno-izhodne naprave sem za projekt izdelal zračno miško z uporabo STM32F407G-DISC1 mikrokontrolerja in STEVAL-MKI197V1 plošče, na katerem se nahaja merilnik pospeška in žiroskop. S pregledom podatkovnih listin podanih senzorjev in mikrokontrolerja ter podobnih projektov, sem izdelal 2 obliki zračne miške. Pri prvi obliki se uporablja le STM32F407G-DISC1, ki ima že vgrajen merilnik pospeška, s pomočjo katerega se ob nagibih mikrokontrolerja nadzira premike kazalca miške na zaslonu. Pri drugi obliki pa sem dodatno priključil omenjeno ploščo s senzorji in nadziral kazalec miške z uporabo žiroskopa, kar je omogočalo premike glede na usmeritev senzorja. Obe zračni miški lahko izvedeta pritisk levega miškinega gumba s pritiskom na gumb, ki se nahaja na plošči mikrokontrolerja.

Ključne besede: zračna miška, STM32F407G-DISC1, STM32CubeIDE, programski jezik C, LSM6DSOX, STEVAL-MKI197V1.

Abstract

Title: Air mouse

Author: Bernard Kuchler

For my project, for the subject of Vhodno-izhodne naprave (Input-output devices), i have designed an air mouse using the STM32F407G-DISC1 microcontroller and the STEVAL-MKI197V1 board, which, amongst other sensors, houses an accelerometer and gyroscope. By reviewing the datasheets of the given sensors and microcontroller and analyzing similar projects, I made 2 versions of an air mouse. In the first version, only the STM32F407G-DISC1 is used, which already has a built-in accelerometer, used to control the movements of the mouse cursor on the computer screen when the microcontroller is tilted. In the second version, I additionally connected the mentioned sensor board and controlled the mouse cursor using a gyroscope, which allowed cursor movement according to the orientation of the sensor. Both air mice are capable of simulating the left mouse button click by pressing a button located on the microcontroller board.

Keywords: air mouse, STM32F407G-DISC1, STM32CubeIDE, programming language C, LSM6DSOX, STEVAL-MKI197V1.

1. Uvod

Za projekt pri predmetu vhodno-izhodne naprave, sem želel izdelati zračno miško. V poročilu sem opisal uporabljene komponente, povezave senzorjev s mikrokrmilnikom, napisano kodo in nastali obliki zračne miške ter njuno uporabo. Prvo opisana oblika zračne miške deluje že samo s ploščo STM32F407G-DISC1, ki ima vgrajen merilnik pospeška. Za drugo opisano obliko zračne miške, pa je potrebno še dodatno na omenjeno ploščo povezati STEVAL-MKI197V1 ploščo na kateri se nahaja, poleg drugih senzorjev, žiroskop. Obe zračni miški sta delujoči, vendar se slednjo obliko nekoliko lažje in bolj natančno uporablja ter je njena uporaba tudi bolj podobna zračnim miškam na trgu.

2. Zračna miška upravljana preko merilnika pospeška

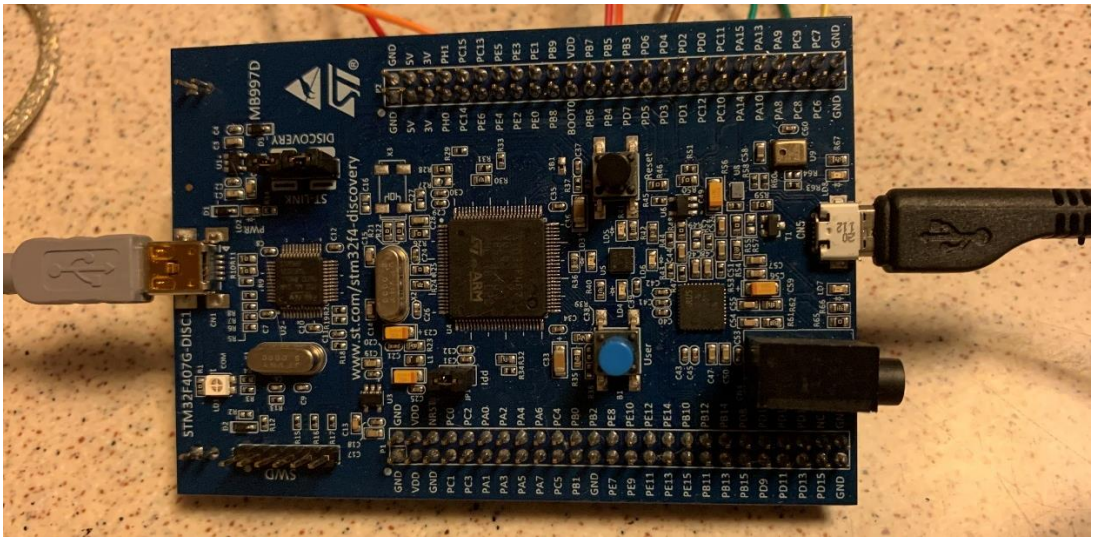
2.1 Opis delovanja in uporabe

Prva oblika zračne miške deluje preko uporabe merilnika pospeška LIS3DSH, ki je vgrajen na ploščo STM32F407G-DISC1 in je tako že povezan na mikrokrmilnik. Senzor sporoča mikrokrmilniku svoje meritve preko SPI komunikacije, mikrokrmilnik pa dobljene meritve ustrezno obdela v obliko podatkov, kot jih pošilja računalniška miška in jih nato pošlje preko USB povezave na računalnik. Operacijski sistem računalnika nato sam prepozna povezano ploščo kot računalniško miško in podatke, ki se pošiljajo v ustrezni obliki, uporablja za nadzorovanje kazalca miške na zaslonu računalnika.

Pri uporabi zračne miške, jo je najprej potrebno povezati z računalnikom preko priključkov na plošči, ki se povežeta z dvema USB priključkoma na računalniku. Za povezavo je potrebno uporabiti USB tip A do mini B kabla in USB tip A do mikro B kabla. Ob priključitvi plošče z uporabo kablov ter tudi ob pritisku na črno tipko Reset, je najprej potrebno počakati približno 6 sekund, da se nastavi senzor in izvede kalibracija. V tem času je potrebno držati ploščo v položaju, v katerem želimo uporabljati zračno miško. Izbran položaj bo nastavljen kot osnova in glede na nagibe plošče iz osnovnega položaja, se bo upravljalo s položajem kazalca miške na zaslonu računalnika. Po izvedeni kalibraciji se na plošči prižge modra LED luč, kar pomeni, da je zračna miška pripravljena na uporabo. Sedaj lahko nadzorujemo gibanje kazalca miške na zaslonu računalnika. Ob pritisku modrega gumba User, ki se nahaja na plošči, se izvede pritisk levega miškinega gumba na trenutnem položaju kazalca miške.

2.2 Komponente

Prva oblika zračne miške je sestavljena iz: STM32F407G-DISC1 plošče z mikrokontrolerjem, USB tip A do mini B kabla in USB tip A do mikro B kabla. Prav tako se uporablja merilnik pospeška LIS3DSH, ki je vgrajen na omenjeno ploščo. Omenjena kabla se povežeta preko ustreznih USB mini B in mikro B priključkov na plošči ter USB tip A priključkov na izbranem računalniku.



