SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 5397

PROJEKTIRANJE I IZVEDBA DIGITALNIH SUSTAVA U CPLD I FPGA TEHNOLOGIJI

Matija Jurišić

Zagreb, lipanj 2018.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA ZAVRŠNI RAD MODULA

Zagreb, 9. ožujka 2018.

ZAVRŠNI ZADATAK br. 5397

Pristupnik: **Matija Jurišić (0036494683)**
Studij: Elektrotehnika i informacijska tehnologija
Modul: Elektroničko i računalno inženjerstvo

Zadatak: **Projektiranje i izvedba digitalnih sustava u CPLD i FPGA tehnologiji**

Opis zadatka:

U okviru završnog rada potrebno je osmisliti, pripremiti i dokumentirati primjere projektiranja i izvedbe jednostavnih digitalnih logičkih sklopova korištenjem CPLD i FPGA programabilnih sklopova. Ovi primjeri će, ako se pokažu edukativno korisnim, biti iskorišteni kao zamjena za postojeće laboratorijske vježbe iz prvog ciklusa predmeta Ugradbeni računalni sustavi. Ove vježbe su do sada obuhvaćale postupke projektiranja jednostavnih kombinacijskih i sekvencijalnih digitalnih sklopova uz primjenu jednostavnih PAL sklopova programiranih u programskom alatu PALASM. Druga vježba bila je posvećena projektiranju logičkih automata s konačnim brojem stanja. U završnom radu treba predložiti modernizaciju postojećih vježbi korištenjem suvremenih sklopovskih programabilnih platformi i odgovarajućih razvojnih alata za sintezu i simuliranje. Istražiti mogućnost i pogodnost korištenja jezika za opis sklopovlja (npr. VHDL ili Verilog) uvažavajući očekivano predznanje studenata i ograničeno vrijeme studenta raspoloživo za pripremu i provedbu laboratorijske vježbe.

Zadatak uručen pristupniku: 16. ožujka 2018.
Rok za predaju rada: 15. lipnja 2018.

Mentor:
Prof. dr. sc. Davor Petrinović

Djelovođa:
Prof. dr. sc. Dražen Jurišić

Predsjednik odbora za završni rad modula:
Prof. dr. sc. Mladen Vučić

SADRŽAJ

1. Uvod

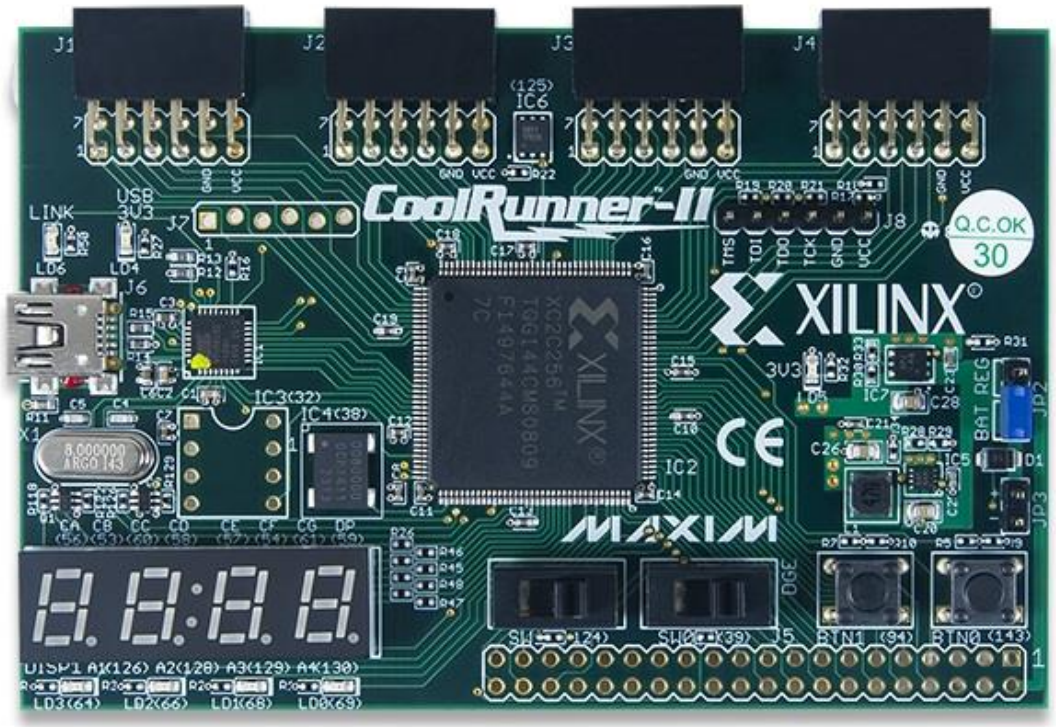
U ovom završnome radu je riješen zadatak projektiranja i izvedbe jednostavnih digitalnih logičkih sklopova korištenjem CPLD I FPGA programabilnih sklopova. Rad sadrži primjere laboratorijskih vježbi iz predmeta Ugradbeni računalni sustavi riješene korištenjem jezika za opis sklopovlja VHDL. Time je predložena modernizacija postojećih vježbi koje su do sada bile riješene primjenom jednostavnih PAL sklopova programiranih u programskom alatu PALASM. Predloženi primjeri su ostvareni korištenjem ISE Design Suite razvojnog alata za sintezu i simuliranje. Za obavljanje te funkcije u radu i u vježbi korištena je pločica CoolRunner-II CPLD Starter Board.

U drugom poglavlju rad se bavi opisom pločice i njegovom konfiguracijom, te se u sljedećem poglavlju opisuje sklop Coolrunner-II CPLD i sve njegove karakteristike. Ostala poglavlja opisuju četiri primjera zadataka iz laboratorijskih vježbi i pojašnjenje načina njihovog rješavanja korištenjem VHDL koda i dodjeljivanja priključaka. U četvrtom poglavlju je također opisan način rukovanja ISE Design Suite-om.

2. Opis pločice

2.1 Općenito

Coolrunner-II Starter Board je platforma sa USB priključkom za Xilinx-ov Coolrunner-II CPLD. Pločica sadrži CPLD na kojeg je spojen programabilni oscilator, USB2 port za napajanje pločice i programiranje CPLD-a, JTAG konektore, nekoliko ulazno-izlaznih uređaja, regulator napajanja i jednožični DS28E01Q EEPROM. Na pločici je također ugrađeno pet dodatnih konektora koji omogućuju dostupnost 64 ulazno-izlaznih signala iz CPLD-a vanjskom krugu. Slika pločice i njen sadržaj je prikazan na slici 2.1.



Slika 2.1 Sadržaj pločice

2.2 Konfiguracija

Kako bi pločica bila u mogućnosti izvoditi bilo kakve funkcije, korisnik prvo mora programirati CPLD pločicu. Izvorna datoteka koja sadrži opis funkcija sklopa se ostvaruje shematski ili nekim od HDL jezika. Iz izvorne datoteke se generira potrebna datoteka za pločicu pomoću Xilinx-ova programa ISE Design Suite. Nakon toga konfiguracijsku datoteku prenosimo na pločicu pomoću USB kabela i Xilinx-ova iMPACT softvera ili pomoću vanjskog programabilnog kabela. Coolrunner-II ima ugrađen EEPROM, što znači da se funkcija pločice može brisati i ponovno programirati spajanjem na računalo i prenošenjem nove funkcije na pločicu. Također sadrži flash memoriju, koja čuva podatke kada je pločica isključena sa napajanja.

2.3 Izvor napajanja

Coolrunner-II pločica se napaja preko integriranog USB porta ili preko vanjskog izvora napajanja spojenog na konektor JP3. Položaj premosnika na konektoru JP2 određuje način napajanja pločice. U položaju REG pločica se napaja preko USB porta, a u položaju BAT preko vanjskog izvora. Kada koristimo vanjski izvor moramo dovesti izvor napajanja od 4.5 V do 9 V na konektor JP3. Vanjsko napajanje se dovodi na regulator koji tada daje napajanje od 1.8 V za CPLD i 3.3 V za ulazno-izlazne konektore.

