SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

ELEKTROTEHNIČKI FAKULTET

Sveučilišni studij

DIZAJN KOMUNIKACIJSKOG UPRAVLJAČKOG SKLOPA PRIMJENOM FPGA

Završni rad

Dragan Slišković

Osijek, 2009.

SADRŽAJ

1. UVOD

Cilj ovog završnog rada je rješavanje problema FPGA komunikacijskog sučelja PC – FPGA s pomoću paralelnog protokola EPP. U radu je opisan EPP protokol, njegov najvažniji dijelovi hardwerska i programska podrška. Također je opisana FPGA arhitektura i dizajn EPP-a pomoću programskog koda VHDL. U prvom poglavlju opisan je i predstavljen projekt izrade komunikacijskog sučelja EPP na FPGA platformi. Komunikacija je uspostavljena sa računalom preko aplikacije izrađene u C++ programskom jeziku. Za provjeru slanja i primanja podataka izrađen je hardverski pokaznik sa LED diodama za prikaz stanja na podatkovnim i upravljačkim pinovima EPP porta.

U radu je predstavljen FPGA dizajn komunikacijskog sučelja PC – FPGA s pomoću paralelnog protokola EPP. Opisan je EPP protokol i FPGA tehnologija. Definirano je EPP komunikacijsko sučelje na FPGA koristeći VHDL jezik za opis hardvera HDL (engl. *hardware description language*). Najčešći HDL (engl. *hardware description language*) programski jezici su VHDL i Verilog.

U drugom poglavlju prestavljen je FPGA, te su opisane njegove glavne karakteristike. Opisana je arhitektura FPGA čipa, njegova usporedba te primjena u tehnologiji. Poglavlje tri sadrži opis i slike hardverskog dijela sustava za komunikaciju između računalnog sklopovlja i FPGA sklopovlja. Opisan je također dizaj kako se izvodi programski tijek pomoću programskkog koda VHDL.

2. ENHANCED PARALLEL PORT (EPP)

EPP protokol je dio IEEE 1284 paralelnog port standarda. IEEE također definira SPP (engl. *Standard Parallel Port*) i ECP (engl. *Extended Capability Port*) protokole, ali EPP(engl. *Enhanced Parallel Port*) je bolji ima bolje karakteristike brzine i jednostavniju građu. Brzina slanja/primanja podataka EPP protokola kreće se od 500 kb/s do 2MB/s.

Pomoću EPP (engl. *Enhanced Parallel Port*) porta komunikacija između računala i periferijalnih uređaja je lakša i brža. EPP porta je dio IEEE 1284 paralelnog port standarda. IEEE također definira SPP (Standard Parallel Port) i ECP (Extended Capability Port) portove, ali EPP je bolji ima bolje karakteristike brzine i jednostavniju građu.

EPP glavne karakteristike:

EPP (engl. *Enhanced Parallel Port*) omogućava dvosmjernu komunikaciju preko paralelnog porta tj. može čitati iz i upisivati podatke u periferijalne dijelove računala npr: printer, skener i dr. Nema točno definiran koncept po kojemu se šalju i primaju podatci. Podatci se mogu slati jedan bajt podataka ili 10000 bajt-a ako to želimo. Stanja se mogu izmjenjivati da podatke pišemo ili čitamo bez točno definiranog reda. EPP port omogućava komunikaciju “adresa“ i “podataka“.

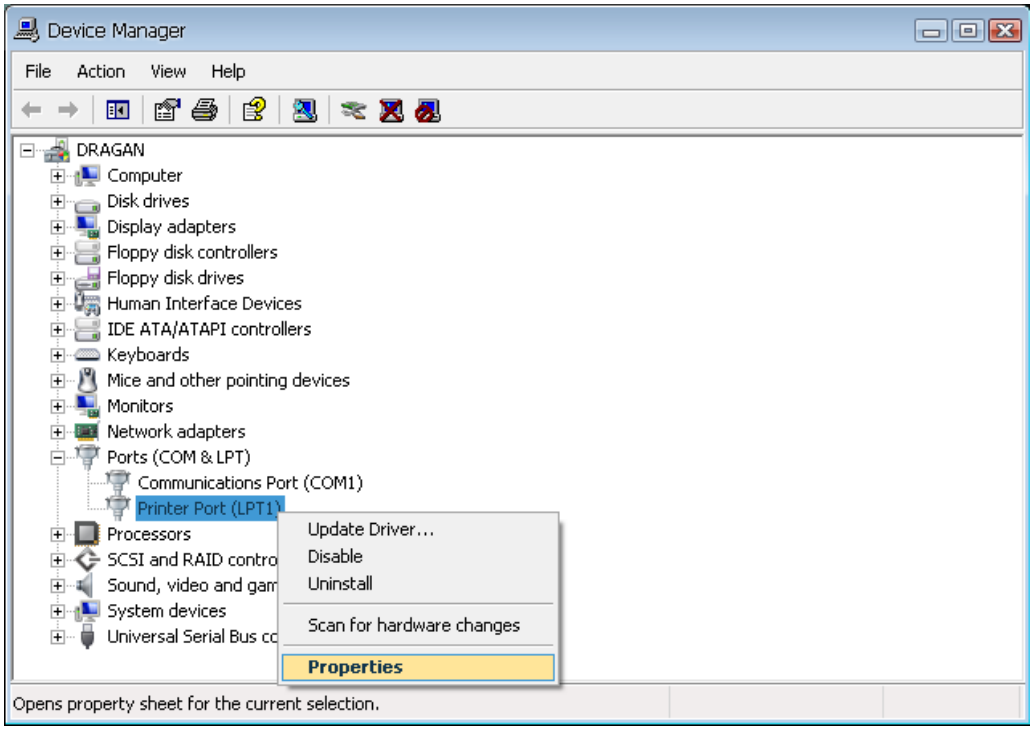
Postoje četiri vrste EPP transakcija koje se mogu narediti od strane računala. To su:

