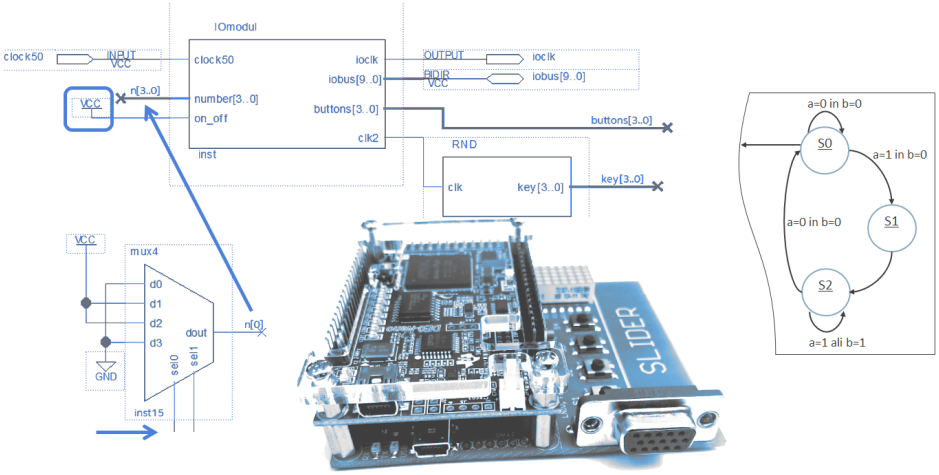
Programirljivi digitalni sistemi: priročnik za vaje

Janez Perš
Andrej Trost

2022



Kataložni zapis o publikaciji (CIP) pripravili v Narodni in univerzitetni knjižnici v Ljubljani

COBISS.SI-ID 120486915

ISBN 978-961-243-438-0 (PDF)

URL: https://lniv.fe.uni-lj.si/pds-vaje.html

Založnik: Založba FE, Ljubljana
Izdajatelj: Fakuleta za elektrotehniko, Ljubljana
Urednik: prof. dr. Sašo Tomažič
1. elektronska izdaja

Priročnik vsebuje navodila za dvanajst laboratorijskih vaj pri predmetu Programirljivi digitalni sistemi na študijski smeri Aplikativna elektrotehnika.

Uvodno poglavje nas seznani s programsko opremo Intel Quartus Lite, v kateri načrtujemo, prevajamo in nalagamo digitalno vezje na razvojno ploščo DE0-Nano. V okviru vaj razvijamo gradnike elektronskega sefa, se spoznamo s shematskim načrtovanjem digitalnih vezij, tiskanih vezij, načrtovanjem v strojno-opisnem jeziku in izdelamo digitalne sisteme s preprosto centralno-procesno enoto.

Vaje so opremljene z razlago, opisom naloge in možnosti izvedbe. Poudarek je na uporabi gradnikov iz osnov digitalnih vezij in brezplačnih razvojnih orodij, zato so vaje uporabne tudi za sorodne študijske predmete ali samostojno delo.

Kazalo

1 Uvod v delo **1**
1.1 Izdelava novega projekta 2
1.2 Kaj imate že na razpolago? 4
1.3 Kaj morate narediti vi? 4

2 Elektronska ključavnica **5**
2.1 Kaj imate že na razpolago? 6
2.2 Kaj morate narediti vi? 6
2.3 Možnosti izvedbe 7

3 Prikazovalnik za ključavnico **9**
3.1 Prikazovalnik 9
3.2 Kaj imate že na razpolago? 10
3.3 Kako naredim dekodirnik z izbiralniki? 11
3.4 Možnosti izvedbe 12

4 Generator ključev **13**
4.1 Sekvenčno vezje 13
4.2 Štiri-bitni psevdonaključni generator 15
4.3 Kaj morate narediti vi? 16
4.4 Možnosti izvedbe 16

5 Načrtovanje tiskanega vezja **17**
5.1 Kratka navodila za risanje sheme 17
5.2 Postopek načrtovanja tiskanega vezja 20
5.3 Kaj morate narediti vi? 22

6 Potujoča luč **23**
6.1 Diagram prehajanja stanj 24
6.2 Shema sekvenčnega vezja 25
6.3 Načrt vezja v strojno-opisnem jeziku 26
6.4 Kaj morate narediti vi? 28

7 Rotacijski kodirnik **29**
7.1 Rotacijski inkrementalni kodirnik 29
7.2 Diagram stanj 30
7.3 Vezje za štetje korakov v SHDL 31
7.4 Kaj morate narediti vi? 32