2.4 Oscilatori

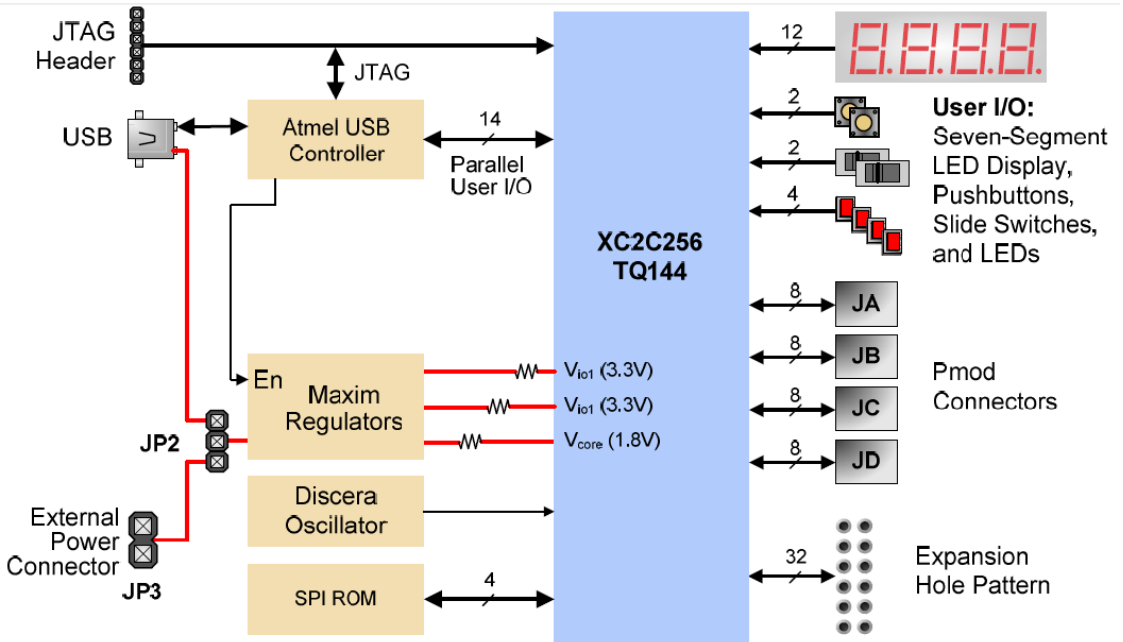
Pločica sadrži oscilator fiksne frekvencije koji stvara signal od 8 MHz, a koji je spojen na CPLD. Na pločicu je moguće spojiti i standardni kristalni oscilator na lokaciju 32, odnosno na IC3.

2.5 Ulazno-izlazni uređaji

Na pločici se nalaze dvije sklopke i dvije tipke kao ulazi, te četiri svjetleće diode i četiri crvena 7-segmentna zaslona kao izlazi. Također sadrži tri diode koje nam pokazuju status USB-a i njegovog napajanja, te status napajanja pločice. Svjetleće diode su aktivne na visoku razinu.

2.6 Dodatni konektori

Uz već prije navedene konektore, Coolrunner-II ima četiri 12-pinska priključka. Svaki od tih konektora ima dva priključka s naponom visoke razine V_{DD} i dva priključka s naponom GND. Isto tako, svaki konektor može primiti jedan 12-pinski Pmod ili dva 6-pinska Pmoda. Pločica također sadrži i 40 pinski priključak s tri signala napajanja i 37 pojedinačnih ulazno-izlaznih signala.

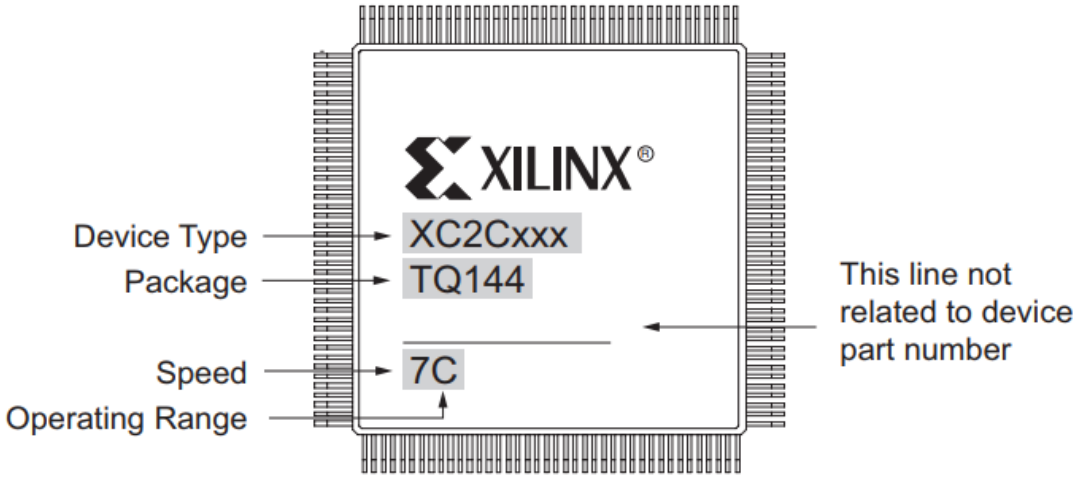


Slika 2.2 Shema pločice

3. Sklop CPLD XC2C256 TQ144

3.1 Opis sklopa

XC2C256 Coolrunner-II CPLD ima vrlo dobra svojstva uz malu potrošnju snage. Proizvođač sklopa je tvrtka Xilinx. CPLD sklop se nalazi u TQ144 ili Thin Quad Flat Pack kućištu. Kućište je veličine 20mm x 20mm i sadrži 118 ulazno-izlaznih priključaka s razmakom od 0.5 mm između njih. Sklop je napravljen za rad na naponu od 1.8 V, ali ulazi i izlazi mogu raditi na više naponskih razina od 2.5 V do 3.3 V.



Slika 3.1 Oznake na sklopu

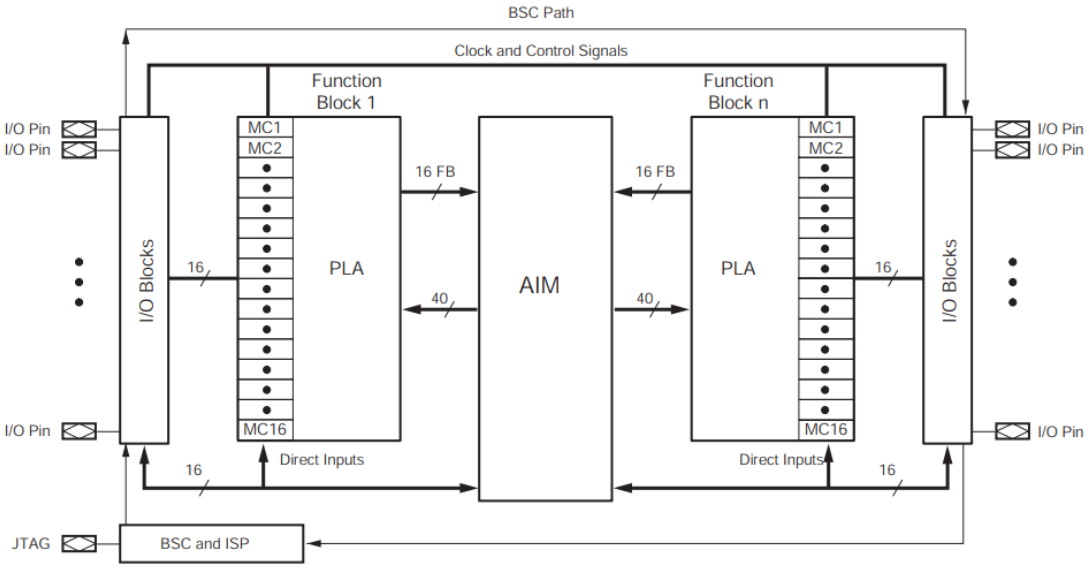
3.2 Karakteristike sklopa

Neka od sistemskih svojstva CPLD-a su: upravljanje malom snagom pomoću DataGATE-a, CoolCLOCK za smanjenu potrošnju uređaja, Schmittov okidni sklop kroz koji opcionalno mogu dolaziti ulazi, ulazno-izlazni blokovi, sklop za

kašnjenje, djelitelj takta i DualEDGE tehnologija. Sklop također ima PLA arhitekturu, makroćelije u kojima se obavlja logička funkcija, te *open drain* izlaze.