- Upiši adresu
- Pročitaj adresu
- Upiši podatak
- Pročitaj podatak

2.1 Programsko sučelje

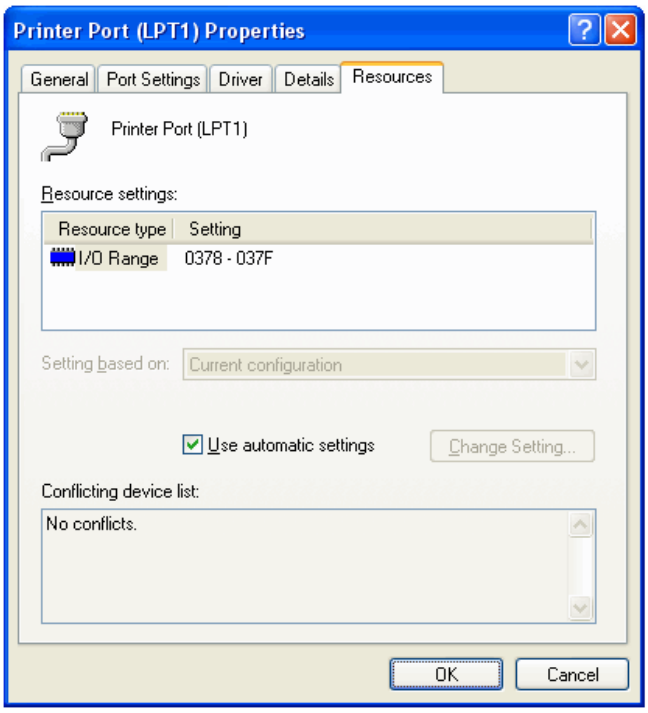
EPP programsko sučelje je vrlo jednostavno. Da bih koristili EPP protokol prvo moramo u računalnom dijelu BIOS-u omogućiti rad EPP porta. Najpoznatija adresa EPP porta je 0x378h, oznaka h uz adresu znači da je to adresa porta data u heksadecimalnom kodu.

Na slici (**Sl. 2.1.***Pronalaženje adrese paralelnog porta*) vidimo gdje se u operaciskom sustavu (Windows) pronalazi adresa porta.



Sl. 2.1.*Pronalaženje adrese paralelnog porta*

Zatim se prikazuje nova kartica „printer port“ gdje se vidi detaljnij prikaz adresa paralelnog porta. Pod karticom printer port izaberemo opciju izvori (engl. *resources*)i tamo vidimo protokol sa adresom 0x378 do 0x37F. Vidljivo iz slike (**Sl. 2.2***adresa porta*).



Sl. 2.2.*Adresa porta*

Pronalaženjem osnovne adrese paralelnog porta možemo odrediti I ostale adrese za registar podataka, statusni registar I kontrolni registar. Osnovne adrese se odzačavaju sa LPT1 I LPT2 za registre podataka, status registar i kontrolni registar vidimo u tablici (**Tablica 2.2.** Osnovne Adrese Registra).

Tablica 2.2. Osnovne adrese registra

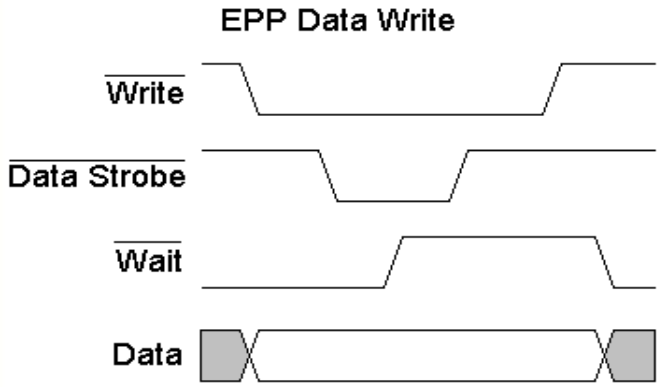
Registracija	LPT1	LPT2
registar podataka (baseaddress + 0)	0x378	0x278
status register (baseaddress + 1)	0x379	0x279
kontrola registrirati (baseaddress + 2)	0x37a	0x27a

2.2 Stanja EPP (Enhanced Parallel Port) porta

Kada se želi ostvariti komunikacija sa EPP (engl. *Enhanced Parallel Port*) portom moraju se zadovoljiti određeni uvjeti. Kod EPP porta ti uvjeti su njegov ciklus koji govori u kojem stanju se moraju nalaziti signali da bi se mogla ižvršiti komunikacija odnosno pisanje i četanje podataka.

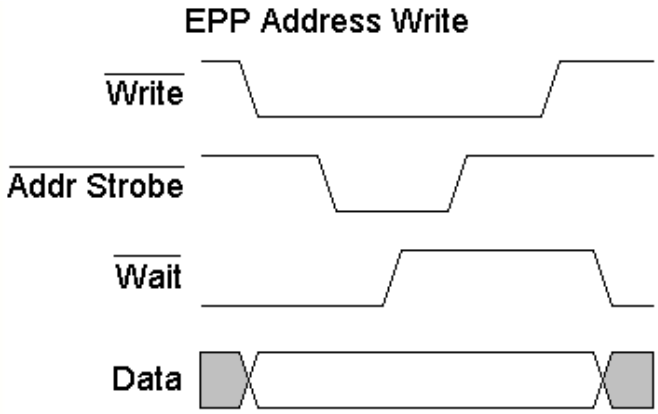
U daljnjem tekstu prikazani su ciklusi koje EPP (engl. *Enhanced Parallel Port*) port mora zadovoljiti da bi se čitao podatak ili zapisivao podatak. Na slici (**Sl. 2.3.***EPP (Enhanced Parallel Port) ciklus podatkovnog zapis*) se nalazi ciklus za podatkovni zapis podataka, sljedeće slika (**Sl. 2.4.***EPP (Enhanced Parallel Port) ciklus addresnog zapisa)* prikazuje ciklus adresnog zapisa podatak.

Sljedeća dva dijagrama prikazuju suprotne cikluse a to su ciklusi za čitanje podataka. Na prvoj slici (**Sl. 2.5.***EPP (Enhanced Parallel Port) ciklus podatkovnog čitanja)* vidimo ciklus za podatkovno čitanje, na posljednoj slici (**Sl. 2.6.** *EPP (Enhanced Parallel Port) ciklus addresnog čitanja*) se nalaze ciklusi za addresno čitanje podataka.



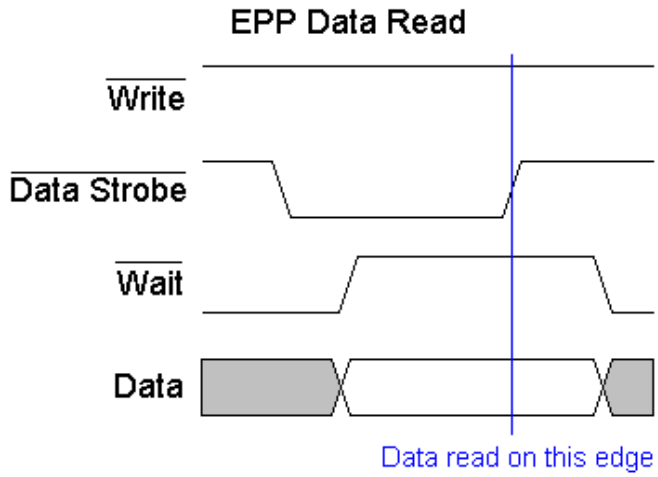
Sl. 2.3.*EPP (Enhanced Parallel Port) ciklus podatkovnog zapis*

Sa slike (**Sl. 2.3.***EPP (Enhanced Parallel Port) ciklus podatkovnog zapis*) se vidi da bi zapisivali podatke stanje Write signala se mora nalaziti na uzlaznom rastućem bridu, signal Data Strobe se mora nalaziti u stanju '0', signal Wait se nalazi u stanju '1' i tek tada EPP data write zadovoljava sve uvjete i može doći do zapisivanja podatka sa podatkovne sabirnice.