8 Starinski sef z elektronsko ključavnico **33**
8.1 Nadgradnja vezja za rotacijski kodirnik 33
8.2 Vaša naloga 34

9 Mikroprocesor CPE4 **37**
9.1 Gradnik: procesorski blok 37
9.2 Program v zbirnem jeziku 38
9.3 Shema procesorskega sistema 40
9.4 Kaj morate narediti vi? 40

10 Pulzno-širinski modulator **43**
10.1 Pulzno-širinska modulacija 43
10.2 Vezje pulzno-širinskega modulatorja 44
10.3 Vaša naloga 45

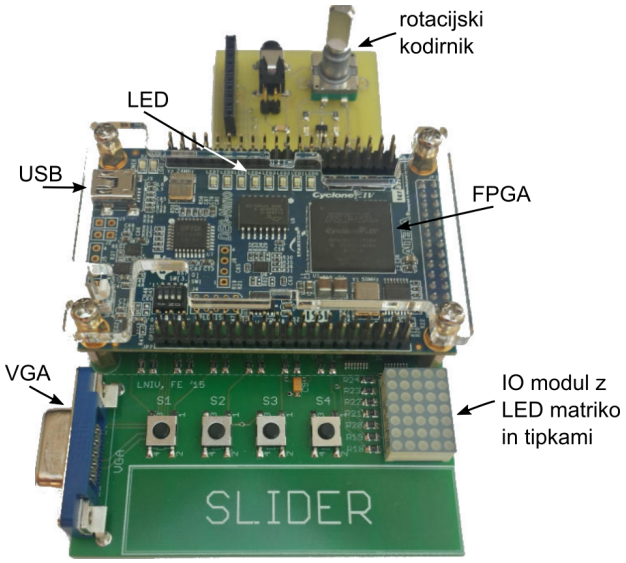
11 Krmiljenje matrike LED **47**
11.1 Uporaba matrike LED 47
11.2 Prikazovanje točke 49
11.3 Kaj morate narediti vi? 49

12 Generiranje signala VGA **51**
12.1 Povzetek naloge 51
12.2 Signal VGA 51
12.3 Kaj morate narediti vi? 53

Vaja 1

Uvod v delo

Na laboratorijskih vajah bomo razvijali digitalna vezja in jih preizkušali na razvojnih ploščah DE0-Nano [1]. Slika 1.1 prikazuje razvojni sistem, ki ga sestavljajo programirljiva razvojna plošča z vezjem FPGA iz družine Cyclone IV E razširitveni IO modul s tipkami, LED matriko in izhodom VGA ter dodatne razširitvene plošče [2, 3].



Slika 1.1: *Razvojni sistem DE0-Nano z razširitvenimi ploščami.*

Razvojno orodje za programirljiva vezja ponuja proizvajalec vezij FPGA - v našem primeru podjetje Intel, ki je pred leti kupilo prvotnega proizvajalca Altera. Orodje **Quartus Prime - Lite** je na voljo na spletni strani: https://www.altera.com/products/design-software/fpga-design/quartus-prime/download.html.

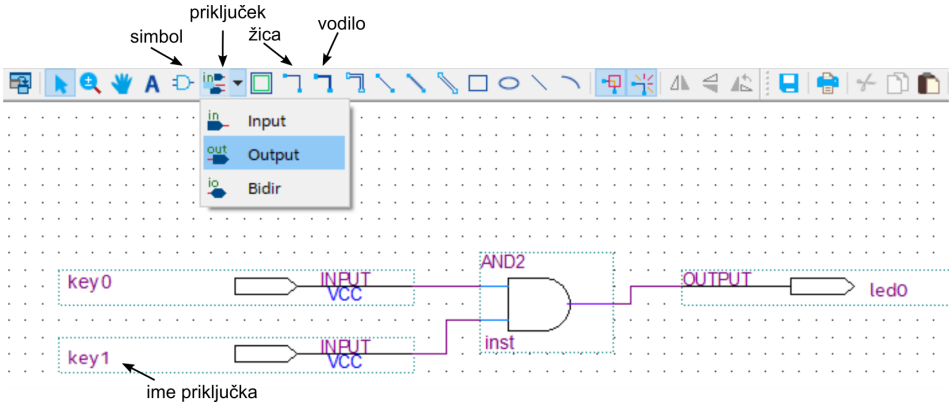
1.1 Izdelava novega projekta

V orodju **Quartus** naredimo nov projekt v nekaj korakih, ki jih ponuja *New Project Wizzard*. Najprej določimo ime projekta (npr. vaja) in delovno mapo, dodamo obstoječe datoteke z opisom vezja, na koncu pa izberemo vrsto vezja FPGA: *Family* Cyclone IV E in *Name* EP4CE22F17C6. Razvojno orodje pripravi v izbrani mapi dve datoteki:

- vaja.qpf - glavna datoteka s katero odpremo projekt
- vaja.qsf - datoteka z nastavitvami: vrsta FPGA, priključki, ipd.