3.3 Opis arhitekture

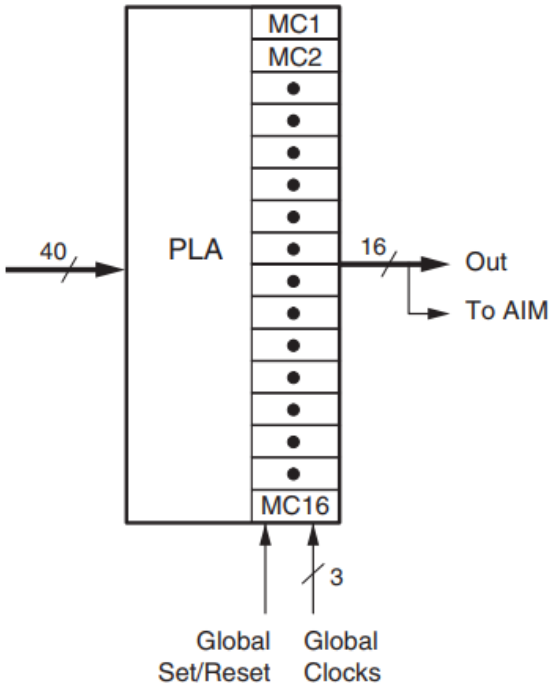
Arhitektura sklopa je tradicionalna CPLD arhitektura izvedena kombiniranjem makroćelija u funkcijske blokove međusobno povezane sa globalnom matricom *Xilinx Advanced Interconnect Matrix (AIM)*. Funkcijski blokovi koriste konfiguraciju sa programabilnim logičkim poljima ili *Programmable Logic Array (PLA)* koja omogućuje dijeljenje logičkih produkata među makroćelijama u funkcijskim blokovima. Sami sklop sadrži 256 makroćelija, a svaki funkcijski blok sadrži 16 makroćelija. Promjene u dizajnu izvode se jednostavno, pomoću upravljanja programabilnim logičkim poljima u svakom funkcijskom bloku.



Slika 3.2 Arhitektura Coolrunner-II CPLD-a

3.4 Funkcijski blokovi

Kao što je već navedeno, funkcijski blokovi sklopa sadrže 16 makroćelija sa 40 ulaznih mjesta za signale za stvaranje i povezivanje logičkih izraza. Svi funkcijski blokovi u sklopu su identični. Na visokoj razini funkcijskih blokova logički produkti su realizirani korištenjem PLA. Bilo koji logički izraz može se priključiti bilo kojem OR sklopu unutar funkcijskog bloka i makroćelija. Unutrašnja struktura omogućava da svaka logička funkcija sadržava do 56 logičkih produkata. Nadalje, isti logički izrazi mogu se koristiti u više OR sklopova unutar makroćelija. Svaki izraz dolazi sa jednakim vremenom kašnjenja.



Slika 3.3 Funkcijski blok

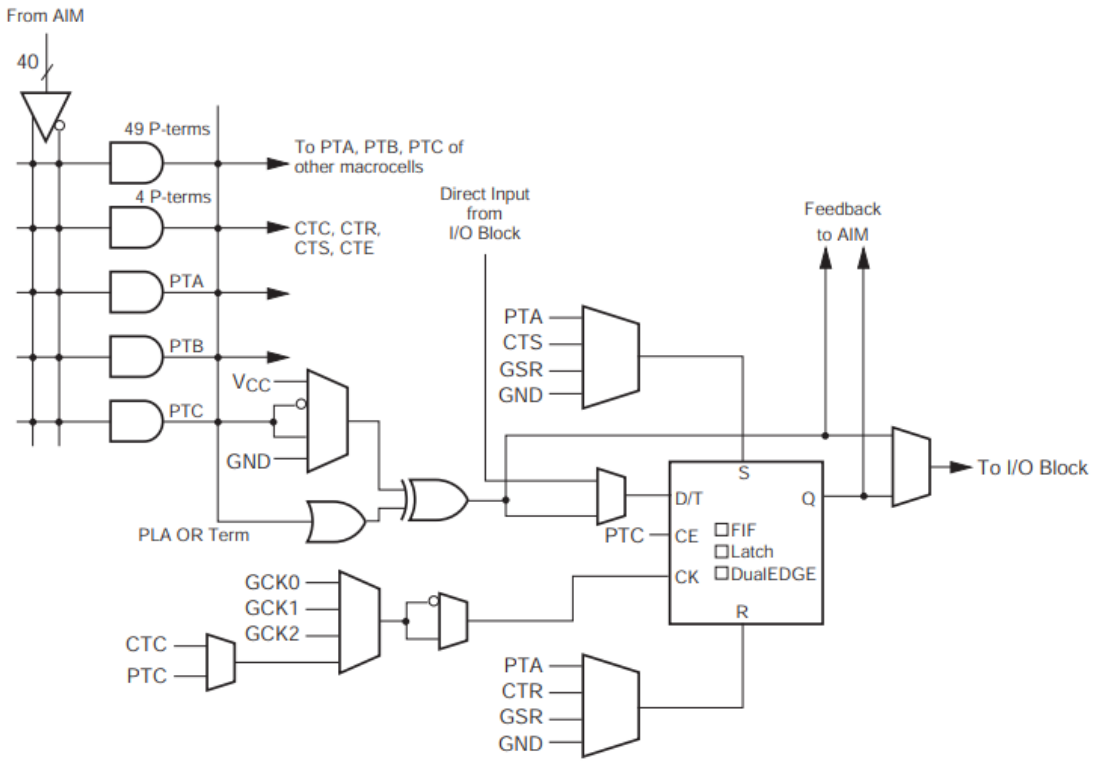
3.5 Makroćelije

Korisnik u makroćeliji može dodati sumu produkata (SOP) do 40 članova i 56 logičkih produkata unutar jednoj funkcijskog bloka. Makroćelija može dalje koristeći XOR sklop kombinirati SOP izraze iz PLA sklopa. Logička funkcija

makroćelije može biti čista kombinatorna ili registarska s memorijskim elementom D ili T bistabilom, ili pak s D *latchom* (*transparent latch*). Svaki bistabil može okidati na rastući brid takta. Također, DualEDGE tehnologija omogućuje okidanje na rastući i padajući brid takta zbog uštede energije.

Na slici 3.4 s lijeve strane vidimo PLA, dok se bistabil nalazi s desne strane s nazivima njegovih vrsta. Makroćelije još sadrže multipleksore sa ulazima za resetiranje i za omogućavanje izlaza ili *output enable.*

Ti ulazi se dalje dijele na podešavanje signala iz produktnog izraza (PTA), kontrolnog izraza (CTS), nijednog uvjeta (GND) ili globalnog omogućavanja odnosno resetiranja (GSR). Izrazi za kontrolu koji se koriste kada treba kontrolirati više makroćelija istim logičkim izrazom postoje za asinkrono postavljanje (CTS) i asinkrono resetiranje (CTR), za izvor takta (CTC) i za omogućavanje izlaza (CTE).



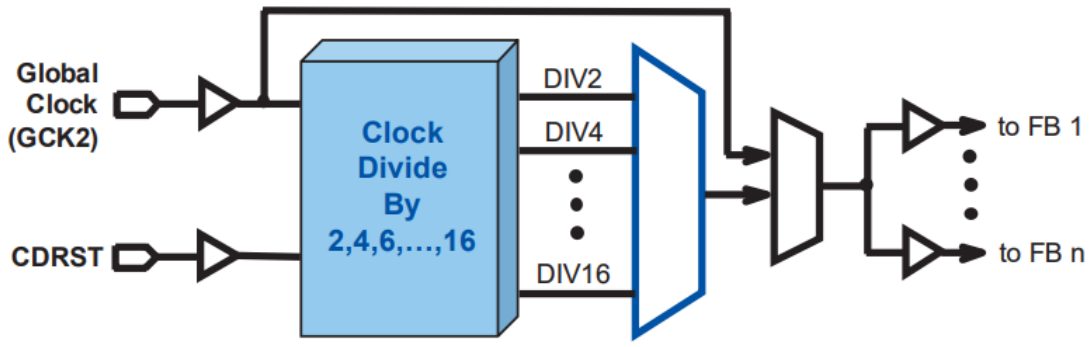
Slika 3.4 Makroćelija Coolrunner-II CPLD-a

3.6 *Advanced Interconnect Matrix (AIM)*

Globalna matrica ili *Advanced Interconnect Matrix* isporučuje do 40 signala svakom funkcijskom bloku za stvaranje logike. Rezultati funkcijskih blokova i ulazi u sklop sa pinova se vraćaju kroz matricu i dalje prosljeđuju ostalim funkcijskim blokovima. Prednost AIM-a je minimiziranje propagacijskog kašnjenja i potrošnje snage.