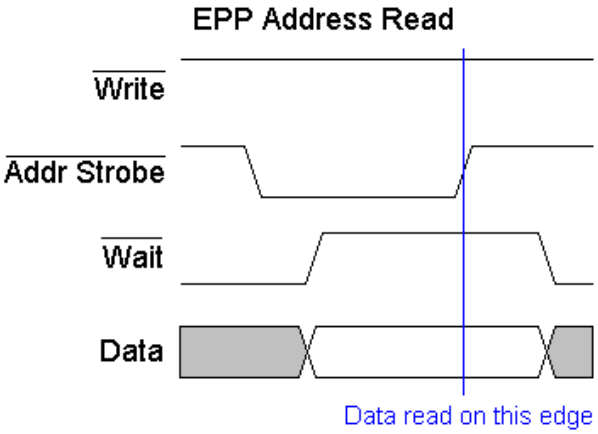
Sl. 2.4.*EPP (Enhanced Parallel Port) ciklus addresnog zapisa*

Na slici (**Sl. 2.4.***EPP (Enhanced Parallel Port) ciklus addresnog zapisa*) se vidi da bi zapisivali adresno stanje, write signala mora se nalaziti u stanju '0', signal addr strobe se mora nalaziti u stanju '0', signal wait se nalazi u stanju '1' i tek tada EPP Addres Write zadovoljava sve uvjete i može doći do zapisivanja podatka sa podatkovne sabirnice.



Sl. 2.5.*EPP (Enhanced Parallel Port) ciklus podatkovnog čitanja*

Na slici (**Sl. 2.5.***EPP (Enhanced Parallel Port) ciklus podatkovnog čitanja*) se nalazi ciklus čitanja podataka sa EPP Data registra, da bi se izvršilo čitanje signal write se mora nalaziti u stanju '1', signal data strobe se nalazi u stanju rastućeg prida a signal wait se nalazi u stanju '1'. Kada su ta stanja zadovoljena tada dolazi do čitanja podataka sa podatkovne sabirnice EPP porta.

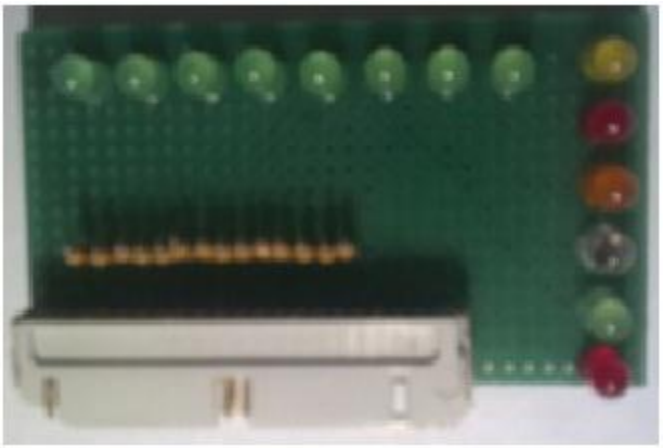


Sl. 2.6. *EPP (Enhanced Parallel Port) ciklus addresnog čitanja*

Slika (**Sl. 2.6.** *EPP (Enhanced Parallel Port) ciklus addresnog čitanja*) nam prikazuje ciklus čitanja addresa sa EPP porta, da bi se izvršilo čitanje addresa signal write se mora nalaziti u stanju '1' kao i kod ciklusa podatkovnog čitanja, signal data strobe se nalazi u stanju rastućeg prida a signal wait se nalazi u stanju '1'. Kada su ta stanja zadovoljena tada dolazi do čitanja addresa sa sabirnice EPP porta.

2.3 Testiranje FPGA konektora

Za provjeru slanja i primanja podataka izrađen je hardverski pokaznik sa LED diodama za prikaz stanja na podatkovnim i upravljačkim pinovima EPP porta kojega vidimo na slici (**Sl.2.7.***Pokaznik sa LED diodama).*



Sl.2.7.*Pokaznik sa LED diodam*

Napravljen je programski kod u C++ programskom jeziku koji služi za provjeru slanja i primanja podataka. U programskom kodu su definirani kontrolni bitovi od 0 do 7, također su definirani portovi i adrese po kojima će se slati podatci. Prikazane su također i funkcije za provjeru slanja podataka Prikazani su osnovni dijelo koda, cijeli programski kod nalazi se u prilogu.

Dijelovi C++ koda:

```
// kontrolni bitovi
#define WRITE_BIT                0x1        // C0
#define DATA_STROBE_BIT          0x2        // C1
#define INIT_BIT                 0x4        //  C2
#define ADDR_STROBE_BIT          0x8        // C3
#define IRQ_ENABLE_BIT           0x10       //  C5
```

Vidimo kontrolne bitove programskog c++ koda, definirani su signali write, data_strobe, addres_strobe irq, re int_bit.

```
// portovi
short _stdcall Inp32(short address);
void _stdcall Out32(short address, short data);
// adresse
short DATA_ADDR          = 0x378;
short STATUS_ADDR        = 0x379;
short CONTROL_ADDR       = 0x37a;
short EPP1_ADDR          = 0x37b;
short EPP2_ADDR          = 0x37c;
```

U ovom dijelu koda prikazani su portovi preko kojih se vrši slanje podataka, te adrese za kontrolu, također vidimo osnovnu adresu EPP-a koja je 0x378. Adrese su data_addr, status_addr, control_addr, short_epp1_addr te short_epp2_addr.

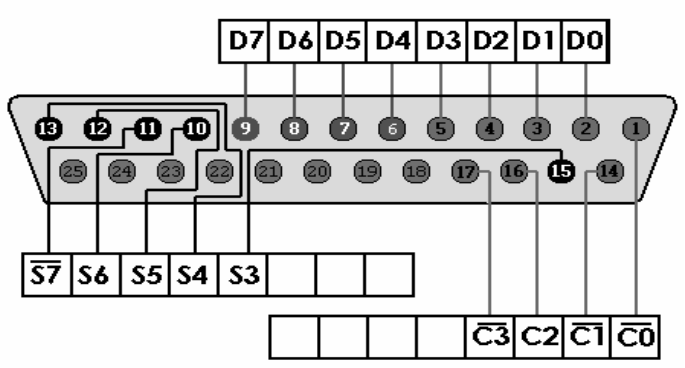
2.4 Hardversko sučelje

Paralelni port PC računala se kontrolira preko tri 8-bitna memorijska registra sa sljedećim nazivima:

- Data port
- Control port
- Status port

Data port sadrži jedan bajt podataka koji će se naći na izlaznim linijama paralelnog porta pod uvjetom da su zadovoljeni određeni zahtjevi. Control port kontrolira operaciju slanja podataka, dok Status port ukazuje na stanje uređaja.