Izdelava sheme vezja

Novo shemo naredimo z ukazom *New, Block Diagram/Schematic File*. Celotno vezje po potrebi razdelimo na več shem. Glavno shemo, v kateri je celotno vezje, poimenujemo enako kot projekt (npr. vaja.bdf).



Slika 1.2: *Orodja za risanje sheme vezja.*

Slika 1.2 prikazuje urejevalnik in orodja oz. ikone za risanje sheme, ki

so sestavljajo logični simboli iz knjižnice, povezave in zunanji priključki. Simbolom zunanjih priključkov določimo ime, da jih bomo lahko povezali s priključnimi nožicami integriranega vezja.

Prevajanje in nalaganje

Najprej izvedemo analizo načrta vezja: *Processing, Start, Analysis, Ctrl+K*. V tem koraku se združijo vse datoteke z opisom vezja v enoten načrt. Quartus pri analizi javlja morebitne napake, npr. nepovezane vhode ali kratek stik izhodov. Če ni napak, izvedemo simulacijo ali pa nadaljujemo s prevajanjem oz. tehnološko preslikavo. V novem projektu moramo pred prevajanjem določiti lokacije zunanjih priključkov z orodjem *Pin Planner*, da bodo signali povezani z ustreznimi perifernimi enotami (tipkami, LED, ...).

Celotno prevajanje v izbrano tehnologijo izvedemo z ukazom: *Processing, Start Compilation, Ctrl+L*. Prevajanje je zahteven proces, ki traja nekaj minut, potrebno pa je pri vsaki spremembi vezja, vključenih datotek ali nastavitev priključkov.

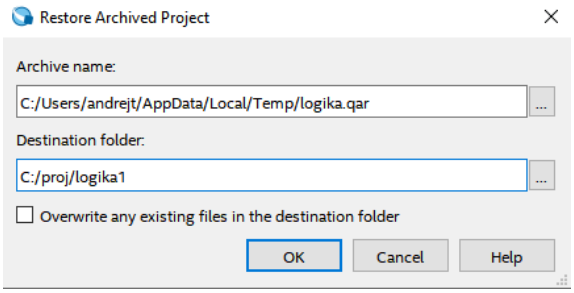
Pred pričetkom nalaganja (programiranja) vezja FPGA povežemo razvojno ploščo z računalnikom. Razvojna plošča ima vgrajen programator in se računalniku predstavi kot naprava *USB-Blaster*. Ko je povezava vzpostavljena izberemo v orodju **Quartus** *Tools, Programmer* in kliknemo gumb Start za prenos podatkov.

Arhiviranje

Orodje **Quartus** izdela med prevajanjem precej pomožnih datotek in map. Projekt za zelo majhna vezja vsebuje skoraj 100 datotek v skupni velikosti nekaj MB, čeprav smo za opis potrebovali le tri izvorne datoteke: glavno (.qpf), nastavitve (.qsf) in shemo (.bdf). Za shranjevanje projektov je na voljo arhiviranje: *Project, Archive Project*, ki v zgoščeno datoteko shrani le izvorne datoteke. Določiti je potrebno le ime arhiva, ki se shrani v datoteko s končnico **.qar** znotraj projektne mape.

1.2 Kaj imate že na razpolago?

Na laboratorijskih vajah bomo uporabljali arhivirane projekte za varnostno kopiranje in kot predloge posamezne vaje. Predloga projekta bo vsebovala delno narisan načrt vezja, specifične komponente (npr. IOmodul) ter vse nastavitve. Ob dvokliku na datoteko s končnico *.qar* odpremo **Quartus**, ki restavrira projekt iz arhiva. Pri tem moramo določiti ciljno mapo (*Destination Folder*) v kateri bo celoten projekt, kot prikazuje slika 1.3.



Slika 1.3: *Restavriranje projekta iz arhiva.*

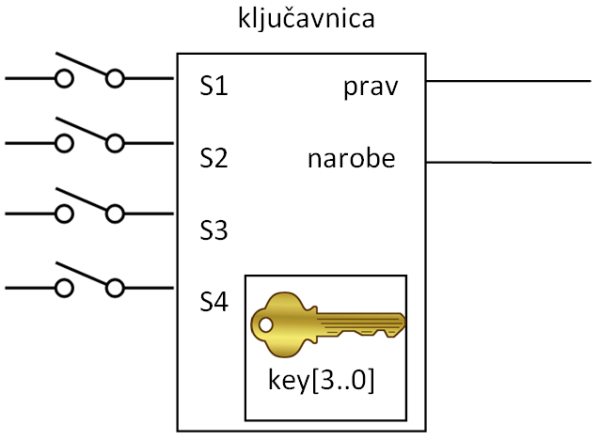
1.3 Kaj morate narediti vi?

