Elektrotehnički fakultet
Univerzitet u Banjoj Luci

IZVJEŠTAJ PROJEKTNOG ZADATKA

iz predmeta
UGRAĐENI RAČUNARSKI SISTEMI

Damjan Prerad 1103/16
Slaven Smiljanić 1104/16

Profesor: prof. dr Zlatko Bundalo
Asistent: mr Miladin Sandić

April, 2020. godine

1. Projektni zadatak

U okviru ovog projektnog zadatka, bilo je potrebno uraditi sljedeće: Napisati C program koji očitava podatke koje šalje USB miš, te ih prikazati na 7SEG displejima dostupnim na DE1-SoC razvojnom okruženju. Na prva dva displeja se prikazuju podaci x-ose, na sljedeća dva displeja podaci y-ose, na sljedećem jednom displeju broj lijevih klikova, te na posljednjem, šestom displeju broj desnih klikova. Na DE1-SoC ploči treba prethodno biti podignut Linux OS.

U izvještaju je potrebno priložiti postupak projektovanja, kao i opis korištenih komponenti iz IP kataloga.

2. Rješenje

2.1 Uvodna razmatranja

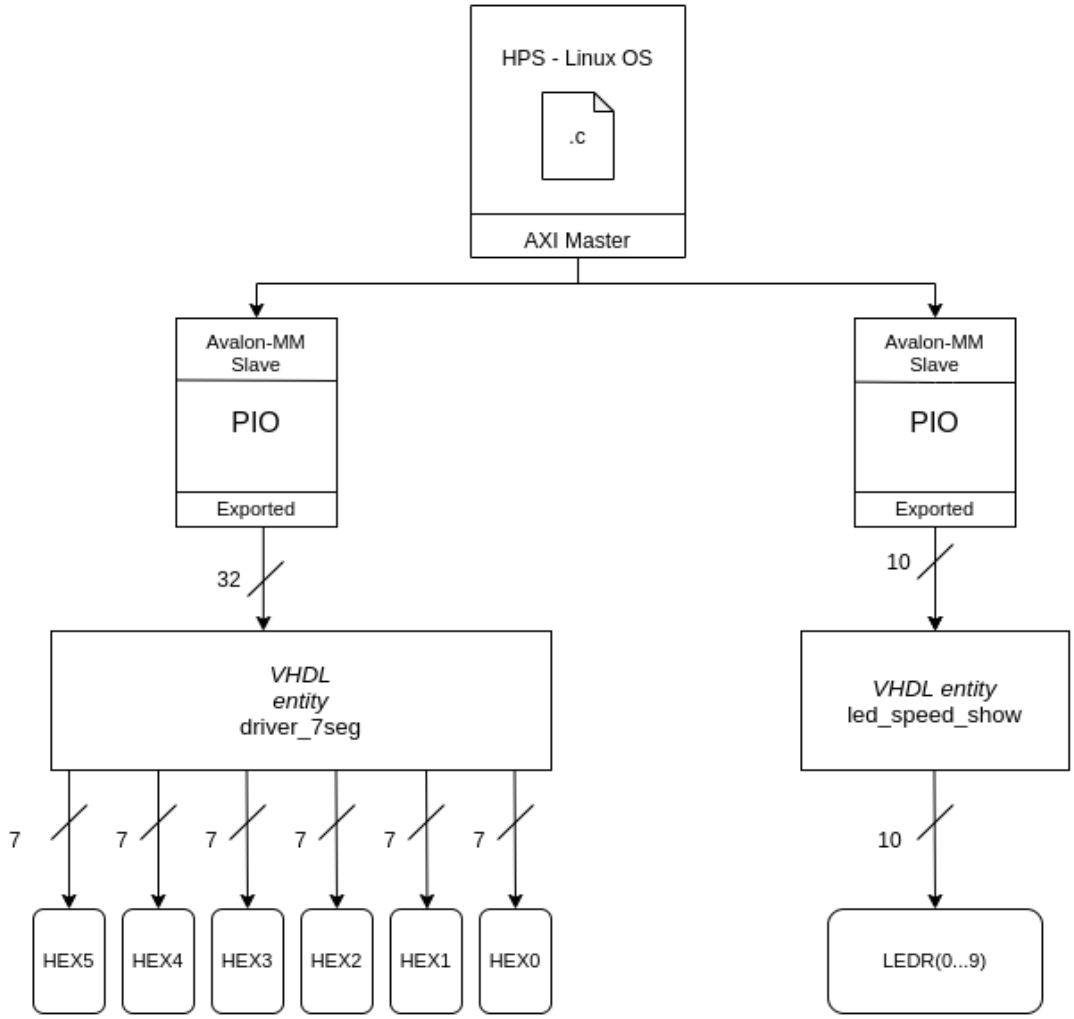
Rješenje problema postavljenog ovim projektnim zadatkom implementirano je na DE1-SoC hardverskoj platformi. Od komponenti koje posjeduje DE1-SoC, za izradu ovog projektnog zadatka iskorišteni su Altera Cyclone V FPGA(uključujući i FPGA fabric i HPS), 6 7-segmentnih displeja i 10 LE dioda.