3.7 Dijeljenje signala takta

Sklop za dijeljenje takta pruža mogućnost dijeljenja ulaznog takta, a onda i globalnu distribuciju podijeljenog takta svim makroćelijama. Djelitelj takta također pruža i dodatnu uštedu energije pomoću tehnologije DualEDGE, koja je već spomenuta kod makroćelija. Sklop za dijeljenje takta je spojen na globalnom ulazu takta GCK2 i može dijeliti takt sa 2, 4, 6, 8, 10, 12, 14 i 16. Uz navedene mogućnosti, sklop sadrži i sinkroni reset, pod nazivom CDRST, koji za visoku razinu onemogućuje izlaz iz djelitelja takta nakon trenutnog ciklusa, a za nisku djelitelj takta postaje aktivan na sljedeći ulaz takta GCK2. Sklop za dijeljenje ima ugrađen i sklop za kašnjenje koji odgađa izlaz iz djelitelja za jedan ciklus.



Slika 3.5 Sklop za dijeljenje takta

4. Primjer 1

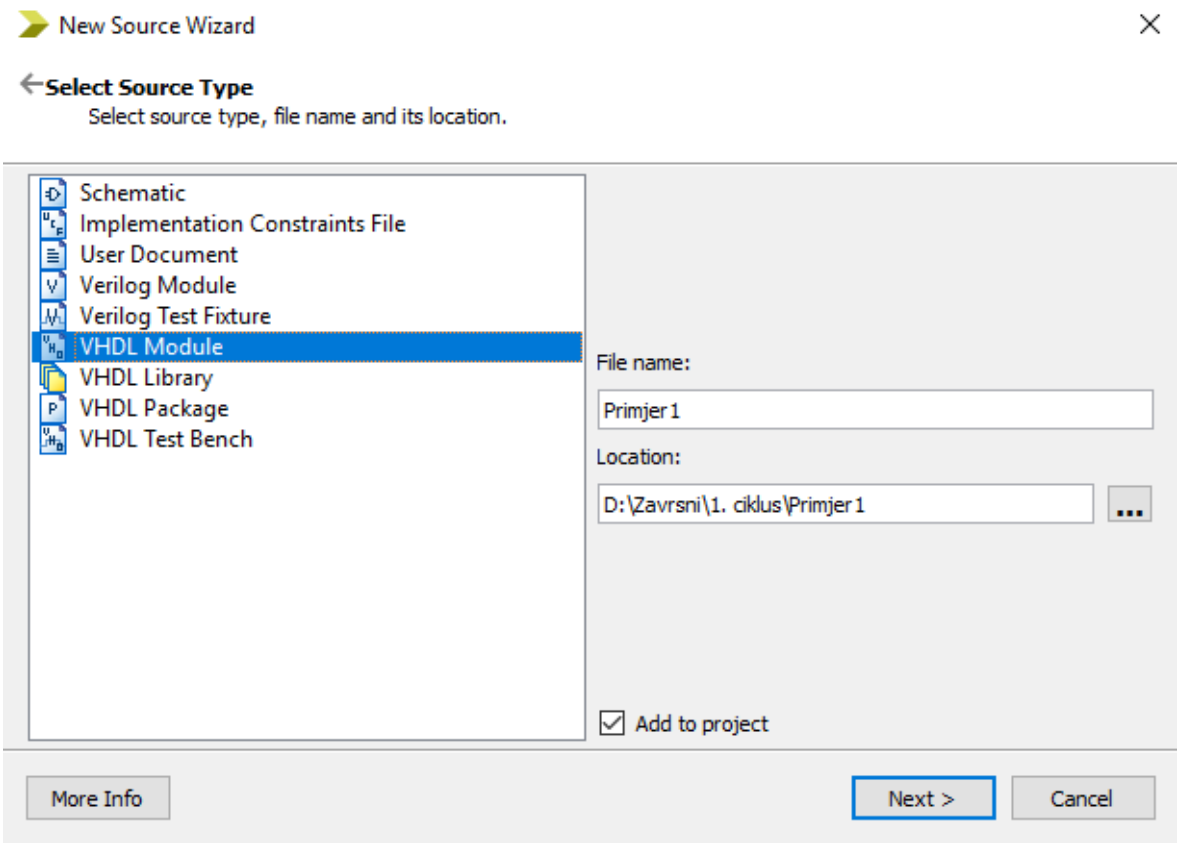
4.1 Zadatak

Treba realizirati enkoder s 4 ulaza na 2 izlaza. Enkoder je sklop koji na temelju jedne od 2^n ulaznih linija generira podatak na n izlaznih bita. U Primjeru 1. stanja na 4 ulazne linije bit će enkodirane kao 2-bitni podatak i prikazane na dvije izlazne linije.

Slika 4.1 Enkoder 4/2

4.2 Kreiranje vhd datoteke

Nakon dodavanja novog projekta u ISE Design Suite-u i postavljanja HDL-a kao *top level source type-a* još je potrebno odabrati sklop na kojem se radi i konfigurirati ga. Tek nakon toga se dodaje nova *source* datoteka sa *New source* i odabirom *VHDL Module* kao na slici 4.2.



Slika 4.2 Dodavanje nove *source* datoteke

Nakon što je potvrđen odabir sa *Next*, u sljedećem prozoru potrebno je potvrditi *Entity name* i *Architecture name* ponovo sa *Next* i time je stvorena nova *source* datoteka.

4.3 VHDL kod

U novostvorenoj vhd datoteci rad sklopa se opisuje koristeći jedan od HDL jezika, odnosno u ovom slučaju VHDL. Opis sklopa u VHDL-u sastoji se od dva dijela, sučelja i tijela opisa arhitekture. Sučelje se opisuje u prvom dijelu, odnosno

na početku i ono se naziva entitet ili *entity*. Unutar *entity* dijela koda se definiraju ulazno izlazni signali. Svaki ulaz i izlaz sklopa je definiran s imenom, tipom podataka i načinom rada. Načini rada mogu biti ulazni, izlazni, dvosmjerni i registarski. Također, sami broj ulaza i izlaza sklopa mora biti konstantan, a njihov tip može imati maksimalno jedno dimenziju, to jest može biti vektor. U Primjeru 1 se koristi četiri ulaza u enkoder, koji su označeni sa jednim D vektorom od četiri člana. Slično su ostvarena i dva izlaza iz enkodera sa C vektorom od dva člana. Uz to je korišten i jedan izlaz ANODA, koji će kasnije biti potreban za odabir kojeg se od četiri 7-segmentih zaslona koristi. Cijeli opis sučelja sklopa se može vidjeti na slici 4.3.

```
entity primjer1 is port (
      D : in std_logic_vector (3 downto 0);
      C : out std_logic_vector (1 downto 0);
      ANODA : out std_logic
      );
end primjer1;
```

Slika 4.3 *entity* opis sklopa

U drugom dijelu opisa sklopa se opisuje arhitektura sklopa i način ponašanja sklopa. Tijelo opisa arhitekture se još naziva *architecture*. Unutar *architecture* bloka nalazi se model koji opisuje ponašanje sklopa. U njemu se mogu deklarirati funkcije, procedure, interne signale, te sekvencijalnu ili funkcijsku logiku. Za ovaj primjer u njemu je potrebno ostvariti ponašanje sklopa po tablici na slici 4.4.

ULAZI				IZLAZI	
D3	D2	D1	D0	C0	C1
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

Slika 4.4 Tablica istinitosti

Za opis takvoga načina rada je korištena *case* naredba. Prije toga se izlaz ANODA, koji je spojen na jedan od 7-segmentnih zaslona, postavlja u '0' kako bi taj izlaz bio aktivan. Također je potrebno invertirati ostale izlaze i dva gumba koji su isto tako aktivni u '0'. Cijeli opis arhitekture sklopa nalazi se na slici 4.5.

```
architecture Behavioral of primjer1 is

begin

    process (D)
        begin
            ANODA <= '0';
            case D is
                when "1011" => C <= "11";
                when "0111" => C <= "01";
                when "0001" => C <= "10";
                when "0010" => C <= "00";
                when others => C <= "11";
            end case;
      end process;

end Behavioral;
```

Slika 4.5 *architecture* opis sklopa

4.4 Kreiranje ucf datoteke i dodjela priključaka

Nakon što je opisan rad sklopa u vhd datoteci, njoj se dodaje ucf datoteka u kojoj se ulazima i izlazima pridjeljuju fizički priključci koji će ih predstavljati. To se radi tako da se opet pritisne *New Source* i u prozoru kao na slici 4.2 ovoga puta odabere *Implementation Constraints File*. Za pridjeljivanje fizičkih priključaka potrebno je pogledati na shemu pločice gdje su označeni njihovi brojevi. Na slici 4.6 je sadržaj ucf datoteke za Primjer 1.