Sledite navodilom asistenta na vajah in naredite enostaven projekt v orodju **Quartus**, ga prevedite, preizkusite na razvojni plošči in na koncu naredite arhiv.

Vaja 2

Elektronska ključavnica

Izdelajte digitalno elektronsko ključavnico na razvojnem sistemu s programirljivim vezjem.

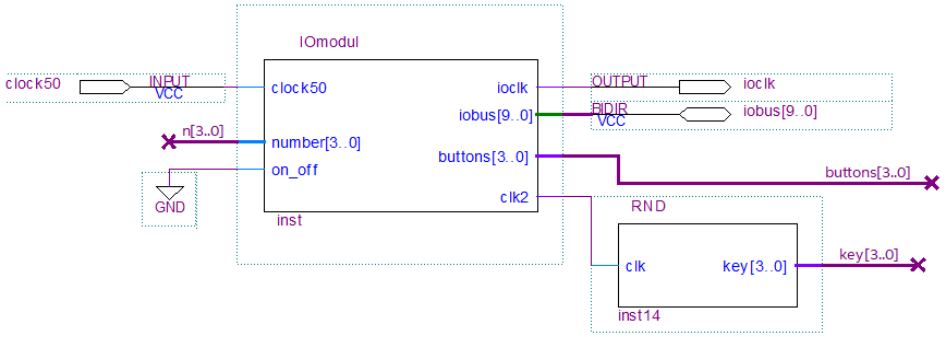


Slika 2.1: *Shema elektronske ključavnice.*

Naredite kombinacijsko vezje, ki bo v elektronski ključavnici preverjalo, ali je uporabnik pritisnil pravo kombinacijo tipk, ki se ujema s *ključem* ključavnice. *Ključ* vaše elektronske ključavnice je štiribitna beseda shranjena v vezju (na primer "0100"), ključavnico pa uporabnik odklene tako, da pritisne kombinacijo tipk **S1-S4**, ki se ujema s ključem.

V primeru ključa **key[3..0]**=”0100” mora uporabnik za odklepanje pritisniti samo tipko **S2**. Da bo naloga bolj zanimiva, je ključavnica narejena tako, da vsake 4 sekunde zamenja ključ po zaporedju, ki ga ne poznate.

2.1 Kaj imate že na razpolago?



Slika 2.2: *Začetna shema z vmesnikom in generatorjem ključev.*

V predlogi projekta iz e-učilnice je glavna shema z dvema komponentama: vhodno-izhodnim vmesnikom *IOmodul* in generatorjem naključnih števil *RND*. Ključ je v shemi viden kot štiribitni signal z oznako **key[3..0]**, mi pa smo že poskrbeli, da se vrednost ključa zamenja vsake štiri sekunde. Zaporedje ključev je takšno, da se ključ 0000 ne more nikoli pojaviti. Kot v prejšnji vaji, imate na razpolago signal iz štirih tipk, z oznako **buttons[3..0]**. Na svetlečih diodah **led4-led7** se prikazuje vrednost trenutnega ključa, da lahko preverite pravilnost delovanja vezja. Pravi uporabnik elektronske ključavnice seveda ključa nikoli ne bi videl!

2.2 Kaj morate narediti vi?

Z logičnimi vrati naredite vezje, ki bo preverilo, ali se kombinacija pritisnjenih tipk ujema s ključem in prikazalo rezultat na svetlečih diodah.

Če se kombinacija tipk ujema s ključem, naj se signal z oznako **prav** postavi na logično 1, sicer pa naj se na 1 postavi signal **narobe**. Zapišite logično funkcijo obeh signalov v odvisnosti od stanja tipk **S1**, **S2**, **S3** in **S4** ter bitov ključa **K0**, **K1**, **K2**, **K3**. Uporabite skrajšan zapis: **key[0]**=**K0**, **key[1]**=**K1**, **buttons[0]**=**S1**, itd.

prav =

---

narobe =

Narišite shemo vezja z izbranimi logičnimi vrati. Signal **prav** naj bo vezan na **led0** in **led1**, signal **narobe** pa na **led2** in **led3**.