Slika 1: STM32F407G-DISC1

2.3 Koda

Koda je bila napisana v STM32CubeIDE. Ob izdelavi novega STM32 projekta (imenovanega STM32_F407G_MISKA_pospesevalniki), sem izbral ploščo STM32F407G-DISC1, zaradi česar mi je IDE že izdelal ioc datoteko, z vsemi vhodi in izhodi, ki je namenjena za to ploščo. Najprej sem se v nastali datoteki STM32_F407G_MISKA_pospesevalniki.ioc, s pomočjo CubeMX, premaknil v razdelek Pinout & Configuration ter izbral zavihek A -> Z in nastavil USB_OTG_FS na nastavitev Device_Only, kar pomeni da se bo plošča priključila kot USB naprava na računalnik. Pod USB_DEVICE sem nastavitvi Class For FS IP izbral vrednost Human Interface Device Class (HID), kar pomeni da bo plošča komunicirala z povezanim računalnikom kot USB naprava (v tem primeru kot računalniška miška), prav tako pa sem v razdelku Device descriptor za vrednost PRODUCT_STRING vpisal STM32F407G MISKA. Vpisano ime se bo uporabljalo na povezanem računalniku za poimenovanje zračne miške. Za komunikacijo med mikrokontrolerjem in senzorjem LIS3DSH, ki je vgrajen na ploščo, sem uporabil SPI (sinhron protokol za komunikacijo). Izbral sem SPI1 in v razdelku Parameter Settings nastavil Prescaler (for Baud Rate) na vrednost 256, hitrost se tako deli z 256 in je zaradi tega pod vrednostjo 1 Mhz. Pri RCC je potrebno nastaviti uro na vrednost Crystal/Ceramic Resonator, v SYS pa za nastavitev Debug vrednost nastaviti Serial Wire. V oknu Pinout view, sem nastavil priključek PA5 na vrednost SPI1_SCK (preko tega priključka se pošilja urin signal), PA6 na SPI1_MISO (mikrokontroler preko tega priključka sprejema podatke kot gospodar, senzor pa pošilja izbrane meritve kot suženj), PA7 na SPI1_MOSI (mikrokontroler preko tega priključka pošilja podatke ali ukaze kot gospodar, senzor pa jih sprejema kot suženj) in PE3 na GPIO_Output (preko tega priključka se sporoči pričetek SPI komunikacije, ko se pošlje logična vrednost 0 in konec ob logični vrednosti 1), te nastavitve priključkov so potrebne za SPI komunikacijo s senzorjem LIS3DSH in so rezervirane za ta namen, kot je opisano v podatkovni listini STM32F407G-DISC1 na straneh 21, 28 in 36 [1]. Način SPI komunikacije za branje in pisanje v registre je opisan podatkovni listini LIS3DSH na straneh 26 in 27 [2]. Priključek PD15 je nastavljen na GPIO_Output (izbrana vrednost priključka je rezervirana za komunikacijo z modro LED lučko na plošči), priključek PA0-WKUP pa na GPIO_EXTI0 (izbrana vrednost priključka je rezervirana za komunikacijo z modrim User gumbom na plošči), tudi te nastavitvi za uporabo modre LED lučke in modrega User gumba sta opisani v podatkovni listini STM32F407G-DISC1 na straneh 20 in 28 [1]. V nastavitvah za GPIO sem nato za priključek PA0-WKUP nastavil vrednost za GPIO mode na External Interrupt Mode with Rising edge trigger detection in GPIO Pull-up/Pull-down na vrednost Pull-down. Znotraj GPIO sem v zavihku NVIC, vključil EXTI line0 interrupt, zaradi česar se bo, ob pritisku modrega gumba User, sprožila prekinitev. S pritiskom na gumb Build project, ki se nahaja v izvlečnem meniju Project, sem generiral izbrane nastavitve v kodo.

Kodo sem pisal v programskem jeziku C znotraj datoteke main.c, ki se je zgenerirala iz opisanih nastavitev. Poleg že dodanih knjižnic sem dodal še knjižnico usbd_hid.h, za komunikacijo z računalnikom na način kot bi podatke pošiljala računalniška miška.

```
/* Includes ------------------------
#include "main.h"
#include "i2c.h"
#include "i2s.h"
#include "spi.h"
#include "usb_device.h"
#include "gpio.h"

/* Private includes -----------------
/* USER CODE BEGIN Includes */
#include "usbd_hid.h"
/* USER CODE END Includes */
```

Slika 2: Knjižnice dodane v kodi

Nato sem definiral spremenljivke. Spremenljivki indata in outdata, se bosta uporabili za branje in pisanje vrednosti v registre senzorja. V lis_id se bo prebrala identifikacijska številka senzorja. V AccelX, AccelY in AccelZ se bodo hranile meritve pridobljene iz senzorja.

Najmanjše in največje vrednosti glede na x in y os, hranjene v spremenljivkah min_xval, max_xval, min_yval in max_yval, se bodo spremenile po kalibraciji zračne miške. V newxval in newyval se bodo hranile izračunane vrednosti za katere se bo kazalec miške premaknil, glede na os. Spremenljivka button_flag bo hranila stanje modrega gumba User.

```
// Global variables
uint8_t indata[2];
uint8_t outdata[2] = {0,0};
uint8_t lis_id;
int8_t AccelX;
int8_t AccelY;
int8_t AccelZ;
HAL_StatusTypeDef SPIStatus;
```

Slika 3: 1. definiranje spremenljivk

```
/* USER CODE BEGIN 0 */

int16_t min_xval = 128;
int16_t max_xval = -128;
int16_t min_yval = 128;
int16_t max_yval = -128;

int16_t newxval = 0;
int16_t newyval = 0;

uint8_t button_flag = 0;
```

Slika 4: 2. definiranje spremenljivk

Iz datoteke usb_device.c, ki se nahaja znotraj projekta v računalniški mapi USB_device in podmapi App, sem prilepil USBF_HandleTypeDef, ki definira vrsto podatkov, ki se bodo pošiljali na računalnik. Definiral sem strukturo mouseHID, v kateri se nahajajo posamezni podatki, katere pošilja miška na računalnik v ustreznem vrstem redu. Podatki znotraj strukture so: stanje gumba (ali se je pritisnil ali spustil levi miški gumb označen z vrednostjo 1 ali pa desni miškin gumb označen z vrednostjo 2),

položaj kazalca miške glede na x os, položaj kazalca miške glede na y os in vrednost miškinega kolesa. Vse vrednosti so na začetku nastavljene na 0.

```
// MOUSE
extern USBD_HandleTypeDef hUsbDeviceFS;

typedef struct
{
    uint8_t button;
    int8_t mouse_x;
    int8_t mouse_y;
    int8_t wheel;
} mouseHID;

mouseHID mousehid = {0,0,0,0};
```

Slika 5: Struktura uporabljena za ustrezni vrstni red podatkov, ki se pošiljajo na računalnik