Takođe, iskorišten je i postojeći Linux drajver za USB miš, da bi se dobili podaci o x i y koordinati miša, kao i podaci o tome da li je pritisnut neki od tastera na mišu. Potrebno je voditi računa o formatu podataka koje daje ovaj drajver, kao i o činjenici da ovaj drajver daje relativne koordinate miša, odnosno pomjeraj po x i y-osi u odnosu na prethodnu poziciju.
Zbog toga je odlučeno da se prilikom pokretanja programa usvoji koordinatni početak kao trenutna pozicija miša, te da se podaci dobijeni od drajvera akumuliraju na vrijednosti x i y ose, te se prikazuju kao kvazi apsolutne koordinate, u vrijednostima od 0 do 99, s obzirom da su za svaku koordinatu na raspolaganju po dva 7-segmentna displeja.
Osim toga, s obzirom da je za broj lijevih, odnosno desnih klikova dostupan po jedan displej, ovaj broj se prikazuje po modulu 10(tj. u vrijednostima od 0 do 9 ciklično).

Osim problema koji je dat projektnim zadatkom, dodatno je implementirana i demonstracija smjera i brzine kretanja miša po x-osi, što se prikazuje na LE diodama.

2.2 Blok-dijagram sistema

Implementirano rješenje se može konceptualno prikazati sljedećim dijagramom:



Slika 2.1 *Blok-dijagram sistema*

2.3 Qsys sistem

2.3.1 HPS

HPS(Hard Processing System) je dio Cyclone V FPGA čipa koji ima integrisan ARM Cortex-A9 mikroprocesor. HPS predstavja zaseban dio čipa i sa FPGA dijelom je povezan preko "mostova", tj. *bridge*-eva. Postoje 3 *bridge*-a: FPGA-to-HPS, HPS-to-FPGA i Lightweight HPS-to-FPGA. Za potrebe ovog projektnog zadatka korišten je Lightweight HPS-to-FPGA *bridge*, širine 32 bita. Ovi *bridge*-evi koriste AXI komunikacioni protokol.

Na HPS-u je instaliran Linux operativni sistem, na kojem se izvršava C aplikacija koja šalje podatke na 7-segmentne displeje i LE diode. Osim toga, HPS se koristi i da konfiguriše FPGA pri *boot*-ovanju.
Dodatni opis funkcionalnosti HPS-a može se naći u [1].

2.3.2 Parallel I/O - PIO

PIO je postojeća IP komponenta koja pruža memorijski mapiran interfejs između Avalon-MM Slave porta i GPIO pinova. Može da ima maksimalno 32 I/O porta, koji se mogu podesiti da budu ulazni, izlazni ili bidirekcioni.. Oni mogu biti mapirani ili na internu korisničku logiku ili na pinove DE1-SoC ploče.