2.3 Možnosti izvedbe

Da bo naloga bolj zanimiva upoštevajte dodatne pogoje pri načrtovanju logične funkcije in vezja.

a) Uporabite samo operacije ekskluzivni ali (XOR), logični in (AND) in negacijo (NOT).

b) Uporabite samo ekskluzivni ali (XOR), logični ali (OR) in negacijo (NOT).

c) Uporabite samo logični ali (OR), logični in (AND) in negacijo (NOT).

Vezje prevedite, ga naložite na razvojni sistem in preizkusite delovanje. Pazite na to, da bo vaše vezje ustrezalo logični funkciji, ki ste jo zapisali zgoraj – samo v tem primeru bo naloga pravilno rešena!

Razmisli

- Koliko je vseh kombinacij, ki opisujejo delovanje vezja za preverjanje ključa?

- Na kakšen način lahko pokažemo, da bo vezje pravilno delovalo pri vseh kombinacijah?

- Koliko in katere gradnike zasede vezje znotraj FPGA?

Vaja 3

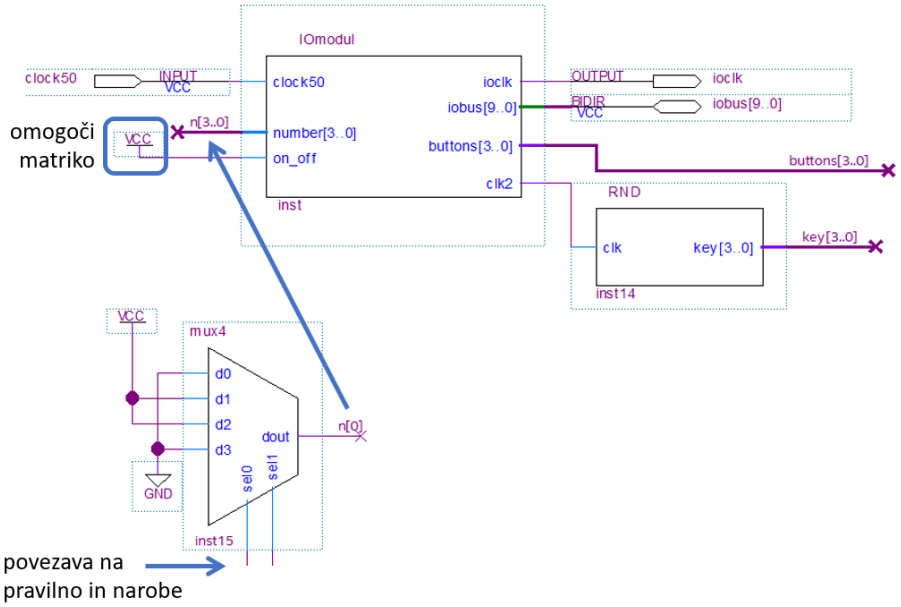
Prikazovalnik za ključavnico

Elektronsko ključavnico iz prejšnje vaje nadgradite s prikazovalnikom, ki bo bolj nazorno pokazal kdaj je nastavljna kombinacija pravilna oz. napačna. Vaša naloga je izdelava dekodirnika za prikazovalnik.

3.1 Prikazovalnik

Na razvojni ploščici je matrični prikazovalnik, ki v povezavi z vhodnim blokom prikazuje številke od 0 do 15 v šestnajstiški obliki (torej 0 do F). Prikazovalnik uporabite tako, da na vhod **number[3..0]** pripeljete štiribitno besedo, katere vrednost se prikaže na matriki. Pozor: do sedaj je bil prikazovalnik izklopljen, saj je bil vhod **on_off** vezan na logično 0 (**GND**). Vklopite ga, ko povežete **on_off** na **Vcc**, kot prikazuje slika 3.1.

Matrični prikazovalnik najprej preizkusite tako, da ga omogočite in naredite povezavo vhoda **number[3..0]** s signalom **buttons[3.0]**, ki predstavlja stanje tipk. Prevedite in naložite vezje ter preizkusite delovanje z nastavljanjem kombinacij na tipkah.



Slika 3.1: *Delna shema rešitve naloge. Prikazana je povezava enega izbiralnika in vmesnika.*

3.2 Kaj imate že na razpolago?

Arhivirajte projekt iz prejšnje vaje in ga razpakirajte v mapo z novim imenom. Narejeno imate že logiko signalov **prav** in **narobe**, ki jih boste povezali na izbirne vhode **sel0** in **sel1** štirih *izbiralnikov* (multiplekserjev, angl. mux). Vsak izbiralnik bo na podlagi **sel0**, **sel1** in vaše vezave podatkovnih vhodov **d0**, **d1**, **d2** in **d3** na maso (**GND**) ali na napajanje (**Vcc**), generiral en bit štiribitne besede. Izhode izbiralnikov povežite na vodilo **n[3..0]** priključeno na vhod vmesnika **number[3..0]**.