Na slici (**Sl. 2.8.***DB-25 konektor*) vidimo prikazan DB25 printer konektora koji se nalazi na matičnoj ploči računala. Taj konektor će mo upotrijebiti da spojimo Xilinx Spartan pločicu sa računalom.



Sl. 2.8.*DB-25 konektor*

Na DB25 konektoru pinovi od 2 do 9 su 8-bitni, u EPP načinu rada 8-bitni pinovi rade dvosmjerno tj. mogu i primati podatke i slati podatke, pinovi 18-25 na DB-25 konektoru koriste se za masu (engl. *ground*).

Na slici je prikaz FPGA konektor koji služi za komunikaciju odnosno prijenos podataka između PC i FPGA pomoću EPP porta. Konektor se spaja na paralelni port računala, drugi kraj se spaja na maketu Spartan 3, zatim se na maketu Spartan 3 spaja testnu maketu sa ledicama, te se tako provjerava dali svi signali rade na FPGA konektoru odnosno dali se podatci šalju i primaju.

Sl. 2.9.FPGA k*onektor*

2.4.1 Raspored signala na FPGA konektoru

Tablica (**Tablica 2.3.** pinovi na FPGA konektoru, RS232 kablu, njihova funkcija i ufc pinovi) prikazuje pinove na GPGA konektoru, RS232 kablu, te njihovim funkcijama njihova funkcija i ufc pinovi koje ćemo kasnije definirati u Xilinx programskom alatu.

Tablica 2.3. Pinovi na FPGA konektoru, RS232 kablu, njihova funkcija i ufc pinovi

FPGA konektor	RS232 kabel	EPP funkcija	IN/OUT	UFC pinovi
1	18 – 25	Ground	-	-
5	2	Data 0	IN/OUT	N7
7	3	Data 1	IN/OUT	T8
9	4	Data 2	IN/OUT	R6
11	5	Data 3	IN/OUT	T5
13	6	Data 4	IN/OUT	R5
15	7	Data 5	IN/OUT	C2
17	8	Data 6	IN/OUT	C1
19	9	Data 7	IN/OUT	B1

Ostali važni pinovi koji se nalaze na DB25 konektoru se nalaze u (**Tablica 2.4.** *ostali pinovi paralelnog porta*).

Tablica 2.4. Ostali pinovi paralelnog porta

FPGA konektor	RS232 kabel	EPP funkcija	IN/OUT	UFC pinovi
4	17	Adress Strobe	OUT	N8
6	14	Data Strobe	OUT	L5
8	1	Write	OUT	N3
10	11	Wait	IN	M4
12	16	Reset	OUT	M3
14	10	Interrupt	IN	L4

2.4.2 Opis signala EPP konektora

U ovom dijelu nalazi se opis glavnih signala koje korisnimo prilikom slanja odnosno primanja podataka u komunikaciji između računala i FPGA.

nStrobe: Ovaj signal je registar u kojem se nalaze ili adrese ili podatci.

data 0-7: Podatci 0-7 sadržavaju addrese, podatke i RLE podatke. Podatkovni pinovi od 0 do 7 upotrebljavaju se i za slanje i primanje podataka tj. mogu se koristiti u oba smjera.

Busy: Ovaj signal nam govori dali periferalna jedinica može primiti podatak ili slati ona nam govori dali je sabirnica zauzeta ili je sabirnica slobodna za slanje odnosno primanje podataka.

Select: Signal select govori da je periferalna jedinica detektirana.

nAutoFd: Zahtjeva podatke od periferalne jedinice kada su svi uvjeti zadovoljeni odnosno kada je ciklus u nAck u povratnom smjeru. A kada je u odlaznom stanju tada signal govori da podatkovne linije sadržavaju ili adrese ili podatke.

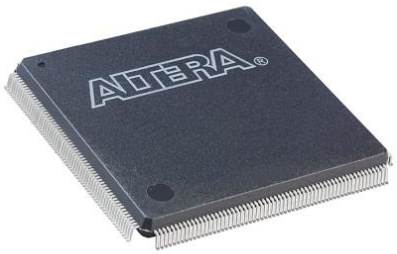
nFault: Ovaj signal Generira kada je došlo do reške prilikom slanja podataka.

nInit: Postavlja transferni smjer, odnosno ako je u visokom stanju tada se podatci vraćaju, a ako je u niskom stanju podatci se prosljeđuju.

nSelectIn: Ovaj signal govori nam u kakvom je stanju EPC mod.

3. FPGA SKLOPOVLJE

FPGA (engl. *Field programmable gate array*) je integrirani sklop dizajniran da bude konfiguriran od strane kupca ili dizajnera.



Sl. 3.1.*FPGA čip*

FPGA konfiguracija definira se programskim jezikom jezik za hardverski opis HDL (engl. *hardware description language*), taj programski jezik sličan je onome koji koriste za aplikacije specifičnih integrirani sklop ASIC (engl. *Applicationspecific integrated circuit*).

Ukratko ćemo objasniti što znače pojmovi, jezik za hardverski opis HDL (engl. *hardware description language*) te aplikacije specifičnih integriranih sklopova ASIC (engl. *Applicationspecific integrated circuit).*

Aplikacija specifičan integrirani sklop (ASIC) je integrirani sklop IC (engl. *integrated circuit*) prilagođene za određenu upotrebu, odnosno namijenjen za opće namjene koriste. Na primjer, čip dizajniran isključivo za pokretanje mobitela je ASIC. U elektronici,jezik za hardverski opis HDL (engl. *hardware description language*) je bilo koji jezik iz klase programskih jezika za formalni opis elektroničkih sklopova, također se sa jezikom za hardverski opis (HDL) mogu opisati i digitalni logički sklopovi. Jezik za Hardverski opis se koristi za pisanje izvršne specifikacije pojedinih dijelova hardvera.

Za manje dizajniranje i niže proizvodne volumene, FPGA može biti učinkovitiji i isplativiji od ASIC tehnologije u proizvodnji. Zbog toga se i više primjenjuje.

Na slici (**Sl. 3.2**. *Maketa Sparta3 Pegasus*) prikazana je digitalnu maketu (engl. *Digilent Xilinx Spartan3 Pegasus*) koja se koristiti prilikom slanja odnosno primanja podataka između računala i FPGA.