U kontekstu ovog projektnog zadatka, PIO služi da preko Avalon interfejsa dostavi podatke koje je poslao HPS do unutrašnje logike za kontrolu displeja, odnosno LE dioda. Da bi bilo moguće pristupiti ovim podacima, *external_connection*(*Conduit*) port ove komponente se eksportuje, te se tako njime može manipulisati iz VHDL koda.
Avalon-MM Slave interfejs ove komponente se može direktno povezati na AXI Master port HPS-a. Razlog zbog kojeg je ovo moguće je taj što se tzv. *Interconnect* komponenta automatski ubacuje između ova dva porta, te tako omogućava njihovu kompatibilnost.
Ono što je bitno u kontekstu C aplikacije na Linux-u jeste činjenica da se PIO komponenti pristupa kao memorijski mapiranoj komponenti, što olakšava njenu manipulaciju i upis podataka.

Sistem koji je implementiran koristi dvije PIO komponente, jednu za dostavljanje podataka drajveru za 7-segmentne displeje, a drugu za dostavljanje podataka drajveru za LE diode.

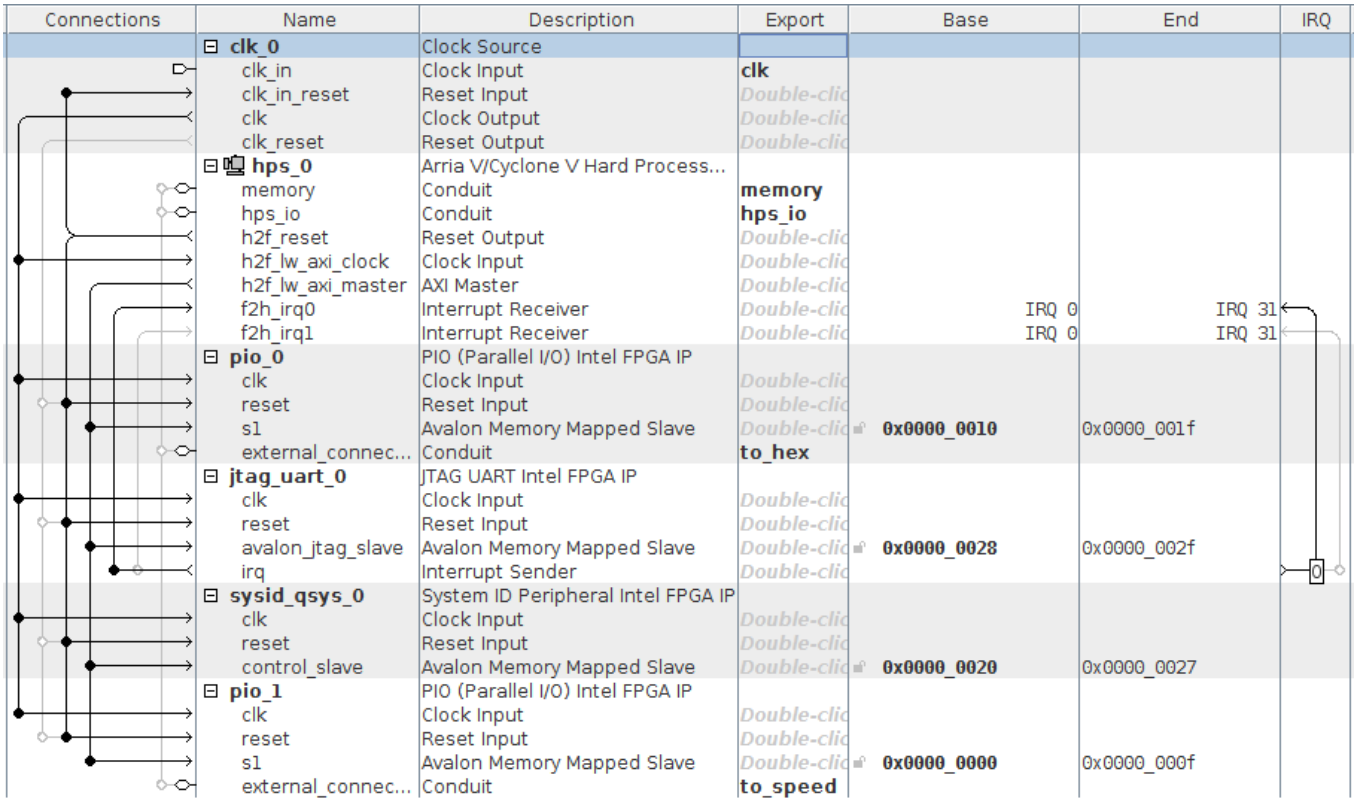
2.3.3 JTAG UART IP

Ova komponenta se koristi za serijsku komunikaciju između *host* računara i DE1-SoC. Korištenjem ove komponente se izbjegava potreba za odvojenom RS-232 konekcijom sa računarom, te se omogućava jednostavan pristup preko USB i Serial konekcije putem programa poput *PuTTY.*

2.3.4 System ID Peripheral IP

Ova komponenta cijelom Qsys sistemu daje jedinstven identifikator. Koristi se da bi se izvršila provjera da li je dati izvršni fajl kompajliran za trenutnu hardversku konfiguraciju FPGA.

Sljedeća slika prikazuje Qsys(u novijim verzijama Platform Designer) sistem sa komponentama koje su prethodno opisane.



Slika 2.2 *Qsys sistem*

2.4 Specifično projektovane komponente

Osim postojećih IP komponenata prikazanih na Slici 2.2, bilo je neophodno projektovati i hardver za manipulaciju podacima koje je dostavio HPS, tačnije bilo je neophodno projektovati modul za *drajvovanje* 7-segmentnih displeja i LE dioda.

2.4.1 Drajver za 7-segmentne displeje