S funkcijo Calibrate, se na začetku izvajanja kode v 50 ponovitvah zanke, kar skupno traja približno 5 sekund, nastavijo najmanjše in največje vrednosti meritev glede na x in y os. V vsaki ponovitvi zanke se preberejo meritve senzorja za posamezne osi in primerjajo s trenutno shranjeno največjo in najmanjšo vrednostjo te osi. V kolikor je nova vrednost manjša od trenutno najmnjše ali pa večja od trenutno največje, se jo shrani kot novo vrednost v ustrezno spremenljivko. Na koncu vsake ponovitve zanke, program počaka 100 milisekund. Ko se funkcija izvede do konca, se prižge modra LED lučka na ploščici in tako nakaže, da je zračna miška pripravljena na uporabo.

```
void Calibrate (void) //Calibrates values from sensor. It is used to move the
{
    for (int i=0; i<50; i++)
    {

        outdata[0] = 0x29 | 0x80 ; // read x
        HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_RESET);
        HAL_SPI_TransmitReceive(&hspi1, &outdata, &indata, 2, HAL_MAX_DELAY);
        // HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_SET);
        AccelX = indata[1];
        outdata[0] = 0x2B | 0x80 ; // read y
        // HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_RESET);
        HAL_SPI_TransmitReceive(&hspi1, &outdata, &indata, 2, HAL_MAX_DELAY);
        // HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_SET);
        AccelY = indata[1];
        outdata[0] = 0x2D | 0x80 ; // read z
        // HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_RESET);
        HAL_SPI_TransmitReceive(&hspi1, &outdata, &indata, 2, HAL_MAX_DELAY);
        HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_SET);
        AccelZ = indata[1];

        min_xval = MIN(min_xval, AccelX);
        max_xval = MAX(max_xval, AccelX);
        min_yval = MIN(min_yval, AccelY);
        max_yval = MAX(max_yval, AccelY);
        HAL_Delay (100);
    }

    // Turns the blue LED on to indicate the completion of calibration
    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, 1);
}
```

Slika 6: Funkcija za kalibracija vrednosti meritev

V funkciji HAL_GPIO_EXTI_Callback, se opazuje zunanje prekinitve in v kolikor je prekinitev sprožil pritisk modrega gumba User, ki se nahaja na priključku GPIO_PIN_0, se bo v spremenljivko button_flag zapisala logična vrednost 1.

```
137 void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
138 {
139     if (GPIO_Pin == GPIO_PIN_0) //Check if the b
140     {
141         button_flag = 1;
142     }
143 }
```

Slika 7: Funkcija za opazovanje prekinitve ob pritisku gumba

Znotraj funkcije main, se preko SPI komunikacije nastavi delovanje senzorja. Najprej se sporoči pričetek komunikacije s spremembo izhoda GPIO_PIN_3 na vrednost logične 0 (tako se nadzoruje vrednost Chip select, pri SPI komunikaciji). V spremenljivko outdata[0] se zapiše naslov registra CTRL_REG4 (naslov registra je 0x20, opis registra pa se nahaja v podatkovni listini LIS3DSH na strani 38 [2]. V spremenljivko outdata[1] se zapiše vrednost 0x47, ki se bo zapisala v izbran register in v tem primeru pomeni, da bo senzor deloval s hitrostjo 25 Hz in merjenje pospeškov po vseh treh oseh: X, Y in Z. S funkcijo HAL_SPI_TransmitReceive se v register, naslov katerega je v spremenljivki outdata[0], zapiše vrednost iz spremenljivke outdata[1]. Po zapisu vrednosti, se SPI komunikacija konča s spremembo izhoda GPIO_PIN_3 na logično vrednost 1 in nato program počaka 500 milisekund. Postopek pisanja v registre z uporabo SPI je opisan v podatkovni listini LIS3DSH na strani 27 [2].

```
192   // Activate accelerometer
193   HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_RESET);
194   outdata[0] = 0x20 ; // switch on axes
195   outdata[1] = 0x47 ;
196   HAL_SPI_TransmitReceive(&hspi1, &outdata, &indata, 2, HAL_MAX_DELAY);
197   HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_SET);
198   HAL_Delay(500);
199   outdata[1] = 0x00 ;
```

Slika 8: Koda za vzpostavitev delovanja senzorja

Po nastavitvi senzorja se kliče in izvede opisana funkcija Calibrate.

```
202   Calibrate();
```

Slika 9: Klic funkcije za kalibracijo

Znotraj zanke, ki se ponavlja brez izhodnega pogoja, se iz registrov OUT_X, OUT_Y in OUT_Z prebere zgornjih 8 bitov meritve pospeška za posamezno os [2]. Pred branjem registrov se ponovno nastavi izhod GPIO_PIN_3 na vrednost logične 0. Pri branju vrednosti se v spremenljivko outdata[0] poleg naslovov registrov za meritve posameznih osi: 0x29, 0x2B in 0x2D, prižge še 8. bit iz leve proti desni, ki pove, da gre za ukaz branja iz registra s podanim naslovom. Funkcija HAL_SPI_TransmitReceive nato prebere meritve senzorja iz registrov za posamezno os in jih shrani v spremenljivko indata[1], ta pa se glede na prebran register shrani v spremenljivke AccelX, AccelY in AccelZ.

```
209   while (1)
210   {
211
212
213
214       outdata[0] = 0x29 | 0x80 ; // read x
215       HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_RESET);
216       HAL_SPI_TransmitReceive(&hspi1, &outdata, &indata, 2, HAL_MAX_DELAY);
217       // HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_SET);
218       AccelX = indata[1];
219       outdata[0] = 0x2B | 0x80 ; // read y
220       // HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_RESET);
221       HAL_SPI_TransmitReceive(&hspi1, &outdata, &indata, 2, HAL_MAX_DELAY);
222       // HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_SET);
223       AccelY = indata[1];
224       outdata[0] = 0x2D | 0x80 ; // read z
225       // HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_RESET);
226       HAL_SPI_TransmitReceive(&hspi1, &outdata, &indata, 2, HAL_MAX_DELAY);
227       HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_SET);
228       AccelZ = indata[1];
```

Slika 10: Koda za branje meritev iz registrov senzorja

Prebrane meritve se nato primerjajo z najmanšimi in največjimi vrednosti meritev osi X in Y, pridobljenih iz funkcije Calibrate. V kolikor je meritev osi X ali Y manjša od najmanjše vrednosti te osi se od vrednosti meritve odšteje omenjena najmanjša vrednost. V kolikor pa je vrednost meritve osi večja od največje vrednosti te osi, se od vrednosti meritve odšteje omenjena največja vrednost. Te razlike se shranijo v spremenljivki newxval in newyval glede na os in nam povejo za koliko se je spremenil pospešek od osnovne lege plošče.

```
245     if (AccelX < min_xval)
246     {
247       newxval = AccelX - min_xval;
248     }
249
250    else if (AccelX > max_xval)
251     {
252       newxval = AccelX - max_xval;
253     }
254
255    if (AccelY < min_yval)
256    {
257      newyval = AccelY - min_yval;
258    }
259
260    else if (AccelY > max_yval)
261    {
262     newyval = AccelY - max_yval;
263    }
```

Slika 11: Koda za računanje količine nagiba plošče

Vrednosti, ki se nahajata v spremenljivkah newxval in newyval, se uporabita le v kolikor sta večji od 10 ali manjši od -10, zaradi česar ne vplivajo na spremembe položaja kazalca, manjše spremembe v meritvah pospeškov. Tako lahko uporabnik spreminja položaj kazalca, ko to želi in ne ko naredi le manjše nagibe s ploščo. V kolikor vrednosti ustrezata enemu izmed omenjenih pogojev, pa se ju deli z