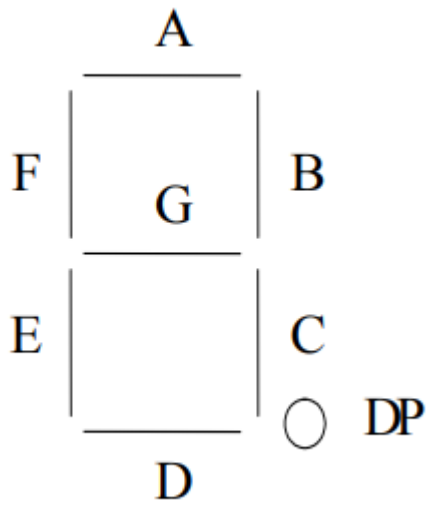
```
NET "D[3]" LOC="P124";
NET "D[2]" LOC="P39";
NET "D[1]" LOC="P94";
NET "D[0]" LOC="P143";

NET "C[0]" LOC="P54";
NET "C[1]" LOC="P53";

NET "ANODA" LOC="P126";
```

Slika 4.6 dodjeljivanje priključaka

5. Primjer 2

5.1 Zadatak

Treba realizirati funkciju prikaza 4-bitnih heksadecimalnih brojeva (0-9, A, B, C, D, E, F) na 7-segmentnom prikazivaču koji ima i decimalnu točku kao osmi segment. Raspored segmenata na prikazivaču prikazan je na slici 5.1. Treba realizirati i funkciju testiranja svih segmenata, tj. da se na zahtjev upale svi segmenti. Također provesti i simulaciju test vektorima.



Slika 5.1 7-segmentni prikazivač s decimalnom točkom

5.2 VHDL kod

Za realizaciju ove funkcije prikaza broja, u sučelju su potrebna četiri ulaza, osam izlaza za segmente prikazivača i još jedan izlaza za anodu. Četiri ulaza je ostvareno sa D vektorom od četiri člana, a osam izlaza sa S vektorom od isto toliko članova. Odabir prikazivača je ostvaren sa izlazom ANODA.

U opisu arhitekture sklopa potrebno je ostvariti ponašanje sklopa po tablici na slici 5.2.

ULAZI					IZLAZI							
/TST	D[3]	D[2]	D[1]	D[0]	S7	S6	S5	S4	S3	S2	S1	S0
0	X	X	X	X	1	1	1	1	1	1	1	1
1	0	0	0	0	1	1	1	1	1	1	0	0
1	0	0	0	1	0	1	1	0	0	0	0	0
1	0	0	1	0	1	1	0	1	1	0	1	0
1	0	0	1	1	1	1	1	1	0	0	1	0
1	0	1	0	0	0	1	1	0	0	1	1	0
1	0	1	0	1	1	0	1	1	0	1	1	0
1	0	1	1	0	1	0	1	1	1	1	1	0
1	0	1	1	1	1	1	1	0	0	0	0	0
1	1	0	0	0	1	1	1	1	1	1	1	0
1	1	0	0	1	1	1	1	1	0	1	1	0
1	1	0	1	0	1	1	1	0	1	1	1	0
1	1	0	1	1	0	0	1	1	1	1	1	0
1	1	1	0	0	1	0	0	1	1	1	0	0
1	1	1	0	1	0	1	1	1	1	0	1	0
1	1	1	1	0	1	0	0	1	1	1	1	0
1	1	1	1	1	1	0	0	0	1	1	1	0

Slika 5.2 Tablica istinitosti

U tablici postoji i dodatni ulaz /TST, koji služi za testiranje svih segmenata na izlazima i koji je zadan u opisu zadatka. U ovom rješenju testiranje izlaza neće biti ostvareno jer korišteni Coolrunner-II CPLD ima samo četiri ulaza i ne može se spojiti peti ulaz za /TST. Također u zadatku zadanu simulaciju test vektorima nije potrebno napraviti u ISE Design Suite-u. Za opis ponašanja sklopa opet je korištena *case* naredba kao u Primjeru 1. Isto tako, sve izlaze i dva ulaza za gumbe potrebno je invertirati jer su aktivni u '0'. Opis arhitekture sklopa za Primjer 2 nalazi se na slici 5.3.

```
architecture Behavioral of primjer2 is

begin

    process (D)
        begin
            ANODA <= '0';
            case D is
                when "0011" => S <= "00000011";
                when "0010" => S <= "10011111";
                when "0001" => S <= "00100101";
                when "0000" => S <= "00001101";
                when "0111" => S <= "10011001";
                when "0110" => S <= "01001001";
                when "0101" => S <= "01000001";
                when "0100" => S <= "00011111";
                when "1011" => S <= "00000001";
                when "1010" => S <= "00001001";
                when "1001" => S <= "00010001";
                when "1000" => S <= "11000001";
                when "1111" => S <= "01100011";
                when "1110" => S <= "10000101";
                when "1101" => S <= "01100001";
                when "1100" => S <= "01110001";
                when others => S <= "XXXXXXXX";
            end case;
    end process;
end Behavioral;
```

Slika 5.3 *architecture* opis sklopa

5.3 Dodjeljivanje priključaka

Sadržaj ucf datoteke i pridjeljivanje fizičkih priključaka za Primjer 2 je prikazano na slici 5.4.

```
NET "D[3]" LOC = "P124";
NET "D[2]" LOC = "P39";
NET "D[1]" LOC = "P94";
NET "D[0]" LOC = "P143";

NET "S[7]" LOC = "P56";
NET "S[6]" LOC = "P53";
NET "S[5]" LOC = "P60";
NET "S[4]" LOC = "P58";
NET "S[3]" LOC = "P57";
NET "S[2]" LOC = "P54";
NET "S[1]" LOC = "P61";
NET "S[0]" LOC = "P59";

NET "ANODA" LOC = "P126";
```

Slika 5.4 Primjer 2 ucf datoteka

6. Primjer 3

6.1 Zadatak

Treba realizirati 3-bitno sinkrono brojilo na način da se brojanje brojila vidi na 7-segmentnom prikazivaču kao paljenje segmenata: A (najznačajniji bit), G i D (najmanje značajan bit) ovisno o stanju brojila. Brojilo se resetira (svi izlazi u 0) dovođenjem aktivnog signala (logičke 0) na ulaznu liniju /RES. Brojilo broji kada je aktivna ulazna linija CE (aktivna u 1), u protivnom ostaje u prethodnom stanju. Treća ulazna linija UD definira da li brojilo broji prema naprijed (ako je u logičkoj 1) ili unazad (stanje logičke 0).

6.2 Rješenje

Zadatak se rješava pomoću 3-bitnog sinkronog brojila i *debouncing* sklopa. Sinkrono binarno brojilo je sekvencijalni sklop koji okida na rastući brid signala takta, odnosno mijenja stanja na izlazu prema nekom prethodno definiranom pravilu. Ta pravila se za sekvencijalne sklopove prikazuju u obliku tablica prelaska u kojoj se očitava iz kojeg stanja se prelazi u koje novo stanje.

ULAZI			TRENUTNO STANJE IZLAZA			SLIJEDEĆE STANJE IZLAZA		
/RES	CE	UD	B2	B1	B0	B2	B1	B0
0	X	X	X	X	X	0	0	0
1	0	X	B_{20}	B_{10}	B_{00}	B_{20}	B_{10}	B_{00}
1	1	1	0	0	0	0	0	1
1	1	1	0	0	1	0	1	0
1	1	1	0	1	0	0	1	1
1	1	1	0	1	1	1	0	0
1	1	1	1	0	0	1	0	1
1	1	1	1	0	1	1	1	0
1	1	1	1	1	0	1	1	1
1	1	1	1	1	1	0	0	0
1	1	0	0	0	0	1	1	1
1	1	0	0	0	1	0	0	0
1	1	0	0	1	0	0	0	1
1	1	0	0	1	1	0	1	0
1	1	0	1	0	0	0	1	1
1	1	0	1	0	1	1	0	0
1	1	0	1	1	0	1	0	1
1	1	0	1	1	1	1	1	0

Slika 6.1 Tablica istinitosti za 3-bitno brojilo

Uz sinkrono brojilo korišten je i *debouncing* sklop za rješavanje problema istitravanja kontakta do kojeg dolazi pri generiranju takta pomoću gumba. On funkcionira tako da njegov izlaz poprima vrijednost ulaza tek nakon što je ulaz u sklop dovoljno dugo nepromijenjen.