Ovaj drajver je implementiran u okviru VHDL entiteta *driver_7seg*. Kao što se može vidjeti na Slici 2.1, ulaz ovog drajvera je 32-bitni podatak, tipa *std_logic_vector*, dok izlaz predstavlja 6 *std_logic_vector*-a od po 7 bita. Ovi izlazi su direktno mapirani na pinove 7-segmentnih displeja pomoću *Pin Planner*-a. Raspored ovih pinova se može naći u [2].

U nastavku je opisan način rada ove komponente.

Iz C aplikacije dobijaju se 32 bita podataka. Prvih 8 bita predstavlja binarno kodovanu vrijednost *x* koordinate, drugih 8 bita je binarno kodovana vrijednost *y* koordinate, trećih 8 bita je broj lijevih, a četvrtih 8 broj desnih klikova. S obzirom na to, potrebno je parsirati ulazni 32-bitni podatak, te ove 8-bitne brojeve pretvoriti u BCD kodovane. Ova konverzija vrši se *Double Dabble* algoritmom([3]). Nakon toga se dobijeni BCD podaci pretvaraju u 7-segmentni kod, posredstvom *bcd_to_7seg* entiteta. Ovaj pomoćni entitet kao ulaz ima 4-bitni BCD podatak, dok je izlaz 7-bitni kod kojim se pobuđuje 7-segmentni displej koji je u konfiguraciji *common anode*.

2.4.2 Drajver za LE diode

Namjena ove komponente je prikazivanje brzine promjene položaja *x* ose, tj. drugačije rečeno, prikazivanje brzine pokazivača miša po *x* osi. Brzina se prikazuje na 10 dostupnih LE dioda na DE1 SoC ploči.
Način na koji se radi prikaz brzine je sledeći:

- U slučaju da nema pomjeraja, tj. da je brzina jednaka nuli, tada svijetle dvije LE diode u sredini, tj. 5. i 6. dioda
- Ako je brzina pozitivna, tj. miš se pomijera u desno, dvije upaljene LE diode se pomijeraju u desno, a pomjeraj u desno je proporcionalan brzini kretanja
- Ako je brzina negativna, tj. miš se pomijera u lijevo, dvije upaljene LE diode se pomijeraju u desno, a pomijeraj u lijevo je proporcionalan brzini kretanja

Entitet prima označenu brojnu vrijednost širine 10 bita, te na osonvu te vrijednosti odlučuje koji par LE dioda treba da bude upaljen.