10, kar zmanjša količino pospeška pri premiku kazalca miške na zaslonu in se ga tako lažje nadzoruje. V primeru, da vrednosti ne ustrezata pogojema se vrednosti premika kazalca po oseh postavita na 0.

```
265     if ((newxval > 10) || (newxval <-10))
266     {
267      mousehid.mouse_y = (newxval/10); //Di
268     }
269     else mousehid.mouse_y = 0;
270
271     if ((newyval > 10) || (newyval <-10))
272     {
273      mousehid.mouse_x= (newyval)/10;
274     }
275     else mousehid.mouse_x = 0;
```

Slika 12: Koda za nastavitev vrednosti premika kazalca miške

Na koncu zanke se preveri stanje spremenljivke button_flag in v kolikor je njena vrednost nastavljena na logično 1, to pomeni da se je pritisnil modri gumb User ter se izvede sledeča koda. Vrednost miškinega gumba se nastavi na logično vrednost 1, kar pomeni pritisk levega miškinega gumba in s pomočjo funkcije USBD_HID_SendReport pošlje v ustrezni obliki na računalnik. Nato se počaka 50 milisekund in ponovno nastavi vrednost miškinega gumba na logično vrednost 0 ter pošlje s funkcijo USBD_HID_SendReport na računalnik. To zaporedje je potrebno za simulacijo pritiska levega miškinega gumba, ker je potrebno gumb pritisniti in spustiti. Vrednost spremenljivke button_flag se nato postavi na logično 0, s čimer se pripravi na zaznavo naslednjega pritiska modrega gumba User. Izven pogoja se nato ponovno kliče funkcija USBD_HID_SendReport, tako, da se tudi v primeru, da se ni zgodila prekinitev zaradi pritiska gumba, pošljejo podatki o novih premikih kazalca po X in Y osi. Pred ponovitvijo zanke se počaka še dodatnih 10 milisekund.

```
277     if (button_flag==1)
278     {
279      mousehid.button = 1; //left click = 1; right click = 2
280      USBD_HID_SendReport(&hUsbDeviceFS, &mousehid, sizeof (mousehid)); //Hold mouse button
281      HAL_Delay (50); //50 ms delay within mouse click
282      mousehid.button = 0;
283      USBD_HID_SendReport(&hUsbDeviceFS,&mousehid, sizeof (mousehid)); //Release mouse button
284      button_flag =0;
285     }
286
287     USBD_HID_SendReport(&hUsbDeviceFS,&mousehid, sizeof (mousehid)); //Send data to USB
288
289
290     HAL_Delay(10);
291
292
293
294   }
295   /* USER CODE END 3 */
296 }
```

Slika 13: Koda za simulacijo pritiska levega miškinega gumba in pošiljanje podatkov na računalnik

3. Zračna miška upravljana preko žiroskopa

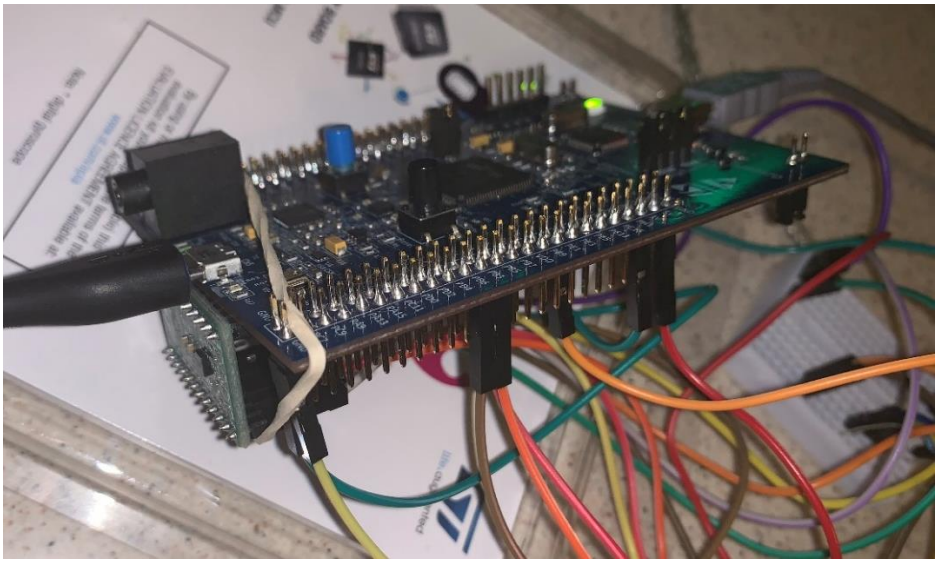
3.1 Opis delovanja in uporabe

Druga oblika zračne miške deluje preko uporabe žiroskopa ter merilnika pospeška, ki se skupaj z merilnikom temperature nahajajo na LSM6DSOX enoti. Omenjena enota je vgrajena na ploščo STEVAL-MKI197V1. Senzorja sporočata mikrokrmilniku svoje meritve preko SPI komunikacije, mikrokrmilnik pa dobljene meritve ustrezno obdela v obliko podatkov, kot jih pošilja računalniška miška in jih nato pošlje preko USB povezave na računalnik. Operacijski sistem računalnika nato sam prepozna povezano ploščo kot računalniško miško in podatke, ki se pošiljajo v ustrezni obliki, uporablja za nadzorovanje kazalca miške na zaslonu računalnika.

Pri uporabi zračne miške, jo je najprej potrebno povezati z računalnikom preko priključkov na plošči, ki se povežeta z dvema USB priključkoma na računalniku. Za povezavo je potrebno uporabiti USB tip A do mini B kabla in USB tip A do mikro B kabla. Zražno miško je priporočljivo usmeriti proti zaslonu, za lažjo uporabo, ker se uporabljajo meritve pospeška za omejevanje kota, znotraj katerega se lahko nadzira premike kazalca miške na zaslonu. S pomočjo teh omejitev se namreč lega zračne miške ne bo preveč spremenila, zaradi česar bi jo bilo v določenih legah težje uporabljati. Sedaj lahko nadzorujemo gibanje kazalca miške na zaslonu računalnika. Ob pritisku modrega gumba User, ki se nahaja na plošči, se izvede pritisk, ob držanju pa držanje levega miškinega gumba na trenutnem položaju kazalca miške.