6.3 VHDL kod

Za implementaciju rješenja u sučelju su korištena četiri ulaza, jedan ulaz za oscilator na pločici i četiri izlaza za segmente prikazivača i anodu. Cijeli *entity* opis sklopa je na slici 6.2.

```
entity primjer3 is
    port(   CLK : in std_logic;
            UD, CE : in std_logic;
            not_RES, BTN_clk : in std_logic;
            ANODA : out std_logic;
            B : out std_logic_vector (2 downto 0) --  B(2) = A , B(1) = G , B(0) = D
        );
end primjer3;
```

Slika 6.2 opis sučelja sklopa

U *architecture* bloku je potrebno ostvariti ponašanje sklopa po slici 6.1 uz korištenje *debouncing* sklopa. Na početku treba definirati konstantu COUNT_MAX koja je iznosom jednaka frekvenciji oscilatora pločice od 8 MHz. Ta konstanta predstavlja granicu do koje će brojač brojati kada je pritisnut gumb za davanje takta, a treba provjeriti dali je još uvijek pritisnut ili je došlo do istitravanja. Uz to, navedeni su signal za brojač, dvije vrste stanja *debouncing* sklopa i signal Bint koji ima istu vrijednost kao izlaz B, ali služi kako bi nad njim mogli obavljati aritmetičke operacije. Također, početno stanje je definirano kao idle odnosno

čekanje da netko pritisne gumb za davanje takta.

U procesu se u slučaju resetiranja sklopa stanje postavlja u idle, a svi izlazni segmenti na pokazivaču se gase. Inače, kada dođe do rastućeg brida signala takta, potrebno je čekati pritisak gumba za davanje takta BTN_clk, kako bi se prešlo u stanje wait_time. U tom stanju se provjerava dali je gumb stvarno pritisnut ili je došlo do istitravanja pomoću već opisanog postupka. Kada je utvrđeno da je gumb stvarno pritisnut, ostvaruje se funkcija zadatka korištenjem niza *if-else* naredbi.

```
process (not_RES, CLK, CE, UD)

begin
    ANODA <= '0';
    if not_RES = '0' then
        state <= idle;
        Bint <= "111";
    elsif rising_edge(CLK) then
        case state is
            when idle =>
                if BTN_clk = '0' then
                    state <= wait_time;
                else
                    state <= idle; -- wait until button is pressed.
                end if;
            when wait_time =>
                if count = COUNT_MAX then
                    count <= 0;
                    if BTN_clk = '0' then
                        if CE = '1' then
                            if UD = '1' then
                                if Bint = "000" then
                                    Bint <= "111";
                                else
                                    Bint <= Bint - 1;
                                end if;
                            elsif UD = '0' then
                                if Bint = "111" then
                                    Bint <= "000";
                                else
                                    Bint <= Bint + 1;
                                end if;
                            end if;
                        end if;
                    end if;
                else
                    count <= count + 1;
                end if;
        end case;
    end if;
end process;
```

Slika 6.3 *architecture* opis sklopa

6.4 Dodjeljivanje priključaka

Sadržaj ucf datoteke i dodjela fizičkih priključaka je prikazana na slici 6.4. CLK je spojen na oscilator koji se nalazi pod priključkom 38.

```
NET "UD" LOC="P124";
NET "not_RES" LOC="P94";
NET "BTN_clk" LOC="P143";

NET "ANODA" LOC="P126";
NET "CE" LOC="P39";

NET "B[0]" LOC="P58"; #D
NET "B[1]" LOC="P61"; #G
NET "B[2]" LOC="P56"; #A

NET "CLK" LOC="P38";
```

Slika 6.4 fizički priključci sklopa za Primjer 3

7. Primjer 4

7.1 Zadatak

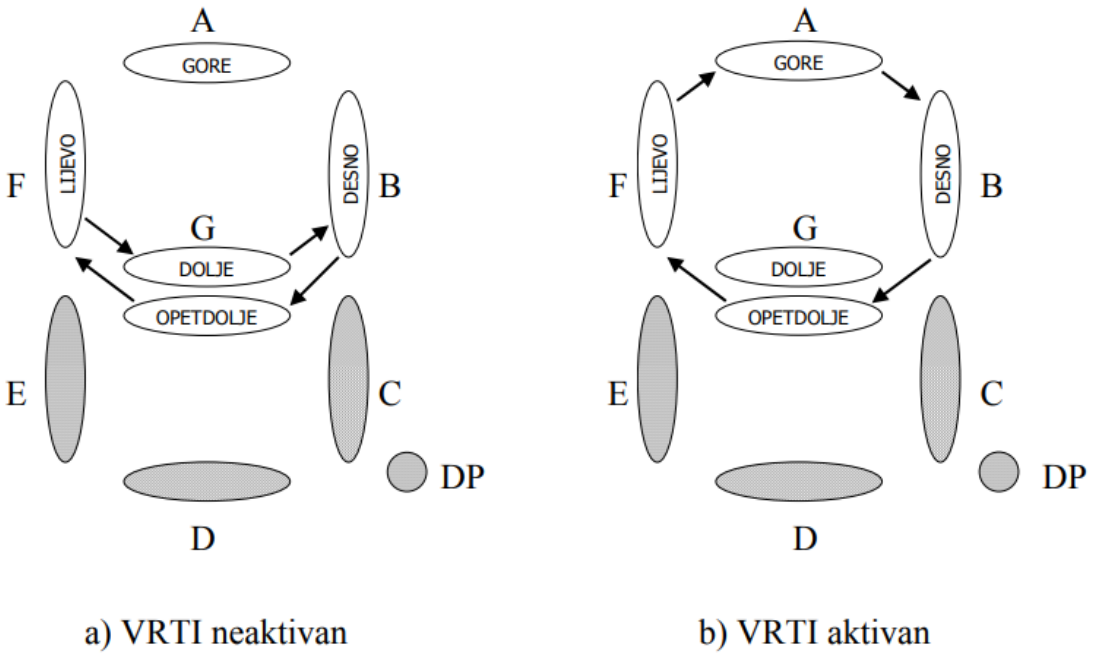
Korištenjem konačnog automata tipa Mealy treba realizirati sljedeću funkciju. Inicijalno stanje kod uspostavljanja napajanja je stanje INIT uz koje ne svijetli niti jedan segment. Nakon inicijalnog stanja treba ući u stanje koje uzrokuje paljenje lijevog segmenta. Nadalje, ovisno o stanju na ulaznoj liniji VRTI treba svijetliti jedan od segmenata A, B, F ili G zaslona. Ako je VRTI u neaktivnom stanju (u logičkoj 0) segmenti trebaju svijetliti sljedećim redoslijedom: F→G→ B→G→F i tako stalno (slika 7.1 a)). Na zaslonu to izgleda kao da se svijetleći segment "ljulja".

Kada je VRTI u aktivnom stanju (logička 1), svijetleći segment treba kružiti u smjeru kazaljke na satu tj. redoslijed je sljedeći: F→A→ B→G→F itd. (slika 7.1 b)).

Ulaz u sklop je i linija /RES (sinkroni reset) s tipke koja u aktivnom stanju (logička nula) uzrokuje da neovisno o tome koji je segment do tada svijetlio, počinje svijetliti segment F i to tako dugo dok je /RES aktivan.

Još jedna ulazna linija INV uvjetuje hoće li aktivni izlazi svijetliti ili će biti ugašeni. Drugim riječima kada je ta linija u 0, svijetlit će samo oni segmenti koji bi po gornjoj funkciji trebali svijetliti, a ostali će biti ugašeni. Ako je na ulazu INV logička jedinica, logički nivoi na izlazima bit će takvi da se onaj segment koji bi po predviđenoj funkciji trebao svijetliti zagasi, a oni koji su neaktivni se sada upale (inverzno stanje svijetli-ne svijetli od originalnog).