2.5 Kreiranje Linux slike

Nakon uspješnog kompajliranja u *Quartus* programskom paketu, dolazi se do izlaznog *.sof* fajla. Ovaj fajl se konvertuje u *.rbf* fajl, koji je potreban Linux image-u za konfigurisanje FPGA.
Potrebno je generisati i Device Tree Blob(*.dtb* fajl) i Device Tree Source(*.dts*). Za ovo se koristi *sopc2dts* program iz Intel EDS paketa. *Board Info* fajlovi su preuzeti sa [4].
Preloader je generisan pomoću *bsp-editor* programa, takođe iz EDS paketa.
Za kreiranje specifičnog *zImage*-a koristi se Buildroot-2017.02-rc1.

Detalji kreiranja Linux slike mogu se pronaći u [5].

2.6 Generisanje *header*-a

Po uspješno generisanom dizajnu, potrebno je napraviti header fajl koji se koristi za ispravan pristup komponentama implementiranim u FPGA dijelu SoC-a sa HPS dijela. Generisani header fajl sadrži neke podatke o tim komponentama, a najbitniji su memorijski *offset*-i pojedinih komponenti i njihova širina (koliko bajtova zauzimaju). Sve komponente koje ulaze u strukturu generisanog header fajla imaju Avalon MM interfejs.
Poznato je da HPS dio SoC-ja može da razmjenuje podatke sa FPGA dijelom samo preko AXI mostova, i kao sto je već rečeno, SOPC builder ubacuje spojnice(eng. *interconnect*) kao spregu između dva interfejsa (AXI i Avalon). Svaka komponenta u dizajnu treba da ima dodijeljen *offset* na magistrali koju ona koristi, bilo da je lightweight AXI ili AXI magistrala. Na osnovu tih *offset*-a HPS će moći da pristupa komponentama. Dakle, u nastalim header fajlovima se nalaze upravo ti *offset*-i ili bazne adrese koje će se koristiti u programu.
Za ispravno generisanje header fajlova koristi se .sopcinfo fajl koji nastaje po uspješnom generisanju Qsys (Platform Designer) dijela projekta u Quartus razvojnom okruženju.
Izvršavanjem skripte *sopc-create-header-files* se dobija željeni header.
Kompletna linija koja daje traženi rezultat je:

```
./sopc-create-header-files "./urs_1..sopcinfo" --single address_base.h --module hps_0
```

Pristup PIO komponentama, koje igraju ključnu ulogu u ovom projektu, ne bi bio moguć bez poznavanja njihovih baznih adresa. Pored toga, u dobijenom header fajlu se mogu naći i bazne adrese SysID komponente i JtagUART komponente (iako se one ne koriste u projektu)

2.7 C aplikacija

U ovom projektu, program koji se izvršava na HPSu, ima višestruku ulogu. Prva je prikupljanje podataka o položaju pokazivača (eng. *cursor*) miša i stanja njegovih tastera (lijevog i desnog), a druga je prilagođenje dobijenih podataka i njhovo prosljeđivanje na FPGA dio koji ce na odgovarajući način da ih i prikaže.

2.7.1 Prikupljanje podataka sa miša

Pošto se program izvršava pod Linux okruženjem, prikupljanje podataka sa miša je u velikoj mjeri olakšano. Poznato je da se u Linux operativnom sistemu sve predstavlja fajlovima, pa tako i dio sistema zadužen sa interfejs sa mišom. Kako već postoji drajver koji je zadužen za upravljanje mišom, dovoljno je otvoriti odgovarajući fajl i pročitati vrijednosti. Fajl koji je potrebno otvoriti nalazi se na putanji */dev/input/mice*. Na putanji */dev/input* se nalaze svi drajveri koji upravljaju ulaznim uređajima, pa tako i mišom. Više o skupu drajvera za podršku ulaznim uređajima se može naći na [6].