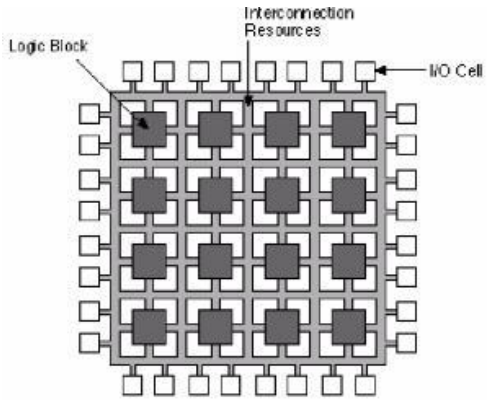
Sl. 3.2. *Maketa Spartan 3 Pegasus*

Da biste definirali ponašanje FPGA korisnik koristi jezik za opis hardvera (HDL) (engl. *hardware description language*) ili shematski dizajn. Pomoću HDL-a (engl. *hardware description language*) može biti lakše raditi s velikim strukturama jer je moguće da ih samo treba navesti brojčano, nije potrebe da se svaki komad crta rukom. Dok pak kod shematskog dizajna prednost je u tome sto možemo slagati sheme i tako dobivamo lakšu vizualizaciju dizajna.Jednom kada se napravi dizajn i provjera valjanosti proces je kompletan, binarna datoteka generirana. Datoteke se zatim prenose na FPGA preko različitih protokola, u našemu slučaju taj protokol je paralelni EEP protokol također se datoteke mogu prenjeti na vanjsku memoriju uređaja poput EEPROM.

Najčešći HDL (engl. *hardware description language*) programski jezici su VHDL i Verilog, pošto su ti programski jezici prilično složeni njihova složenost se pokušava smanjiti uvođenjem alternativnih jezika osnosno funkcija. Kako bi se pojednostavio dizajn složenih sustava u FPGAs, postoje knjižnice predefiniranih složenih funkcije i sklopova koji su testirani i optimizirani za ubrzanje procesa dizajna. Ti krugovi koji su unaprijed definirani obično se nazivaju IP jezgre, i mogu se dobiti od prodavača FPGA.

3.1 FPGA arhitektura

Iako svaki proizvođač FPGA koristi neku svoju specifičnu FGPA arhitekturu, sve one u osnovi su varijacija ahitekture prikazane na (Sl. 1. *FPGA arhitektura*). Arhitekturu čine: (a) logički blokovi, raspoređeni u dvodimenziono polje; (b) U/I blokovi, raspoređeni po obodu čipa i (c) programabilna sprežna mreža smještena u kanalima između logičkih blokova. Logički dijelovi služe za realizaciju logičkih funkcija, mreža omogućava povezivanje logičkih dijelova u cilju kreiranja složenijih funkcija, dok se putem U/I blokova ostvaruje povezanost internih resursa sa pinovima čipa. Dodatno, FPGA može sadržati i različite specijalizirane logičke resurse, kao što su ALU jedince, RAM memorija, dekoderi i dr.



Sl.3.3.*FPGA arhitektura*

Za veće brzine međusobno, neke FPGA arhitekture koriste duže usmjeravanje linije koje premošćuje višestruka logika blokova.

Povijesno, FPGAs su sporiji, manje energetski učinkovitiji i općenito teže je postići funkcionalnost u odnosu na ASIC (engl. *Applicationspecific integrated circuit)* tehnologiju. Prednost FPGA je u tome što posjeduje sposobnost nadogradnje funkcionalnosti nakon što se jednom isprogramira, tj. možemo raditi izmjene programskog koda ako to želimo u bilo kojemu trenutku nakon upotrebe FPGA čipa. Prednosti FPGA tehnologije također uključuje kraće vrijeme na tržištu, te niže inžinjerske troškovi prilikom izrade FPGA.FPGA se može koristiti za implementaciju bilo koje logičke funkcije koju je ASIC mogao obavljati.

FPGA tehnologija primjenjuje se u raznim područima kao naprimjer, obrada digitalnih signala, svemirskim i obrambenim sustavima, ASIC prototipovima, bioinformatici, računalnim hardverima, radio astronomiji i mnogim drugim područima. FPGA se sve više koristi u konvencionalnim visoko performansnim računalnih aplikacijama, kao što su FFT ili konvolucija na FPGA umjesto mikroprocesora.Tradicionalno, FPGA je rezerviran za određene aplikacije gdje je obujam proizvodnje vrlo malen.

3.2 FPGA usporedbe

Povijesno, FPGAs su sporiji, manje energetski učinkovitiji i općenito teže je postići funkcionalnost u odnosu na ASIC (engl. *Applicationspecific integrated circuit)* tehnologiju.

Prednost FPGA je u tome što posjeduje sposobnost nadogradnje funkcionalnosti nakon što se jednom isprogramira, tj. možemo raditi izmjene programskog koda ako to želimo u bilo kojemu trenutku nakon upotrebe FPGA čipa. Prednosti FPGA tehnologije također uključuje kraće vrijeme na tržištu, te niže inžinjerske troškovi prilikom izrade FPGA.

3.3 Primjena FPGA

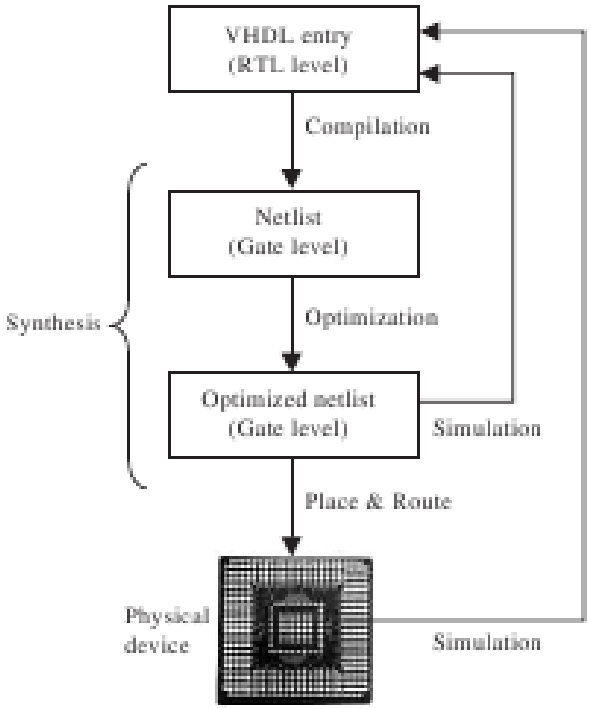
FPGAs može se koristiti za implementaciju bilo koje logičke funkcije koju je ASIC mogao obavljati. FPGA tehnologija primjenjuje se u raznim područima kao naprimjer, obrada digitalnih signala, svemirskim i obrambenim sustavima, ASIC prototipovima, bioinformatici, računalnim hardverima, radio astronomiji i mnogim drugim područima.FPGA se sve više koristi u konvencionalnim visoko performansnim računalnih aplikacijama, gdje se izvode računske jezgri, kao što su FFT ili konvolucija na FPGA umjesto mikroprocesora.