3.2 Komponente

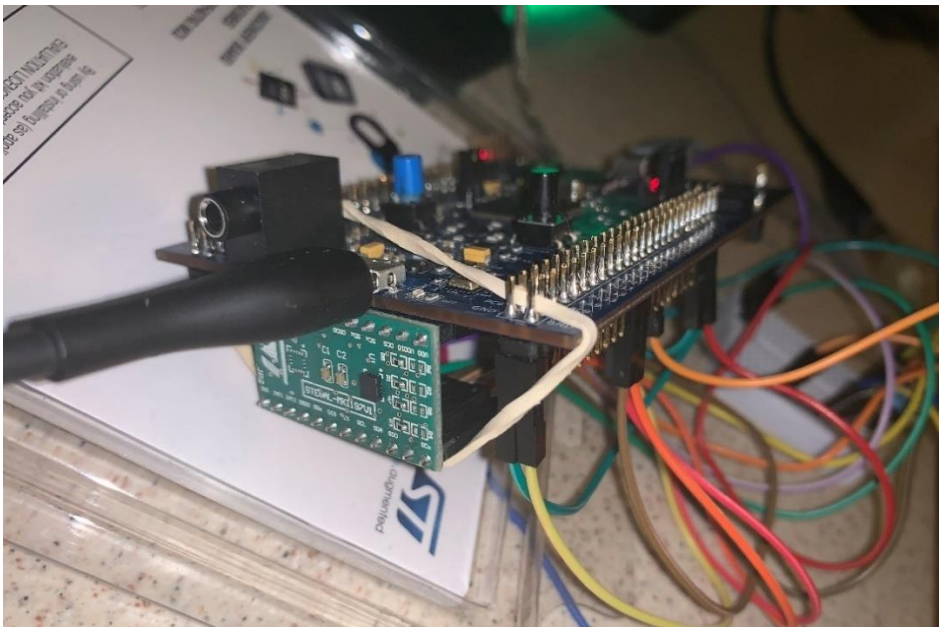
Druga oblika zračne miške je sestavljena iz: STM32F407G-DISC1 plošče z mikrokontrolerjem, STEVAL-MKI197V1 plošče z LSM6DSOX enoto, USB tip A do mini B kabla in USB tip A do mikro B kabla. Omenjena kabla se povežeta preko ustreznih USB mini B in mikro B priključkov na plošči ter USB tip A priključkov na izbranem računalniku. Plošča STEVAL-MKI197V1 je s pomočjo elastike pritrjena na ploščo STM32F407G-DISC1, zaradi česar se zračna miška lažje uporablja.



Slika 14: Zgrajena zračna miška (pogled iz strani)

3.3 Povezave med ploščo s senzorji in mikrokrmilnikom

Ploščo STEVAL-MKI197V1 je bilo najprej potrebno fizično povezati s STM32F407G-DISC1, za komunikacijo z LSM6DSOX enoto. Priključki GND, SDx, SCx in ponovitvi priključkov SDx ter SCx so povezani z ozemljitvijo iz STM32F407G-DISC1 GND priključka. Priključka VDD in VDDIO sta povezana na napajanje 3 V iz STM32F407G-DISC1 3V priključka. Priključek CS je povezan na STM32F407G-DISC1 preko PE2 priključka. Priključek SCL je povezan na STM32F407G-DISC1 preko PB3 priključka. Priključek SDA je povezan na STM32F407G-DISC1 preko PB5 priključka. Priključek SDO je povezan na STM32F407G-DISC1 preko PB4 priključka. Opisana vezava plošč ustreza 1. načinu delovanja LSM6DSOX enote (načini delovanja LSM6DSOX enote in pripadajoče povezave so opisane v podatkovni listini LSM6DSOX na straneh 8 in 9 [4]).



Slika 15: Zgrajena zračna miška (pogled od spredaj)

Tabela povezav med priključki opisanih plošč	
STEVAL-MKI197V1 priključki:	**STM32F407G-DISC1 priključki:**
GND	GND
SDx (oba priključka)	GND
SCx (oba priključka)	GND
VDD	3V
VDDIO	3V
CS	PE2
SCL	PB3
SDA	PB5
SDO	PB4

3.4 Koda

Koda je bila napisana v STM32CubeIDE. Ob izdelavi novega STM32 projekta (imenovanega STM32_F407G_MISKA_Gyro), sem izbral ploščo STM32F407G-DISC1, zaradi česar mi je IDE že izdelal ioc datoteko, z vsemi vhodi in izhodi, ki je namenjena za to ploščo. Najprej sem se v nastali datoteki STM32_F407G_MISKA_pospesevalniki.ioc, s pomočjo CubeMX, premaknil v razdelek Pinout & Configuration ter izbral zavihek A -> Z in nastavil USB_OTG_FS na nastavitev Device_Only, kar pomeni da se bo plošča priključila kot USB naprava na računalnik. Pod USB_DEVICE sem nastavitvi Class For FS IP izbral vrednost Human Interface Device Class (HID), kar pomeni da bo plošča komunicirala z povezanim računalnikom kot USB naprava (v tem primeru kot računalniška miška), prav tako pa sem v razdelku Device descriptor za vrednost PRODUCT_STRING vpisal STM32F407G MISKA. Vpisano ime se bo uporabljalo na povezanem računalniku za poimenovanje zračne miške. Za komunikacijo med mikrokontrolerjem in enoto LSM6DSOX, ki je vgrajena na ploščo STEVAL-MKI197V1, sem uporabil SPI (sinhron protokol za komunikacijo). Izbral sem SPI1 in v razdelku Parameter Settings nastavil Prescaler (for Baud Rate) na vrednost 256, hitrost se tako deli z 256 in je zaradi tega pod vrednostjo 1 Mhz. Pri RCC je potrebno nastaviti uro na vrednost Crystal/Ceramic Resonator, v SYS pa za nastavitev Debug vrednost nastaviti Serial Wire. V oknu Pinout view, sem nastavil priključek PB3 na vrednost SPI1_SCK (preko tega priključka se pošilja urin signal), PB4 na SPI1_MISO (mikrokontroler preko tega priključka sprejema podatke kot gospodar, senzor pa pošilja izbrane meritve kot suženj), PB5 na SPI1_MOSI (mikrokontroler preko tega priključka pošilja podatke ali ukaze kot gospodar, senzor pa jih sprejema kot suženj) in PE2 na GPIO_Output (preko tega priključka se sporoči pričetek SPI komunikacije, ko se pošlje logična vrednost 0 in konec ob logični vrednosti 1), te nastavitve priključkov so bile izbrane za SPI komunikacijo z enoto LSM6DSOX, ker te priključki ob navedenih nastavitvah niso rezervirane za druge namene, kot je opisano v podatkovni listini STM32F407G-DISC1 na straneh 23 in 28 [1]. Način SPI komunikacije za branje in pisanje v registre je opisan v podatkovni listini LSM6DSOX na straneh 20 in 22 [4]. Priključek PA0-WKUP je nastavljen na GPIO_EXTI0 (izbrana vrednost priključka je rezervirana za komunikacijo z modrim User gumbom na plošči), tudi ta nastavitev za uporabo modrega User gumba je opisana v podatkovni listini STM32F407G-DISC1 na straneh 20 in 28 [1]. V nastavitvah za GPIO sem nato za priključek PA0-WKUP nastavil vrednost za GPIO mode na External Interrupt Mode with Rising/Falling edge trigger detection, zaradi česar se lahko prekinitev sproži tako pri pritisku kot tudi spustu gumba in GPIO Pull-up/Pull-down na vrednost Pull-down. Znotraj GPIO sem v zavihku NVIC, vključil EXTI line0 interrupt, zaradi česar se bo, ob pritisku ali spustu modrega gumba User, sprožila prekinitev. V GPIO, sem znotraj zavihka SPI, priključkom PB3, PB4, PB5 nastavitvi Maximum output speed izbral vrednost High. To nastavitev sem nastavil tudi priključku PE2 znotraj GPIO zavihka. Sprememba nazadnje omenjene nastavitve pri vseh navedenih priključkih je pomembna, ker so v nasprotnem primeru prehodi signalov prepočasni in zaradi tega zadnji prebrani bit v komunikaciji velikokrat dobi napačno vrednost (Napaka in načini reševanja napake so opisani v STM32F40x and STM32F41x Errata listini pod poglavjem 2.6.2 [7]). V razdelku Clock Configuration sem nastavitvima APB1 Prescaler in APB2 Prescaler izbral vrednost / 16, s čimer sem povečal delilnik in tako upočasnil hitrost APB ure, kar še dodatno zniža verjetnost da bi pri komunikaciji zadnji bit imel napačno vrednost. S pritiskom na gumb Build project, ki se nahaja v izvlečnem meniju Project, sem generiral izbrane nastavitve v kodo.