Radi ispravnog pribavljanja podataka sa miša potrebno je bolje poznavanje drajvera povezanih sa putanjom /dev/input/mice. Podrška za miš u Linux-u je realizovana skupom modula. Osnovni moduli su *usbhid*, *usbcore*, *uhci_hcd*, *input*, *mousedev*. Prilikom *boot*-anja Linux kernela oni su podrazumijevano umetnuti.
Ukoliko navedeni moduli nisu umetnuti, neće biti moguće dobiti željene podatke sa miša.
Nakon umetanja, kreira se "*node*" fajl koji se koristi za interfejs miša sa sistemom. To se lako postiže izvršavanjem sledećeg niza komandi

cd /dev
mkdir input
mknod input/mice c 13 63

Drajver za miša je *charachter device*, što je ujedno i najjednostavniji tip drajvera.
Da sve radi na ispravan način, može da se potvrdi izvršavanjem naredbe

cat /dev/input/mice

Nakon čega bi trebalo da se dobije ispis "sirovih" podataka sa miša na ekran.

Čitanjem tri bajta sa navedene putanje dobijaju se svi podaci koje miš proslijeđuje u jednom slanju. Podaci su organizovani na sledeći način:

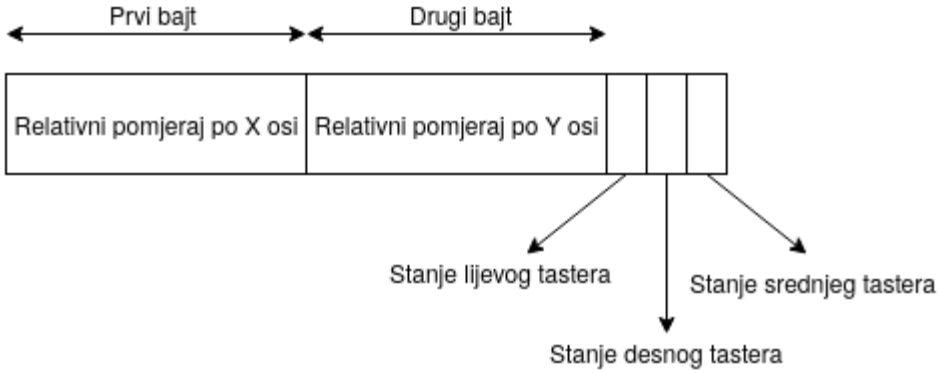
- Prvi pročitani bajt je relativna promjena položaja miša po *x* osi u odnosu na prethodni položaj dat u pikselima
- Drugi pročitani bajt je relativna promjena položaja miša po *y* osi u odnosu na prethodni položaj dat u pikselima
- Treći pročitani bajt posjeduje podatke o stanju tastera miša. Tako prvi bit ima stanje logičke jedinice ako je lijevi taster pritisnut, a ako nije

pritisnut, onda on ima stanje logičke nule. Isto vrijedi i za drugi bit koji govori o stanju desnog tastera, a treći bit o stanju srednjeg tastera.

U zavisnosti od toga kako je miš konfigurisan, moguće je dobiti i stanje scroll-a (prema gore ili prema dolje). Taj podatak će se nalaziti u četvrtom bajtu. Da bi se mogao dobiti taj podatak potrebno je da miš bude podešen kao *IMPS/2* uređaj. U slučaju da je podešen kao *PS/2* uređaj, tada nije moguće dobiti stanje *scroll*-a.

U svrhe projekta dovoljna su prva tri bajta, i prema tome, stanje *scroll*-a ovdje nije od interesa.