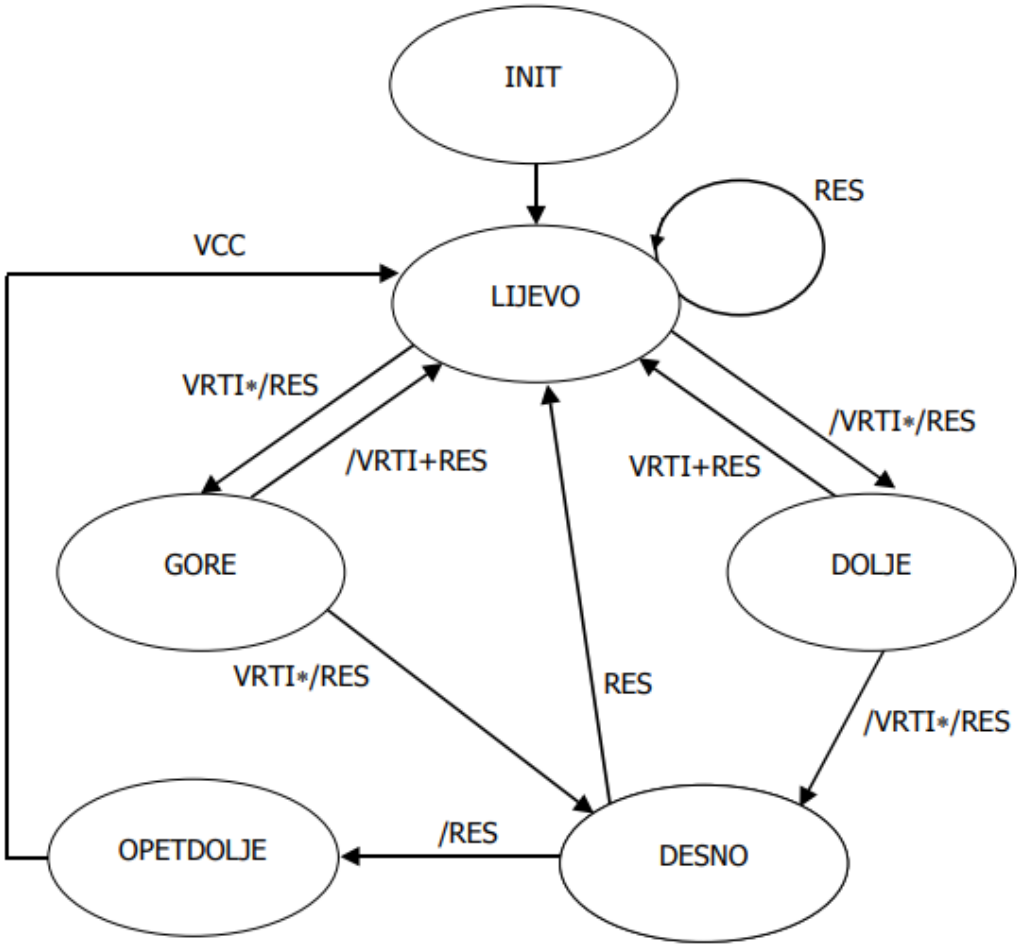
Nacrtati i dijagram prelaska stanja u stanje.



Slika 7.1 Prikaz aktivnosti segmenata zaslona ovisno o liniji VRTI

7.2 Rješenje

Zadatak je potrebno riješiti implementacijom Mealy sinkronog automata. Mealyev automat je vrsta konačnog automata kojem izlaz ovisi o ulazu i trenutnom stanju. Funkciju prelaska konačnog automata iz jednog stanja u drugo za ovaj primjer najlakše je prikazati dijagramom stanja na slici 7.2.



Slika 7.2 Dijagram prelaska stanja

7.3 VHDL kod

Slično kao u Primjeru 3, od ulaza su korišteni CLK za oscilator na pločici, BTN_clk za davanje takta, te ostali ulazi VRTI_in, INVERT i not_RES čija je funkcija navedena u zadatku. Od izlaza su navedeni VRTI koji svijetli kada je VRTI_in aktivan, ANODA i vektor D od četiri člana za segmente 7-segmetnog zaslona.

U *architecture* opisu sklopa navedene su dvije vrste stanja, state_machine i state_type. State_machine sadrži dvije vrste stanja i korišten je kao u prošlom primjeru za *debouncing* sklopa, a state_type sadrži šest stanja sa dijagrama prelaska stanja prikazanog na slici 7.2. Također, trenutno stanje i sljedeće stanje su definirani kao INIT.

```
type state_type is (INIT, LIJEVO, DOLJE, DESNO, OPETDOLJE, GORE);
signal c_state : state_type := INIT;
signal n_state : state_type := INIT;

constant COUNT_MAX : integer := 8000000;
signal count : integer := 0;
type state_machine is (idle,wait_time); --state machine
signal state : state_machine := idle;
```

Slika 7.3 Stanja, signali i konstante

Proces u kojem je opisan *debouncing* sklopa je vrlo sličan onome u Primjeru 3. Razlika je postavljanje stanja LIJEVO kao trenutnog stanja u slučaju resetiranja sklopa, umjesto gašenja segmenta zaslona. Druga razlika je prelazak trenutnog stanja u sljedeće stanje, kada je utvrđeno da je gumb pritisnut, umjesto niza *if-else* naredbi u Primjeru 3.

Drugi proces opisuje funkciju sklopa po dijagramu stanja sa slike 7.2. Na početku su definirane konstante koje simboliziraju segment(e) zaslona koje trebaju svijetliti, radi jednostavnijeg snalaženja u opisu prijelaza između stanja. Opis prijelaza stanja za INIT, LIJEVO i DOLJE nalazi se na slici 7.4. Korištenjem istih naredbi i logike sa dijagrama stanja riješen je i prijelaz stanja za DESNO,

OPETDOLJE i GORE.

```
ANODA <= '0';
VRTI <= not VRTI_in;
case c_state is
     when INIT =>
        D <= START;
        n_state <= LIJEVO;
     when LIJEVO =>
        if INVERT = '1' then
           D <= inv_F;
        else
           D <= F;
        end if;
        if not_RES = '1' then
           if VRTI_in = '1' then
              n_state <= GORE;
           else
              n_state <= DOLJE;
           end if;
        else
           n_state <= LIJEVO;
        end if;
     when DOLJE =>
        if INVERT = '1' then
           D <= inv_G;
        else
           D <= G;
        end if;
        if VRTI_in = '0' and not_RES = '1' then
           n_state <= DESNO;
        else
           n_state <= LIJEVO;
        end if;
```

Slika 7.4 Opis prijelaza stanja za INIT, LIJEVO i DOLJE

7.4 Dodjeljivanje priključaka

Na slici 7.5 je sadržaj ucf datoteke za Primjer 4.

```
NET "D[3]" LOC="P56";
NET "D[2]" LOC="P53";
NET "D[1]" LOC="P54";
NET "D[0]" LOC="P61";
NET "VRTI" LOC="P59";

NET "ANODA" LOC="P126";

NET "VRTI_in" LOC="P124";
NET "not_RES" LOC="P94";
NET "INVERT" LOC="P39";
NET "BTN_clk" LOC ="P143";
NET "CLK" LOC="P38";
```

Slika 7.5 Sadržaj ucf datoteke

8. Zaključak

Pločica Coolrunner-II Starter Board se potvrđuje kao dobro rješenje za implementaciju i izvođenje danih primjera i zadovoljava zahtjeve za modernizacijom laboratorijskih vježbi iz Ugradbenih računalnih sustava. Sadrži potrebna svojstva CPLD-a te dovoljan broj ulaza i izlaza za lakše izvođenje zadataka od trenutno korištenih PAL sklopova. Također, ISE Design Suite alat za sintezu i simuliranje uspješno sintetizira kod i prebacuje opisani dizajn sklopa na pločicu, te se pokazuje kao bolje rješenje od PALASM-a.

LITERATURA

[1] CoolRunner-II CPLD Family Product Specification, Xilinx Inc., 2008, http://www.xilinx.com

[2] CoolRunner-II™ Starter Board Reference Manual, Digilent Inc., 2012, http://www.digilentinc.com

[3] Ugradbeni računalni sustavi – Materijali i laboratorijske vježbe, FER, http://www.fer.unizg.hr/predmet/urs

[4] VHDL coding tips and tricks – Pushbutton DeBounce circuit in VHDL, http://vhdlguru.blogspot.com/

Projektiranje i izvedba digitalnih sustava u CPLD i FPGA tehnologiji

Sažetak

U radu je realizirano projektiranje i izvedba digitalnih sustava u CPLD i FPGA tehnologiji pomoću CoolRunner-II CPLD Starter Board-a koji sadrži CPLD sklop u TQ144 kućištu. Zadani primjeri su rješeni korištenjem alata ISE Design Suite u kojem je za opisivanje rada sklopa korišten VHDL jezik. Ispravnost napisanih rješenja je provjerena pomoću izlaznih svjetlećih segmenata na Coolrunner-II sklopu.

Ključne riječi: CoolRunner-II CPLD Starter Board, CPLD, ISE Design Suite, VHDL

Design and implementation of digital systems using CPLD and FPGA technology

Abstract

This paper describes design and implementation of digital systems in CPLD and FPGA technology using the CoolRunner-II CPLD Starter Board that contains the CPLD chip in the TQ144 package. Given examples are resolved using the ISE Design Suite tool in which VHDL language was used to describe the operation of the circuit. The solution's validity is verified by the output light segments on the Coolrunner-II board.