Shematski urejevalnik orodja **Quartus** vsebuje zahtevnejše, parametrične izbiralnike (LPM). Pri vaji boste raje uporabljali pripravljen shematski gradnik **mux4**, ki bo na voljo v predlogi projekta, Če gradnika ni, prenesite in dodajte dve datoteki z imenom mux4 v mapo projekta.

3.3 Kako naredim dekodirnik z izbiralniki?

Želimo, da prikazovalnik izpiše naslednje vrednosti: 0 ko sta oba signala, **prav** in **narobe** na nizkem nivoju, ZNAK1, ko je pritisnjena pravilna kombinacija tipk, in ZNAK2, če uporabnik pritisne napačno kombinacijo tipk (ZNAK1 in ZNAK2 sta določena na koncu te naloge). V tabeli je navedena tudi kombinacija, ko sta oba signala **prav** in **narobe** na 1. Glede na prejšnjo nalogo se takšna kombinacija ne more pojaviti, vendar jo navedemo v tabeli, da imamo popolnoma določene povezave v vezju.

Sedaj morate ugotoviti, katere podatkovne vhode (**d0**, **d1**, **d2**, **d3**) izbiralnikov morate povezati na **GND** in katere na **Vcc**, pri tem pa si boste pomagali s tabelo, ki smo jo za vas že delno izpolnili.

Prikaz	**sel1** (**narobe**)	**sel0** (**prav**)	Mux **n[3]**	Mux **n[2]**	Mux **n[1]**	Mux **n[0]**
0 ZNAK1 ZNAK2 -	0	0				

Tabela 3.1: *Tabela, s pomočjo katere bomo pravilno povezali vhode izbiralnikov. Preostanek tabele izpolnite sami.*

Vajo rešite tako, da sledite korakom izvedbe naloge:

- Izpolnite tabelo z upoštevanjem dogovorjenih kod za ZNAK1 in ZNAK2.
- V vezje vstavite izbiralnike in jih povežite glede na tabelo. Ne pozabite vklopiti prikazovalnika, kot je to opisano na začetku naloge.
- Vezje prevedite, naložite na razvojni sistem, preizkusite in demonstrirajte delovanje.

3.4 Možnosti izvedbe

Tri različice naloge v tej nalogi predstavljajo različne vrednosti za ZNAK1 in ZNAK2.

a) ZNAK1 naj predstavlja prikaz vrednosti "O"na prikazovalniku (predstavljajte si, da to pomeni "O"kot "OK", ZNAK2 pa naj je "F"(kot "false")

b) ZNAK1 naj predstavlja prikaz vrednosti "A"na prikazovalniku, ZNAK2 pa prikaz vrednosti "B".

c) ZNAK1 naj predstavlja prikaz vrednosti "1"na prikazovalniku, ZNAK2 pa prikaz vrednosti "2".

Optimizacija

Razmisli, kakšno je najmanjše število izbiralnikov za rešitev naloge in poskusi narediti manjše vezje.

- Če želimo nek izhod povezati na več različnih bitov vodila, je potrebno v shemi uporabiti simbol WIRE za vsak del povezave, ki določimo drugačno ime.

Ugotovi, kako bi nalogo rešil le z osnovnimi logičnimi vrati.

Vaja 4

Generator ključev

Naredili ste že preprosto elektronsko ključavnico, pri kateri je moral uporabnik uganiti skriti štiribitni ključ s pritiskom na ustrezno kombinacijo vhodnih tipk. Ključ se je zamenjal vsakih nekaj sekund. Da je bila rešitev naloge lažja, smo vam takrat že pripravili generator ključev.

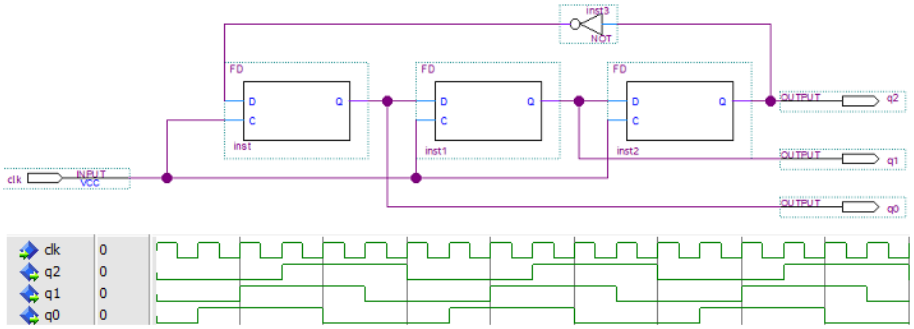
Pri tej vaji boste naredili sekvenčno vezje, ki deluje kot generator naključnih števil in ga vključili v shemo iz prejšnje vaje. Generator ključev naj nadomesti blok RND, ki določa izhodno kombinacijo na vodilu **key[3..0]**.