Kodo sem pisal v programskem jeziku C znotraj datoteke main.c, ki se je zgenerirala iz opisanih nastavitev. Poleg že dodanih knjižnic sem dodal še knižnico usbd_hid.h, za komunikacijo z računalnikom na način kot bi podatke pošiljala računalniška miška.

```
/* Includes ------------------------
#include "main.h"
#include "i2s.h"
#include "spi.h"
#include "usb_device.h"
#include "gpio.h"

/* Private includes ----------------
/* USER CODE BEGIN Includes */
#include "usbd_hid.h"
/* USER CODE END Includes */
```

Slika 16: Knjižnice dodane v kodi

Nato sem definiral spremenljivke. Spremenljivki indata in outdata, se bosta uporabili za branje in pisanje vrednosti v registre senzorja. V lis_id se bo prebrala identifikacijska številka senzorja. V AccelX, AccelY in AccelZ se bodo hranile meritve pridobljene iz merilnika pospeška. V GyroX, GyroY in GyroZ se bodo hranile meritve pridobljene iz žiroskopa.

V vertValue in horzValue se bodo hranile izračunane vrednosti za katere se bo kazalec miške premaknil, glede na os. Spremenljivka button_flag bo hranila stanje modrega gumba User.

```
// Global variables
uint8_t indata[2];
uint8_t outdata[2] = {0,0};
uint8_t lis_id;
int8_t AccelX;
int8_t AccelY;
int8_t AccelZ;
int8_t GyroX;
int8_t GyroY;
int8_t GyroZ;
HAL_StatusTypeDef SPIStatus;
```

Slika 17: 1. definiranje spremenljivk

```
uint8_t button_flag = 0;



int16_t vertZero = 0;
int16_t horzZero = 0;
int16_t vertValue = 0;
int16_t horzValue = 0;
int16_t sensitivity = 30;
int16_t summ_horz = 0;
```

Slika 18: 2. definiranje spremenljivk

Iz datoteke usb_device.c, ki se nahaja znotraj projekta v računalniški mapi USB_device in podmapi App, sem prilepil USBF_HandleTypeDef, ki definira vrsto podatkov, ki se bodo pošiljali na računalnik. Definiral sem strukturo mouseHID, v kateri se nahajajo posamezni podatki, katere pošilja miška na računalnik v ustreznem vrstem redu. Podatki znotraj strukture so: stanje gumba (ali se je pritisnil ali spustil levi miški gumb označen z vrednostjo 1 ali pa desni miškin gumb označen z vrednostjo 2), položaj

kazalca miške glede na x os, položaj kazalca miške glede na y os in vrednost miškinega kolesa. Vse vrednosti so na začetku nastavljene na 0.

```
100 // MOUSE
101 extern USBD_HandleTypeDef hUsbDeviceFS;
102
103 typedef struct
104 {
105     uint8_t button;
106     int8_t mouse_x;
107     int8_t mouse_y;
108     int8_t wheel;
109 } mouseHID;
110
111 mouseHID mousehid = {0,0,0,0};
```

Slika 19: Struktura uporabljena za ustrezni vrstni red podatkov, ki se pošiljajo na računalnik

V funkciji HAL_GPIO_EXTI_Callback, se opazuje zunanje prekinitve in v kolikor je prekinitev sprožil pritisk ali spust modrega gumba User, ki se nahaja na priključku GPIO_PIN_0, se bo v spremenljivko button_flag zapisalo prebrano stanje gumba oziroma prebrana vrednost iz priključka PA0. Gumb torej proži prekinitev ob spremembi njegovega stanja.

```
148 void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
149 {
150     if (GPIO_Pin == GPIO_PIN_0) //Check if the blue USER butt
151     {
152         //button_flag = 1;
153         button_flag = HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0);
154     }
155 }
```

Slika 20: Funkcija za opazovanje prekinitve ob pritisku ali spustu gumba

Znotraj funkcije main, se preko SPI komunikacije nastavi delovanje senzorja. Najprej se sporoči pričetek komunikacije s spremembo izhoda GPIO_PIN_2 na vrednost logične 0 (tako se nadzoruje vrednost Chip select, pri SPI komunikaciji). V spremenljivko outdata[0] se zapiše naslov registra CTRL1_XL (naslov registra je 0x10, opis registra pa se nahaja v podatkovni listini LSM6DSOX na strani 56 [4]). V spremenljivko outdata[1] se zapiše vrednost 0x20, ki se bo zapisala v izbran register in v tem primeru pomeni, da bo senzor deloval s hitrostjo 26 Hz ter ga tako tudi vklopi. S funkcijo HAL_SPI_TransmitReceive se v register, naslov katerega je v spremenljivki outdata[0], zapiše vrednost iz spremenljivke outdata[1]. Po zapisu vrednosti, se SPI komunikacija konča s spremembo izhoda GPIO_PIN_2 na logično vrednost 1 in nato program počaka 500 milisekund. Postopek se ponovi še za register CTRL2_G, najprej s spremembo izhoda GPIO_PIN_2 na vrednost logične 0. V spremenljivko outdata[0] se tokrat zapiše naslov 0x11 (opis registra se nahaja v podatkovni listini LSM6DSOX na strani 57 [4]). V spremenljivko outdata[1] pa se zapiše vrednost 0x20, ki se bo zapisala tudi v ta register in ponovno pomeni, da bo senzor deloval s hitrostjo 26 Hz ter ga tako tudi vklopi. Po zapisu vrednosti v naslovnjen register z uporabo funkcije HAL_SPI_TransmitReceive ter končanju SPI komunikacije s spremembo izhoda GPIO_PIN_2 na logično vrednost 1, program počaka 500 milisekund. Postopek pisanja v registre z uporabo SPI je opisan v podatkovni listini LSM6DSOX na strani 22 [4].

```
227   // nastavi zacetne nastavitve (aktivacija senzorjev in frekvence)
228   HAL_GPIO_WritePin(GPIOE, GPIO_PIN_2, GPIO_PIN_RESET);
229
230   outdata[0] = 0x10 ; // register za nastavitev acellometra
231   outdata[1] = 0x20 ; // frekvenca na 26Hz
232   HAL_SPI_TransmitReceive(&hspi1, &outdata, &indata, 2, HAL_MAX_DELAY);
233
234
235   HAL_GPIO_WritePin(GPIOE, GPIO_PIN_2, GPIO_PIN_SET);
236   HAL_Delay(500);
237   outdata[1] = 0x00 ;
238   HAL_GPIO_WritePin(GPIOE, GPIO_PIN_2, GPIO_PIN_RESET);
239
240
241   outdata[0] = 0x11 ; // register za nastavitev gyroskopa
242   outdata[1] = 0x20 ; // frekvenca na 26Hz
243   HAL_SPI_TransmitReceive(&hspi1, &outdata, &indata, 2, HAL_MAX_DELAY);
244
245
246   HAL_GPIO_WritePin(GPIOE, GPIO_PIN_2, GPIO_PIN_SET);
247   HAL_Delay(500);
248   outdata[1] = 0x00 ;
```