Tradicionalno, FPGA je rezerviran za određene aplikacije gdje je obujam proizvodnje vrlo malen.

4. DIZAJN EPP SUČELJA

U ovom radu koristio se VHDL programski jezik za programiranje FPGA.VHDL je programski jezik za opis elektroničkog sklopovlja (engl. *hardware description language*). Opisuje ponašajne karakteristike električnog sklopa ili sustava iz kojih se može izvesti fizički sklop ili sustav. VHDL je namijenjen u svrhu*sinteze* sklopovlja, a isto tako i za njihovu *simulaciju*. Premda se VHDL može u potpunosti simulirati, nisu sve konstrukcije podložne simulacijama. Naredbe VHDL jezika se izvode istodobno, tj. paralelno što je u suprotnosti s običnim programskim jezicima koji su sekvencijalni. Iz tog razloga se VHDL odnosi na *kod*, prije nego na *program*. U VHDL-u se sekvencijalno izvode samo naredbe smještene unutar funkcija PROCESS, FUNCTION ili PROCEDURE.

Kao što je ranije naglašeno, jedna od glavnih značajki VHDL-a je mogudnost sinteze elektroničkih krugova ili sustava unutar programabilnih logičkih sklopova (PLD ili FPGA) ili na ASIC logičkim sklopovima. Na slici (**Sl.4.1.** *Sažeti prikaz toka VHDL dizajna* **)** je prikazan sažeti oblik koraka projektiranja.



Sl.4.1.*Sažeti prikaz toka VHDL dizajna*

4.1 Programiranje EPP porta

Programiranje EPP-a počinje pisanje samog VHDL koda, koji je spremljen u datoteci s ekstenzijom *.vhd* te je istog imena kao i ENTITET (*engl. ENTITY*). Prvi korak u procesu sinteze jest *kompilacija* (*engl. compilation*). Kompilacija predstavlja konverziju višeg programskog VHDL jezika, koji opisuje sklopovlje na RTL nivou (engl. *Register Transfer Level*), u *netliste* (*engl. netlist*), na razini logičkih sklopova.

Sljedeći korak dizajniranja je optimizacija koja se izvodi na netlistama logičkih sklopova zbog brzine ili raspoložive površine. U ovoj fazi dizajn može biti podvrgnut simulaciji. Na kraju de tzv. *place-and-route* (*fitter*) softver generirati fizički razmještaj ili shemu PLD/FPGA čipa.

VHDL kod

U ovom dijelu opisan je VHDL kod,koji će biti opisan u više djelova kako bi se moglo što razumjeti bit zadatka.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

U prva četiri reda ovog *VHDL* koda deklarirali smo sve biblioteke koje smo koje ćemo koristiti u kodu. Paket *std_logic_1164* iz *ieee* biblioteke precizira višerazinski logički sustav,*std* je resursna biblioteka (tipovi podataka, text i/o itd.) za VHDL dizajnersko okružje.

```
30  entity EPP is
31  port
32  (
33      nDataStrobe : IN Std_Logic;
34      nAddrStrobe : IN Std_Logic;
35      nWri        : IN Std_Logic;
36      nReset      : IN Std_Logic;
37      Data        : INOUT Std_Logic_Vector(7 DOWNTO 0);
38      nWait       : OUT Std_Logic;
39      nAck        : OUT Std_Logic;
40      SysClk      : in Std_Logic;
41      Reset       : in Std_Logic;
42      Led         : out std_logic_vector(3 downto 0)
43
44  );
45  end EPP;
```

U entityju su definirani ulazni signali Sysclk, reset, Sysclk signal je glavni signal takta. Reset signal služi za glavni reset FSM-a, također imamo još definirane signale nDataStrobe_in i nAddreStrobe_in, data_inout to je signal vektor u iz kojega isčitavamo stanja, nWrite_in je signal koji određujestanje kada upisujemo ili ispisujemo podatke, nWait_out je također signal koji ima stanje iz kojega gledamo koji slijedi ciklus upis/ispis.

```
22  ARCHITECTURE Action OF EPP IS
23
24  component lpmramdq IS
25          PORT
26          (
27                  address      : IN STD_LOGIC_VECTOR (4 DOWNTO 0);
28                  clock        : IN STD_LOGIC ;
29                  data         : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
30                  wren         : IN STD_LOGIC ;
31                  q            : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
32          );
33
34  END component;
```

U ovom dijelu definiramo glavni dio koji se naziva arhitecture EPP gdje su definirani portovi i njihovi vektori a to su, address_in koja je u obliku vektora (4 downto 0) sljedeći vektor je data_in (7 downto 0). U type state su definirana stanja FSM-a, dok su registri stanja definirani kao st_curr (trenutno stanje) i st_next (sljedeće stanje).

```
36  signal Rst : std_logic;
37  signal nDataStrobeLatch,nAddrStrobeLatch,nWriLatch,nResetLatch : std_logic;
38  signal RegDataTempIn  : std_logic_vector(7 downto 0);
39  signal RegAddrTempIn  : std_logic_vector(7 downto 0);
40  signal RegDataTempOut : std_logic_vector(7 downto 0);
41  signal RegAddrTempOut : std_logic_vector(7 downto 0);
42  signal wrDataClk : std_logic;
43  signal DataOutLatch: std_logic_vector(7 downto 0);
```

U ovom dijelu programskog koda definirani su svi registry koji će se koristiti prilikom slanja i primanja podataka EPP porta, vidimo register RegDataTempIn koji ima veličinu (7 downto 0) također tu se nalaze registri RegAddrTemo_In, RegDataTemp_In, RegAddrTemo_Out, i dr.

```
79  begin
80      if Rst = '0' then
81          OE <= '0';
82          SelIn <= "11";
83          nWait <= '0';
84      elsif  rising_edge(SysClk) then
85      if (nWriLatch = '0' and nDataStrobeLatch = '0') then
86                  OE <= '0';
87                  SelIn <= "11";
88                  nWait <= '1';
89                  Data <= "ZZZZZZZZ";
90                  DataOutLatch <= Data;
91                  Led <= "0001";
92       elsif (nWriLatch = '1' and nDataStrobeLatch = '0') then
93                  OE <= '1';
94                  SelIn <= "10";
95                  Data <= "10101010";
96                  nWait <= '1';
97                  Led <= "0010";
```