4.1 Sekvenčno vezje

Sekvenčno vezje sestavljajo pomnilni elementi flip-flopi, ki za en cikel ure shranijo stanje na izhodu. Generator ključev spada med sekvenčna vezja s povratno zanko, kjer je izhod vezja preko logike povezan na vhod.

Pri generatorju naključnih vrednosti se izhodi ne spreminjajo v pravilnem zaporedju. Najbolj preprosti so digitalni generatorji naključnih vrednosti s pomikalnim registrom. Sestavljeni so iz zaporedno povezanih flip flopov in logike, ki določa vhod prvega flip-flopa. Takšni generatorji niso popolnoma naključni, ker se vrednosti na izhodih flip-flopov po določenem številu ciklov ponavljajo, zato jim pravimo generatorji psevdonaključnih vrednosti.

Slika 4.1 prikazuje vezje iz treh D flip-flopov in negatorja v povratni vezavi. Flip-flopi na shemi imajo skupno uro (signal clk) in delujejo tako, da spreminjajo vrednost na izhodu le ob prehodu ure iz 0 na 1. Izhod posameznega flip-flopa je zakasnjen glede na podatkovni vhod D za en cikel ure.



Slika 4.1: *Shema in simulacija 3-bitnega pomikalnega registra s povratno zanko.*

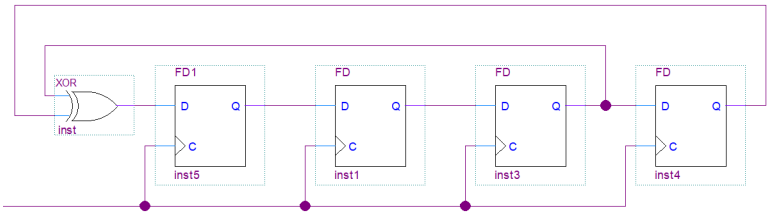
Vsi izhodi flip-flopov naj bodo na začetku simulacije postavljeni na 0. Zaradi negatorja, je vhod prvega flip-flopa na 1, zato se ob naslednjem ciklu postavi **q0** na 1. Naslednji cikel ure se ta vrednost prenese na izhod flip-flopa **q1**, nato pa še na **q2**. Zaradi negatorja, bo sedaj na vhodu prvega flip-flopa logična 0, ki se bo pomikala, kot prikazuje simulacija. Spreminjanje stanj flip-flopov predstavimo s tabelo:

cikel	q0	q1	q2
0	0	0	0
1	1	0	0
2	1	1	0
3	1	1	1
4	0	1	1
5	0	0	1
6	0	0	0

Stanja se po šestih ciklih začnejo ponavljati. Katere kombinacije manjkajo? Če bi na naredili podobno vezje iz štirih flip-flopov, bi videli, da se ponavlja 8 od 16 mogočih kombinacij. Za vezje, ki generira daljše in bolj naključno zaporedje potrebujemo namesto negatorja drugačno logično funkcijo v povratni vezavi.

4.2 Štiri-bitni psevdonaključni generator

Shema 4-bitnega generatorja z maksimalnim zaporedjem je na sliki 4.2. Vezje je sestavljeno iz štirih D flip-flopov in logičnih vrat XOR. Vsi flip-flopi so vezani na isto uro, njihovi izhodi pa predstavljajo stanje vezja.



Slika 4.2: *Pomikalni register s povratno zanko za generiranje naključnih vrednosti.*

Prvi flip-flop na shemi ima nekoliko drugačno oznako (FD1), ker je njegovo začetno stanje 1, ostali flip-flopi (FD) pa imajo začetno stanje 0. Preden začnete z reševanjem naloge naredite analizo vezja na papirju. Začetno stanje vezja je kombinacija 1000 (skupaj napisani izhodi vseh flip-flopov). Ob naslednjem ciklu ure se vrednost iz prvega prenese v drugega, iz drugega v tretjega, ... prvi pa dobi vrednost, ki jo določa funkcija XOR. Napiši stanje vezja ob zaporednih ciklih ure:

cikel	stanje	cikel	stanje
0	1 0 0 0	10	
1		11	
2		12	
3		13	
4		14	
5		15	
6		16	
7		17	
8		18	
9		19	