Slika 21: Koda za vzpostavitev delovanja merilnika pospeška in žiroskopa

Znotraj zanke, ki se ponavlja brez izhodnega pogoja, se iz registrov OUTX_H_G, OUTY_H_G in OUTZ_H_G prebere zgornjih 8 bitov meritve žiroskopa za posamezno os (opisano v podatkovni listini LSM6DSOX na straneh 70 in 71 [4]). Pred branjem vsakega izmed registrov se ponovno nastavi izhod GPIO_PIN_2 na vrednost logične 0. Po branju pa se nastavi izhod GPIO_PIN_2 na vrednost logične 1. Pričetek in prekinitev SPI komunikacije med branji posameznih meritev po oseh je pomembna, ker se v nasprotnem primeru za vse osi preberejo iste vrednosti meritev, kar ni pravilno. Pri branju vrednosti se v spremenljivko outdata[0] poleg naslovov registrov za meritve posameznih osi: 0x23, 0x25 in 0x27, prižge še 8. bit iz leve proti desni, ki pove, da gre za ukaz branja iz registra s podanim naslovom. Funkcija HAL_SPI_TransmitReceive nato prebere meritve senzorja iz registrov za posamezno os in jih shrani v spremenljivko indata[1], ta pa se glede na prebran register shrani v spremenljivke GyroX, GyrolY in GyroZ.

```
284       //Branje gyroskopa
285       outdata[0] = 0x23 | 0x80 ; // read x (pitch) 0x4B
286       HAL_GPIO_WritePin(GPIOE, GPIO_PIN_2, GPIO_PIN_RESET);
287       HAL_SPI_TransmitReceive(&hspi1, &outdata, &indata, 2, HAL_MAX_DELAY);
288            HAL_GPIO_WritePin(GPIOE, GPIO_PIN_2, GPIO_PIN_SET);
289       GyroX = indata[1];
290
291       outdata[0] = 0x25 | 0x80 ; // read y (roll) 0x4D
292            HAL_GPIO_WritePin(GPIOE, GPIO_PIN_2, GPIO_PIN_RESET);
293       HAL_SPI_TransmitReceive(&hspi1, &outdata, &indata, 2, HAL_MAX_DELAY);
294            HAL_GPIO_WritePin(GPIOE, GPIO_PIN_2, GPIO_PIN_SET);
295       GyroY = indata[1];
296
297       outdata[0] = 0x27 | 0x80 ; // read z (yaw) 0x4F
298            HAL_GPIO_WritePin(GPIOE, GPIO_PIN_2, GPIO_PIN_RESET);
299       HAL_SPI_TransmitReceive(&hspi1, &outdata, &indata, 2, HAL_MAX_DELAY);
300       HAL_GPIO_WritePin(GPIOE, GPIO_PIN_2, GPIO_PIN_SET);
301       GyroZ = indata[1];
```

Slika 22: Koda za branje meritev žiroskopa iz registrov enote LSM6DSOX

Podobno se iz registrov OUTX_H_A, OUTY_H_A in OUTZ_H_A prebere zgornjih 8 bitov meritve pospeška za posamezno os (opisano v podatkovni listini LSM6DSOX na straneh 72 in 73 [4]). Pred branjem vsakega izmed registrov se ponovno nastavi izhod GPIO_PIN_2 na vrednost logične 0. Po branju pa se nastavi izhod GPIO_PIN_2 na vrednost logične 1. Pričetek in prekinitev SPI komunikacije med branji posameznih meritev po oseh je tudi tukaj pomembna, ker se v nasprotnem primeru za vse osi preberejo iste vrednosti meritev, kar ni pravilno. Pri branju vrednosti se v spremenljivko outdata[0] poleg naslovov registrov za meritve posameznih osi: 0x29, 0x2B in 0x2D, prižge še 8. bit iz leve proti desni, ki pove, da gre za ukaz branja iz registra s podanim naslovom. Funkcija HAL_SPI_TransmitReceive nato prebere meritve senzorja iz registrov za posamezno os in jih shrani v spremenljivko indata[1], ta pa se glede na prebran register shrani v spremenljivke AccelX, AccelY in AccelZ.

```
305         //Branje accelerometra
306         outdata[0] = 0x29 | 0x80 ; // read x 0x51
307         HAL_GPIO_WritePin(GPIOE, GPIO_PIN_2, GPIO_PIN_RESET);
308         HAL_SPI_TransmitReceive(&hspi1, &outdata, &indata, 2, HAL_MAX_DELAY);
309              HAL_GPIO_WritePin(GPIOE, GPIO_PIN_2, GPIO_PIN_SET);
310         AccelX = indata[1];
311         outdata[0] = 0x2B | 0x80 ; // read y 0x53
312              HAL_GPIO_WritePin(GPIOE, GPIO_PIN_2, GPIO_PIN_RESET);
313         HAL_SPI_TransmitReceive(&hspi1, &outdata, &indata, 2, HAL_MAX_DELAY);
314              HAL_GPIO_WritePin(GPIOE, GPIO_PIN_2, GPIO_PIN_SET);
315         AccelY = indata[1];
316         outdata[0] = 0x2D | 0x80 ; // read z 0x55
317              HAL_GPIO_WritePin(GPIOE, GPIO_PIN_2, GPIO_PIN_RESET);
318         HAL_SPI_TransmitReceive(&hspi1, &outdata, &indata, 2, HAL_MAX_DELAY);
319         HAL_GPIO_WritePin(GPIOE, GPIO_PIN_2, GPIO_PIN_SET);
320         AccelZ = indata[1];
```

Slika 23: Koda za branje meritev pospeškov iz registrov enote LSM6DSOX

Vrednosti meritev žiroskopa glede na os Y in X se prepišeta v spremenljivki vertValue in horzValue. Vrednosti, ki se sedaj nahajata v spremenljivkah vertValue in horzValue, se uporabita le v kolikor sta večji od 2 ali manjši od -2, zaradi česar ne vplivajo na spremembe položaja kazalca, manjše spremembe v meritvah žiroskopa, prav tako pa lahko še vedno izvedemo manjše premike zračne miške za bolj natančen pomik kazalca miške na manjše tarče na zaslonu. V kolikor vrednosti ustrezata enemu izmed omenjenih pogojev, se za vertikalen premik miške še dodatno preveri ali je zračna miška nagnjena pod določenim kotom s pomočjo meritve pospeška po Z osi. Tako se prepreči, da bi se zračna miška pri večjih spremembah usmerjenosti zračne miške, ko je kazalec miške na robu zaslona, ta nato začela uporabljati v položajih v katerih bi njena uporaba postala težja. Za horizontalen premik takšne možnosti omejevanja kota uporabe ni, ker merilnik pospeška ni zmožen izmeriti spremebe horizontalne rotacije, saj se takrat ne spremeni vpliv gravitacije na senzor. Tako sem za omejevanje horizontalne orientacije zračne miške uporabil seštevek horizontalnih meritev žiroskopa. Tak način omejevanja orientacije na žalost ni tako natančen in ima težave ob hitrih horizontalnih premikih zračne miške, ko zaradi velike spremembe vrednosti ni možno premikati kazalca miške po delu zaslona in je potrebno pritsniti na gumb Reset za ponastavitev vrednosti (zaradi te težave bi bilo morda bolje odstraniti horizontalno omejevanje orientacije). Ob izpolnjenih pogojih se izmerjeno vertikalno in horizontalno vrednost deli z 1.3, s čimer se zmanjša količina pospeška pri premiku kazalca miške na zaslonu in se ga tako lažje nadzoruje. V primeru, da vrednosti ne ustrezata pogojema se vrednosti premika kazalca po oseh postavita na 0.

```
380     vertValue = GyroY; // / 3.142857 * 180; /*- vertZero;*/
381     horzValue = GyroX; // / 3.142857 * 180; /*- horzZero;*/
382     vertZero = GyroY;
383     horzZero = GyroX;
384
385
386     if ((AccelZ < 16 && vertValue > 2) || (vertValue < -2 && AccelZ > -36)) { //gor gre
387         mousehid.mouse_y = (vertValue / 1.3 /* * sensitivity*/);    // move mouse on y a
388     }
389     else
390         mousehid.mouse_y = 0;
391
392     if ((summ_horz > -1050 && horzValue > 2) || (summ_horz < 1050 && horzValue < -2)) {
393         mousehid.mouse_x= (horzValue / 1.3 /* * sensitivity*/);
394         summ_horz = summ_horz + (horzValue / 1.3);
395     }
396     else {
397         mousehid.mouse_x= 0;
398         //if (horzValue > 2 || horzValue < -2)
399         summ_horz = summ_horz + (horzValue / 1.3);
400     }
```