Ovaj dio koda je glavni dio koda za slanje i primanje podataka slanje podataka između računala i FPGA. Kod razdvajamo na dva dijela, u jednom dijelul je pisanje I čitanje podataka a u drugom dijelu se nalazi pisanje I čitanje adresa. Dio koda *if (nWriLatch = '0' and nDataStrobeLatch = '0') then* tu počinje zapisivanje podataka epp porta prema FPGA, u dijelu *DataOutLatch <= Data;* paralelni port zapisuje podatke prema periferijalnim uređajima u našem slučaju to je FPGA. U dijelu *elsif (nWriLatch = '1' and nDataStrobeLatch = '0') then* počinje čitanje podatka od računala preko EPP porta, u dijelu *Data <= "10101010";* nalazi se podatak koji se čita.

```
 98          elsif (nWriLatch = '0' and nAddrStrobeLatch = '0') then
 99              OE <= '0';
100              SelIn <= "11";
101              nWait <= '1';
102              Led <= "0100";
103          elsif (nWriLatch = '1' and nAddrStrobeLatch = '0') then
104              OE <= '1';
105              SelIn <= "01";
106              nWait <= '1';
107              Led <= "1000";
108          else
109              OE <= '0';
110              SelIn <= "11";
111              nWait <= '0';
112              Led <= "1111";
113          end if;
114      end if;
115  end process;
```

Drugi dio koda u kojem se nalaze ciklusi pisanja I čitanja adresa. U dijelu koda *elsif (nWriLatch = '0' and nAddrStrobeLatch = '0') then* počinje zapisivanje adresa epp porta prema FPGA, dio koda *elsif (nWriLatch = '1' and nAddrStrobeLatch = '0') then*u njemu je definirano čitanje adresa sa računala pomoću EPP protokola.

```
100  process(SysClk, Rst)
101  begin
102      if Rst = '0' then
103          wrDataClk <= '1';
104          wrAddrClk <= '1';
105      elsif rising_edge (SysClk) then
106          wrDataClk <= nDataStrobeLatch or nWriLatch;
107          wrAddrClk <= nAddrStrobeLatch or nWriLatch;
108      end if;
109  end process;
```

U ovom dijelu koda realiziran je prikaz generiranja WrDataClk, WrAddrClk tu su ciklusi koji su definirani za vanjske odnosno periferijalne uređaje.

```
120   process(SysClk, Rst)
121   begin
122       if Rst ='0' then
123           RegDataTempIn <= "00000000";
124       elsif rising_edge (SysClk) THEN
125           if wrDataClk ='0' THEN
126               RegDataTempIn <= DataOutLatch;
127           else
128           end if;
129       end if;
130   end process;
```

U ovom dijelu koda definirano je ispisivanje podataka EPP porta prema izlazu odnosno prema periferijalnim uređajima. U dijelu koda RegDataTemp_In vidimo registar od 8-bitova i u njemu je definirano resetiranje podataka koji trebaju biti poslani prema izlazu.

Kompletan VHDL kod komunikacije računala između FPGA pomoću EPP () porta nalazi se u prilogu jedan.

5. ZAKLJUČAK

Zadatak ovog završnog rada je rješavanje problema FPGA komunikacijskog sučelja između računala i FPGA s pomoću paralelnog protokola EPP. U radu je opisan EPP protokol, njegov najvažniji dijelovi hardwerska i programska podrška. Predstavljen je projekt izrade komunikacijskog sučelja EPP na FPGA platformi. Komunikacija je uspostavljena sa računalom preko aplikacije izrađene u C++ programskom jeziku. Za provjeru slanja i primanja podataka izrađen je hardverski pokaznik sa LED diodama za prikaz stanja na podatkovnim i upravljačkim pinovima EPP porta.

Također FPGA čip je opisan VHDL programskim kodom, u kojem smo definirali komunikaciju slanja i primanja podataka. Tijekom rada objašnjena je izrada koda i najvažniji djelovi koda.

Prestavljena FPGA tehnologija, te su opisane njegove glavne karakteristike. Opisana je arhitektura FPGA čipa, njegova usporedba te primjena.

LITERATURA

[1] S. Šegvić, Uvod u programski jezik VHDL – Predavanja, FER, Zagreb, 2002 – 2003.

[2] Volnei A. Pedroni, Circuit Design with VHDL – Published by Massachusetts Institute of Technology, 2004.

[3] Circuit Design with VHDL, Volnei A. Pedroni – Published by John Wiley & Sons, Inc., Hoboken, New Jersey.

[4] Clive "Max" Maxfild, The Design Warrior' s Guide to FPGAs – Published by Mentor Graphics Corporatio and Xilinx, 2004.

SAŽETAK

Cilj ovog završnog rada je rješavanje problema FPGA komunikacijskog sučelja PC – FPGA s pomoću paralelnog protokola EPP. U radu je opisan EPP protokol, njegov najvažniji dijelovi hardverska i programska podrška. Također je opisana FPGA arhitektura i dizajn EPP-a pomoću programskog koda VHDL. U drugom poglavlju prestavljen je FPGA, te su opisane njegove glavne karakteristike. Opisana je arhitektura FPGA čipa, njegova usporedba te primjena u tehnologiji. Poglavlje tri sadrži opis i slike programskog dijela sustava za komunikaciju između računalnog sklopovlja i FPGA sklopovlja.

Ključne riječi: FPGA, Spartan-3 *FPGA* čip, VHDL, EPP (*Enhanced Parallel Port*)

ABSTRACT

The aim of this paper is the final problem solving FPGA communication interfaces PC - FPGA using parallel EPP protocol. The paper describes the EPP protocol, its most important parts hardwerska and software. It also describes the FPGA architecture and design of EPP using VHDL code. The second chapter describes the FPGA, and describes its main characteristics. It describes the architecture of FPGA chip, its comparison and application of the technology. Chapter three contains a description and pictures of the programming system for communication between computer hardware and FPGA hardware.

Keywords: FPGA, Spartan-3 *FPGA* čip, VHDL, EPP (*Enhanced Parallel Port*)

ŽIVOTOPIS

Dragan Slišković rođen 1987. Godine u Žepču, BiH. Gdje upisuje osnovnu školu i nakon završenog trećeg razreda osnovne škole seli u Hrvatsku, Đakovo. U Đakovu nastavlja sa osnovnim školovanjem i uspješno ga završava. Završetkom osnovne škole upisuje srednju strukovnu školu „Braće Radić“ također u Đakovu, srednju školu završava sa vrlo dobrim uspjehom. Godine 2006. prijavljuje se na raredbeni postupak upisa Elektrotehničkog fakulteta u Osijeku, uspješno polaže razredbeni postupak te upisuje Sveučilišni preddiplomski studij računarstva.