Kao što je već rečeno, pomjeraj miša se daje u relativnim koordnatama u odnosu na prethodni položaj miša. Radi dobijanja apsolutnih kooridnata vrši se akumulacija relativnih pomjeraja. Pošto se *x* i *y* koordinate prikazuju na 7-segmentnom displeju i to za svaku koordinatu po dva displeja, tj. maksimalno vrijednost u iznosu od 99 može da se prikaže, te su vrijednosti tih koordinata ograničene na opseg od 0 do 99. Drugim riječima, akumulacija relativnih pomjeraja može da ode u saturaciju.
Brojanje pritiska tastera se prikazuje na jednom 7-segmentnom displeju (po jedan displej za lijevi i za desni taster). Brojanje se radi detekcijom "rastuće" ivice odgovarajućeg bita za dati taster, tj. svaki prelaz sa 0 na 1 se računa kao jedan pritisak tastera.



Slika 2.3 *Struktura podataka iz /dev/input/mice*

Dake, u C programu, za pribavljanje podataka sa miša, je potrebno otvoriti fajl koji se nalazi na putanji */dev/input/mice*, pročitati 3 bajta i izvući informaciju iz pročitanih podataka na odgovarajući način.
Radi manipulacije fajlovima u program se uključuju <*fcntl.h*> i <*unistd.h*> header fajlovi.

Definiše se putanja `#define MOUSE_PATH "/dev/input/mice"` i otvara fajl

```
if((fd_mouse = open(MOUSE_PATH, O_RDWR)) == -1)
{
    printf("Can't open %s!\n", MOUSE_PATH);
    return 0;
}
```

Potrebno je deklarisati strukturu u koju će biti upisivani "sirovi" podaci

```
int8_t raw_data[3];
int num_of_bytes_read;
```

Te nakon čega je moguće izvršiti čitanje podataka

```
num_of_bytes_read = read(fd_mouse, raw_data, sizeof(raw_data));
```

Obrada podataka, akumulacija i brojanje pritisaka tastera su sada omogućeni.

```
x_pos += raw_data[1];
y_pos += raw_data[2];

if(x_pos <= LOWER_LIMIT) x_pos = LOWER_LIMIT;
if(y_pos <= LOWER_LIMIT) y_pos = LOWER_LIMIT;
if(x_pos >= UPPER_LIMIT) x_pos = UPPER_LIMIT;
if(y_pos >= UPPER_LIMIT) y_pos = UPPER_LIMIT;
```

Akumulacija promjene položaja je jednostavna u odnosu na brojanje pritisaka tastera, što se može zaključiti i sa sljedećeg isječka koda.

```
p_l_btn = l_btn;
p_r_btn = r_btn;

l_btn = !!(raw_data[0] & 1);
r_btn = !!(raw_data[0] & 2);

button_press_count(p_l_btn, l_btn, &l_count);
button_press_count(p_r_btn, r_btn, &r_count);

l_count = l_count % 10;
r_count = r_count % 10;
```

Te funkcija *button_press_count* je

```
void button_press_count(int8_t p_btn, int8_t btn, uint8_t* count)
{
    if(btn - p_btn == 1)//rising edge
    {
        (*count)++;
    }
}
```

2.7.2 Proslijeđivanje podataka FPGA dijelu SoC-a

U ovom koraku se koristi prethodno generisani header fajl sa baznim adresama. Pristup AXI magistralama sa HPS dijela u linuxu je omogucen kroz fajl */dev/mem*. Ovim fajlom, čitav adresni prostor uređaja povezanih na HPS se prikazuje kao "*flat memory space*". Na ovaj način svim periferijama se jednosavno pristupa i vrši razmjena podataka. U [2] se može vidjeti na kojoj pozicij su mapirane H2F, F2H AXI magistrale i Lightweight AXI magistrala koje se koriste za pristup FPGA dijelu. U projektu se koristi samo LW AXI, koji spada u hardverske periferije Cyclone V SoC-a. Početak LW AXI-ja je na *offset*-u 0xff200000.

```
#define ALT_LWFPGASLVS_OFST 0xFF200000

#define HW_REGS_BASE ( 0xFC000000 )
#define HW_REGS_SPAN ( 0x04000000 )
#define HW_REGS_MASK ( HW_REGS_SPAN - 1 )
```