Slika 24: Koda za nastavitev vrednosti premika kazalca miške

Na koncu zanke se preveri stanje spremenljivke button_flag, katere vrednost se spremeni ob prekinitvah, ki se zgodijo ob spremebah stanja modrega gumba User. V kolikor je njena vrednost nastavljena na logično 1, to pomeni da se je pritisnil ali pa se drži modri gumb User ter se izvede prvi pogoj. Vrednost miškinega gumba se nastavi na logično vrednost 1, kar pomeni pritisk levega miškinega gumba in s pomočjo funkcije USBD_HID_SendReport pošlje v ustrezni obliki na računalnik. V kolikor pa je njena vrednost nastavljena na logično 0 to pomeni, da modri gumb User ni pritisnjen in se izvede drugi pogoj. Vrednost miškinega gumba se nastavi na logično vrednost 0, kar pomeni spust levega miškinega gumba in s pomočjo funkcije USBD_HID_SendReport pošlje v ustrezni obliki na računalnik. Izven pogojev se nato ponovno kliče funkcija USBD_HID_SendReport za pošiljanje podatkov o novih premikih kazalca po X in Y osi. Pred ponovitvijo zanke se počaka še dodatnih 10 milisekund.

```
402     if (button_flag==1)
403     {
404         mousehid.button = 1; //left click = 1; right click = 2
405         USBD_HID_SendReport(&hUsbDeviceFS, &mousehid, sizeof (mousehid)); //Hold mouse button
406     }
407     else if (button_flag==0) {
408         mousehid.button = 0; //left click = 1; right click = 2
409         USBD_HID_SendReport(&hUsbDeviceFS, &mousehid, sizeof (mousehid));
410     }
411
412     USBD_HID_SendReport(&hUsbDeviceFS,&mousehid, sizeof (mousehid)); //Send data to USB
413
414
415     HAL_Delay(10);
```

Slika 25: Koda za simulacijo pritiska ali držanja levega miškinega gumba in pošiljanje podatkov na računalnik

4. Zaključek

S izdelanima oblikama zračnih mišk, še posebej drugo obliko, sem zadovoljen, saj se ju lahko uporablja podobno kot prave zračne miške. Imata zmožnost izvedbe osnovnih operacij zračnih mišk kot sta nadzorovanje kazalca miške na zaslonu ter simulicija pritiska ali držanja levega miškinega gumba.

Skozi izdelovanje projekta, sem se naučil brati in uporabljati podatkovne listine naprav, ter videl razne napake, ki lahko nastanejo ob komunikaciji med napravami. Prav tako sem se naučil praktično uporabljati meritve senzorjev, kot na primer za nadzor premikov kazalca miške ter kateri podatki se pošiljajo znotraj komunikacije med računalnikom in miško. Ob izdelovanju izdelka, sem prav tako naletel na težave, kot na primer napačno prebrana vrednost zadnjega bita iz registrov enote LSM6DSOX. V tem primeru je bila težava hitrost prehodov signalov, ki jo je bilo potrebno prestaviti, v nastavitvah priključkov, iz običajne nizke hitrosti na visoko. Prav tako je nekoliko otežilo izdelovanje druge oblike zračne miške, manjša količina virov na spletu, ki bi poskušali izvesti podobne povezave in komunikacijo med uporabljenima ploščama.

Za nadgradnjo projekta, bi bilo možno dodati drugi obliki zračne miške še magnetometer, s katerim bi lahko omejili premikanje kazalca miške po horizontalni osi zaslona, zaradi česar miška, med uporabo, ne bi prešla v položaj v katerem bi jo bilo težje uporabljati. Takšno omejevanje trenutno deluje le po vertikalni osi zaslona ob uporabi meritev pospeška, po horizontalni osi pa nažalost ne deluje, saj se ob horizontalnih rotacijah zračne miške ne spremeni vpliv gravitacije na merilnik pospeška. Poleg tega bi bilo možno dodati še kakšen gumb, s pomočjo katerega bi se simuliral pritisk desnega miškinega gumba.

Literatura

[1] Podatkovna listina za STM32F407G-DISC1. Ucilnica.fri.uni-lj.si [Online]. Dosegljivo: Discovery kit with STM32F407VG MCU (uni-lj.si) (zadnji obisk 29. 5. 2022).

[2] Podatkovna listina za senzor LIS3DSH. Mouser [Online]. Dosegljivo: MEMS digital output motion sensor: ultra-low-power high-performance three-axis “nano” accelerometer (mouser.com) (zadnji obisk 29. 5. 2022).

[3] Podatkovna listina za ploščo STEVAL-MKI197V1. ST [Online]. Dosegljivo: LSM6DSOX adapter board for a standard DIL24 socket - Data brief (zadnji obisk 29. 5. 2022).

[4] Podatkovna listina za senzor LSM6DSOX. ST [Online]. Dosegljivo: Datasheet - LSM6DSOX - iNEMO inertial module: always-on 3D accelerometer and 3D gyroscope (st.com) (zadnji obisk 29. 5. 2022).

[5] Emulate STM32F103 as a MOUSE. Controllers tech [Online]. Emulate STM32F103 as a MOUSE » ControllersTech (zadnji obisk 29. 5. 2022).

[6] LSM6DSO issues with SPI. (15. 10. 2020). ST Community [Online]. Dosegljivo: https://community.st.com/s/question/0D53W00000LBjhlSAD/lsm6dso-issues-with-spi (zadnji obisk 29. 5. 2022).

[7] STM32F40x and STM32F41x Errata sheet. ST [Online]. Dosegljivo: es0182-stm32f405407xx-and-stm32f415417xx-device-limitations-stmicroelectronics[17520].pdf (zadnji obisk 29. 5. 2022).

[8] DIY IMU-based SmartTV controller/mouse with Arduino Micro. Hackster.io [Online]. Dosegljivo: https://www.hackster.io/movensllc/diy-imu-based-smarttv-controller-mouse-with-arduino-micro-548353 (zadnji obisk 3. 6. 2022).