_____

Dragan Slišković

PRILOZI

Prilog 1: VHDL kod EPP-a

```
library ieee;
use ieee.std_logic_1164.all;
USE IEEE.Std_Logic_Unsigned.ALL;
USE IEEE.Std_Logic_Arith.ALL;

ENTITY MCU IS
PORT
(
   nDataStrobe : IN Std_Logic;
   nAddrStrobe : IN Std_Logic;
   nWri        : IN Std_Logic;
   nReset      : IN Std_Logic;
   Data        : INOUT Std_Logic_Vector(7 DOWNTO 0);
   nWait       : OUT Std_Logic;
   SysClk      : in Std_Logic;
   Reset       : in Std_Logic;
);
END MCU;
```

```
ARCHITECTURE Action OF MCU IS

component lpmramdq IS
	PORT
	(
		address		: IN STD_LOGIC_VECTOR (4 DOWNTO 0);
		clock		: IN STD_LOGIC ;
		data		: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
		wren		: IN STD_LOGIC ;
		q		: OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
	);

END component;

signal Rst : std_logic;
signal OE  : std_logic;
signal SelIn : std_logic_vector(1 downto 0);
signal nDataStrobeLatch,nAddrStrobeLatch,nWriLatch,nResetLatch : std_logic;
signal RegDataTempIn  : std_logic_vector(7 downto 0);
signal RegAddrTempIn  : std_logic_vector(7 downto 0);
signal RegDataTempOut : std_logic_vector(7 downto 0);
signal RegAddrTempOut : std_logic_vector(7 downto 0);
signal wrDataClk : std_logic;
signal wrAddrClk : std_logic;
signal DataOutLatch: std_logic_vector(7 downto 0);
```

```
begin
    Rst <= Reset;
process(Rst,SysClk)
begin
    if Rst = '0' then
        OE <= '0';
        SelIn <= "11";
        nWait <= '0';
    elsif  rising_edge(SysClk) then
        if (nWriLatch = '0' and nDataStrobeLatch = '0') then
            OE <= '0';
            SelIn <= "11";
            nWait <= '1';
            Data <= "ZZZZZZZZ";
            DataOutLatch <= Data;
            Led <= "0001";
        elsif (nWriLatch = '1' and nDataStrobeLatch = '0') then
            OE <= '1';
            SelIn <= "10";
            Data <= "10101010";
            nWait <= '1';
            Led <= "0010";
        elsif (nWriLatch = '0' and nAddrStrobeLatch = '0') then
            OE <= '0';
            SelIn <= "11";
```

```
            nWait <= '1';
            Led <= "0100";
        elsif (nWriLatch = '1' and nAddrStrobeLatch = '0') then
            OE <= '1';
            SelIn <= "01";
            nWait <= '1';
            Led <= "1000";
        else
            OE <= '0';
            SelIn <= "11";
            nWait <= '0';
            Led <= "1111";
        end if;
    end if;
end process;
process(SysClk, Rst)
begin
    if Rst = '0' then
        nDataStrobeLatch <= '1';
        nAddrStrobeLatch <= '1';
        nWriLatch        <= '1';
        nResetLatch      <= '1';
    elsif rising_edge (SysClk) then
        nDataStrobeLatch <= nDataStrobe;
        nAddrStrobeLatch <= nAddrStrobe;
        nWriLatch        <= nWri;
        nResetLatch      <= nReset;
```

```
    end if;
end process;
process(SysClk, Rst)
begin
    if Rst = '0' then
        wrDataClk <= '1';
        wrAddrClk <= '1';
    elsif rising_edge (SysClk) then
        wrDataClk <= nDataStrobeLatch or nWriLatch;
        wrAddrClk <= nAddrStrobeLatch or nWriLatch;
    end if;
end process;

process(SysClk, Rst)
begin
    if Rst ='0' then
        RegDataTempIn <= "00000000";
    elsif rising_edge (SysClk) THEN
        if wrDataClk ='0' THEN
            RegDataTempIn <= DataOutLatch;
        else
        end if;
    end if;
end process;

process(SysClk, Rst)
begin
```

```
    if Rst ='0' then
        RegAddrTempIn <= "00000000";
    elsif rising_edge (SysClk) then
        if wrAddrClk ='0' then
            RegAddrTempIn <= DataOutLatch;
        else
        end if;
    end if;
end process;

process (OE, SelIn, Data)
begin
end process;

u1:lpmramdq port map
(
            address       => "00010",
            clock        => SysClk,
            data        => Data,
            wren        => '1',
            q             => RegDataTempIn
);

end Action;
```

Prilog 2: C++ kod testiranje paralelnog kabela

```
#include <stdio.h>
#include <conio.h>

// kontrolni bitovi
#define WRITE_BIT          0x1          // ~C0
#define DATA_STROBE_BIT    0x2          // ~C1
#define INIT_BIT           0x4          //  C2
#define ADDR_STROBE_BIT    0x8          // ~C3

#define IRQ_ENABLE_BIT     0x10         //  C5
#define BI_DIRECTION       0x20         //  C6

typedef unsigned __int8 BYTE;           // 0 - 255

BYTE Idle  = WRITE_BIT & ~DATA_STROBE_BIT & ~ADDR_STROBE_BIT
INIT_BIT;                                       // 0x05
BYTE nInit= 0x01;
BYTE nDst= 0x07;
BYTE nAst= 0x0D;
BYTE inPst= 0x24;

// portovi
short _stdcall Inp32(short address);
void _stdcall Out32(short address, short data);
```

```
// adresse
short DATA_ADDR         = 0x378;
short STATUS_ADDR       = 0x379;
short CONTROL_ADDR      = 0x37a;
short EPP1_ADDR         = 0x37b;
short EPP2_ADDR         = 0x37c;

// funkcije
void SetAddress(BYTE address)
{
Out32(DATA_ADDR, address);
Out32(CONTROL_ADDR, nAst);
Out32(CONTROL_ADDR, Idle);
}

void WriteData(BYTE data)
{
Out32(DATA_ADDR, data);
Out32(CONTROL_ADDR, nDst);
Out32(CONTROL_ADDR, Idle);
}


BYTE ReadData()
{
BYTE data;
//Out32(CONTROL_ADDR, inPst);
```

```
//Out32(CONTROL_ADDR, inPst | DATA_STROBE_BIT);
data = (BYTE)Inp32(DATA_ADDR);
Out32(CONTROL_ADDR, inPst & ~DATA_STROBE_BIT);
Out32(CONTROL_ADDR, Idle);
return data;
}

void Reset()
{
Out32(CONTROL_ADDR, INIT_BIT | BI_DIRECTION);
}

void Delay()
{
for(int i = 10000; i>0; i--)
{       for(int j=1000; j>0; j--)
{
int A = 9;
A = A * A;
A/= A;
}
}
}
```

```
// test program
void main()
{
BYTE data1, data2;
Reset();
data1 = 200;
while(true)
{
WriteData(data1);
data2 = ReadData();
Delay();
printf("poslano %3d procitano %3d\n", data1, data2);
data1++;
}
getch();
}
```