Zbog toga što se program izvršava pod Linux operativnim sistemom u korisničkom prostoru (eng. *userspace*), direktan pristup fizičkim lokacijama nije moguć, jer Linux OS koristi mehanizam virtuelne memorije. U okviru pristupa FPGA dijelu sa HPS dijela, Linux operativni sistem donekle usložnjava čitav postupak. Radi zaobilaženja problema koji nasaje iz potrebe pristupa fizičkim memorijskim lokacijama, postoje funkcije *mmap()* i *munmap()* koje su dio standardne, *glibc*, biblioteke. Ove funkcije omogućavaju da se određeni adresni opseg mapira na memorijski objekat kojim se može pristupati iz programa. Na ovaj način je omogućen pristup periferijama iz korisničkog prostra u Linux operativnom sistemu.
Sljedeći isječak koda otvara */dev/mem* fajl i vrši gore navedeno mapiranje.

```
if ((fd_mem = open(MEM_PATH, (O_RDWR | O_SYNC))) == -1) {
            printf("Can't open %s!\n", MEM_PATH);

            return 1;
    }

lw_virtual_base = mmap(NULL, HW_REGS_SPAN, (PROT_READ | PROT_WRITE), MAP_SHARED, fd_mem, HW_REGS_BASE);

if (lw_virtual_base == MAP_FAILED) {
            printf("mmap() failed\n");
            close(fd_mem);
            close(fd_mouse);
            return 1;
    }
```

Sada se određuju počeci pojedinih komponenti u FPGA dijelu i to koristeći bazne adrese dobijene u generisanom header fajlu.

```
display_virtual_addr = lw_virtual_base +
                                    ((unsigned long)(ALT_LWFPGASLVS_OFST + PIO_0_BASE) &
                                    (unsigned long)(HW_REGS_MASK));
speed_virtual_addr = lw_virtual_base +
                                    ((unsigned long)(ALT_LWFPGASLVS_OFST + PIO_1_BASE) &
                                    (unsigned long)(HW_REGS_MASK));
```

PIO_0_BASE i *PIO_1_BASE* su bazne adrese PIO (eng. *Parallel Input/Output*) komponenti koje su korištene u projektu.
Sada se radi prilagođenje informacija o mišu i njihovo slanje na FPGA dio. Slanje se veoma jednostavno izvodi dereferenciranjem memorijske lokacije date periferne jedinice i upisivanjem vrijednosti koja želi da se proslijedi.

```
data_to_send = x_pos | (y_pos << 8) | (l_count << 16) | (r_count <<24);
*((uint32_t*)display_virtual_addr) = data_to_send;
*((uint32_t*)speed_virtual_addr) = raw_data[1];
```

Po završetku rada programa radi se osloađanje korištenih resursa i izlazi se iz programa.

```
if (munmap(lw_virtual_base, HW_REGS_SPAN) != 0) {
            printf("munmap() failed...\n");
    }

close(fd_mem);
close(fd_mouse);
return 0;
```

3. Resursi

[1] *Cyclone V Hard Processor System - Technical Reference Guide* (https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/cyclone-v/cv_54001.pdf)

[2] *DE1-SoC User Manual* (https://www.intel.com/content/dam/altera-www/global/en_US/portal/dsn/42/doc-us-dsnbk-42-1004282204-de1-soc-user-manual.pdf)

[3] *Double Dabble* algoritam, *Wikipedia* (https://en.wikipedia.org/wiki/Double_dabble)

[4] https://github.com/mruizglz/UPM-ES-cyclonevbsp

[5] *Embedded Systems - Using Quartus and Buildroot for building Embedded Linux Systems V1.9*, Mariano Ruiz, Antonio Carpeno (http://oa.upm.es/45352/1/DE1-SoC_Embedded_Linux_Systems_1_9.pdf)

[6] Linux Input drivers, Kernel.org (https://www.kernel.org/doc/Documentation/input/input.txt)