Keywords: CoolRunner-II CPLD Starter Board, CPLD, ISE Design Suite, VHDL

Dodatak

primjer1.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity primjer1 is port (
     D : in std_logic_vector (3 downto 0);
     C : out std_logic_vector (1 downto 0);
     ANODA : out std_logic
     );
end primjer1;

architecture Behavioral of primjer1 is

begin

    process (D)
        begin
            ANODA <= '0';
            case D is
                 when "1011" => C <= "11";
                 when "0111" => C <= "01";
                 when "0001" => C <= "10";
                 when "0010" => C <= "00";
                 when others => C <= "11";
            end case;
    end process;

end Behavioral;
```

primjer1.ucf

```
NET "D[3]" LOC="P124";
NET "D[2]" LOC="P39";
NET "D[1]" LOC="P94";
NET "D[0]" LOC="P143";

NET "C[0]" LOC="P54";
NET "C[1]" LOC="P53";

NET "ANODA" LOC="P126";
```

primjer2.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity primjer2 is port (
     D : in std_logic_vector (3 downto 0);
     S : out std_logic_vector (7 downto 0);
     ANODA : out std_logic
     );
end primjer2;

architecture Behavioral of primjer2 is
```

```
begin

    process (D)
        begin
            ANODA <= '0';
            case D is
                 when "0011" => S <= "00000011";
                 when "0010" => S <= "10011111";
                 when "0001" => S <= "00100101";
                 when "0000" => S <= "00001101";
                 when "0111" => S <= "10011001";
                 when "0110" => S <= "01001001";
                 when "0101" => S <= "01000001";
                 when "0100" => S <= "00011111";
                 when "1011" => S <= "00000001";
                 when "1010" => S <= "00001001";
                 when "1001" => S <= "00010001";
                 when "1000" => S <= "11000001";
                 when "1111" => S <= "01100011";
                 when "1110" => S <= "10000101";
                 when "1101" => S <= "01100001";
                 when "1100" => S <= "01110001";
                 when others => S <= "XXXXXXXX";
            end case;
    end process;
end Behavioral;
```

primjer2.ucf

```
NET "D[3]" LOC = "P124";
NET "D[2]" LOC = "P39";
NET "D[1]" LOC = "P94";
NET "D[0]" LOC = "P143";

NET "S[7]" LOC = "P56";
NET "S[6]" LOC = "P53";
NET "S[5]" LOC = "P60";
NET "S[4]" LOC = "P58";
NET "S[3]" LOC = "P57";
NET "S[2]" LOC = "P54";
NET "S[1]" LOC = "P61";
NET "S[0]" LOC = "P59";

NET "ANODA" LOC = "P126";
```

primjer3.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;


entity primjer3 is port(
     CLK : in std_logic;
     UD, CE : in std_logic;
     not_RES, BTN_clk : in std_logic;
     ANODA : out std_logic;
     B : out std_logic_vector (2 downto 0) --  B(2) = A , B(1) = G , B(0) = D
     );
end primjer3;

architecture Behavioral of primjer3 is
```

```
constant COUNT_MAX : integer := 8000000;

signal count : integer := 0;
type state_type is (idle,wait_time); -- state machine
signal state : state_type := idle;
signal Bint : unsigned (2 downto 0);

begin

B <= std_logic_vector(Bint);

process (not_RES, CLK, CE, UD)

begin
    ANODA <= '0';
    if not_RES = '0' then
        state <= idle;
        Bint <= "111";
    elsif rising_edge(CLK) then
        case state is
            when idle =>
                if BTN_clk = '0' then
                    state <= wait_time;
                else
                    state <= idle; -- wait until button is pressed.
                end if;
            when wait_time =>
                if count = COUNT_MAX then
                    count <= 0;
                    if BTN_clk = '0' then
                        if CE = '1' then
                           if UD = '1' then
                              if Bint = "000" then
                                 Bint <= "111";
                              else
                                 Bint <= Bint - 1;
                              end if;
                           elsif UD = '0' then
                              if Bint = "111" then
                                 Bint <= "000";
                              else
                                 Bint <= Bint + 1;
                              end if;
                           end if;
                        end if;
                     end if;
                     state <= idle;
                else
                     count <= count + 1;
                end if;
        end case;
    end if;
end process;

end architecture Behavioral;
```

primjer3.ucf

```
NET "UD" LOC="P124";
NET "not_RES" LOC="P94";
NET "BTN_clk" LOC="P143";

NET "ANODA" LOC="P126";
NET "CE" LOC="P39";
```

```
NET "B[0]" LOC="P58"; #D
NET "B[1]" LOC="P61"; #G
NET "B[2]" LOC="P56"; #A

NET "CLK" LOC="P38";
```

primjer4.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;


entity primjer4 is port (
     VRTI_in, INVERT, not_RES, CLK, BTN_clk : in std_logic;
     D : out std_logic_vector (3 downto 0); -- D[3] = A, D[2] = B, D[1] = F,
D[0] = G
     VRTI, ANODA : out std_logic
     );
end primjer4;

architecture Behavioral of primjer4 is

type state_type is (INIT, LIJEVO, DOLJE, DESNO, OPETDOLJE, GORE);
signal c_state : state_type := INIT;
signal n_state : state_type := INIT;

constant COUNT_MAX : integer := 8000000;
signal count : integer := 0;
type state_machine is (idle,wait_time); --state machine
signal state : state_machine := idle;

begin

state_process : process (CLK, not_RES)
begin
        if(not_RES = '0') then
           state <= idle;
           c_state <= LIJEVO;
        elsif(rising_edge(CLK)) then
           case (state) is
               when idle =>
                   if(BTN_clk = '0') then
                      state <= wait_time;
                   else
                      state <= idle; --wait until button is pressed.
                   end if;
               when wait_time =>
                   if(count = COUNT_MAX) then
                      count <= 0;
                      if(BTN_clk = '0') then
                            c_state <= n_state;
                      end if;
                      state <= idle;
                   else
                      count <= count + 1;
                   end if;
           end case;
        end if;
end process;

LED : process (c_state, n_state, VRTI_in, INVERT, not_RES)

CONSTANT START : STD_LOGIC_VECTOR(3 DOWNTO 0) := "1111";
CONSTANT F : STD_LOGIC_VECTOR(3 DOWNTO 0) := "1101";
CONSTANT A : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0111";
```

```
CONSTANT B : STD_LOGIC_VECTOR(3 DOWNTO 0) := "1011";
CONSTANT G : STD_LOGIC_VECTOR(3 DOWNTO 0) := "1110";
CONSTANT inv_F : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0010";
CONSTANT inv_A : STD_LOGIC_VECTOR(3 DOWNTO 0) := "1000";
CONSTANT inv_B : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0100";
CONSTANT inv_G : STD_LOGIC_VECTOR(3 DOWNTO 0) := "0001";

begin

    ANODA <= '0';
    VRTI <= not VRTI_in;
    case c_state is
         when INIT =>
            D <= START;
            n_state <= LIJEVO;
         when LIJEVO =>
            if INVERT = '1' then
               D <= inv_F;
            else
               D <= F;
            end if;
            if not_RES = '1' then
               if VRTI_in = '1' then
                  n_state <= GORE;
               else
                  n_state <= DOLJE;
               end if;
            else
               n_state <= LIJEVO;
            end if;
         when DOLJE =>
            if INVERT = '1' then
               D <= inv_G;
            else
               D <= G;
            end if;
            if VRTI_in = '0' and not_RES = '1' then
               n_state <= DESNO;
            else
               n_state <= LIJEVO;
            end if;
         when DESNO =>
            if INVERT = '1' then
               D <= inv_B;
            else
               D <= B;
            end if;
            if not_RES = '1' then
               n_state <= OPETDOLJE;
            else
               n_state <= LIJEVO;
            end if;
         when OPETDOLJE =>
            if INVERT = '1' then
               D <= inv_G;
            else
               D <= G;
            end if;
            n_state <= LIJEVO;
         when GORE =>
            if INVERT = '1' then
               D <= inv_A;
            else
               D <= A;
            end if;
            if VRTI_in = '1' and not_RES = '1' then
               n_state <= DESNO;
            else
```

```
                n_state <= LIJEVO;
            end if;

    end case;

end process;

end Behavioral;
```

primjer4.ucf

```
NET "D[3]" LOC="P56";
NET "D[2]" LOC="P53";
NET "D[1]" LOC="P54";
NET "D[0]" LOC="P61";
NET "VRTI" LOC="P59";

NET "ANODA" LOC="P126";

NET "VRTI_in" LOC="P124";
NET "not_RES" LOC="P94";
NET "INVERT" LOC="P39";
NET "BTN_clk" LOC ="P143";
NET "CLK" LOC="P38";
```