4.3 Kaj morate narediti vi?

- Naredite analizo generatorja naključnih vrednosti na papirju.
- V projektu iz prejšnje vaje izbrišite gradnik RND z glavne sheme.
- Narišite shemo izbranega vezja z gradniki FD oz. FD1. Dodajte vhodne in izhodne priključke in v orodju **Quartus** generirajte HDL opis (*File, Create, Create HDL Design*), ki ga potrebujete za simulacijo. Po navodilih izvedite simulacijo vezja.
- Novo shemo pretvorite v simbol (*File, Create, Create Symbol Files*), dokončajte glavno shemo in preizkusite delovanje na razvojnem sistemu.

Ugotovi po kakšnem številu ciklov se začnejo kombinacije ponavljati.
Zakaj potrebujemo flip-flop, ki ima začetno stanje 1 ?

4.4 Možnosti izvedbe

a) Naredite naključni generator s funkcijo XOR, kot prikazuje slika 4.2.

b) Naredite naključni generator s štirimi flip-flopi FD in negatorjem v povratni zanki.

c) Naredite naključni generator s štirimi flip-flopi FD in funkcijo XNOR v povratni zanki.

Razmisli

- Kaj se zgodi, če se vezje zaradi kakšne motnje znajde v stanju, ki ni med kombinacijami v tabeli?
- Kako bi naredil generator z daljšim psevdonaključnim zaporedjem, ki imel še vedno le 4-bite na izhodu?

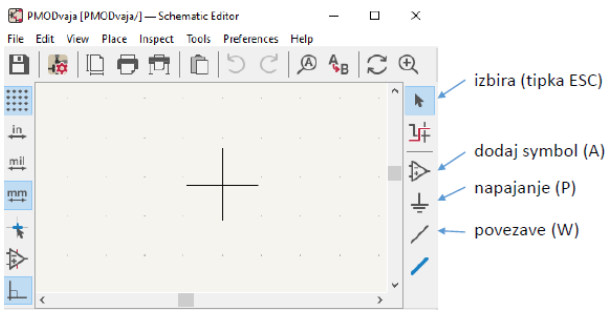
Vaja 5

Načrtovanje tiskanega vezja

Predstavili bomo najosnovnejše korake načrtovanja tiskanega vezja v orodju Kicad na primeru majhnega digitalnega vezja. Kicad je brezplačen program za risanje elektronske sheme in načrtov tiskanega vezja. Dostopen je na: https://www.kicad.org/

5.1 Kratka navodila za risanje sheme

Prvi korak je izdelava novega projekta v katerem bo shema (*sch*) in datoteka z načrtom tiskanega vezja (*pcb*). Dvoklik na datoteko s končnico *sch* odpre shematski urejevalnik:



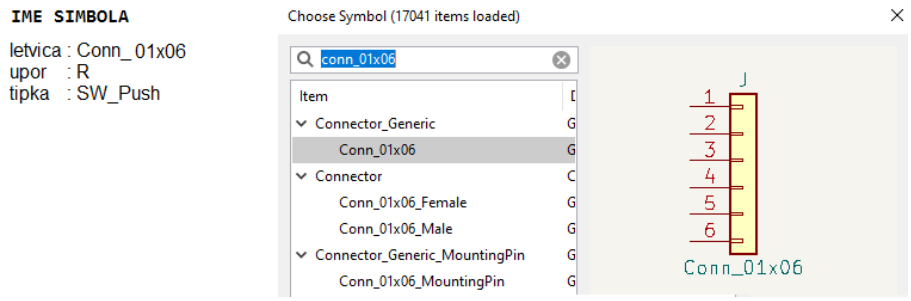
Slika 5.1: *Shematski urejevalnik z ikonami pogosto uporabljenih funkcij.*

Narisali bomo vezje z lestvičnim konektorjem in nekaj pasivnimi elementi.

Korake načrtovanja električne sheme lahko strnemo v 5 točk:

1. Dodaj na list simbole elementov (tipka A) in napajanje (P). Simbol zavrtimo s tipko R, zrcalimo z X ali Y in premikamo z M oz. G (premika tudi povezave).
2. Določi imena in vrednosti posameznih elementov (U, V) ali izberi avtomatsko označevanje (*Annotate Schematic*).
3. Nariši povezave (W), spoje (J) in oznake (L).
4. Preveri shemo (*Electrical Rules Check*) in popravi napake.
5. Določi ohišja elementov (*Assign PCB Footprints*).

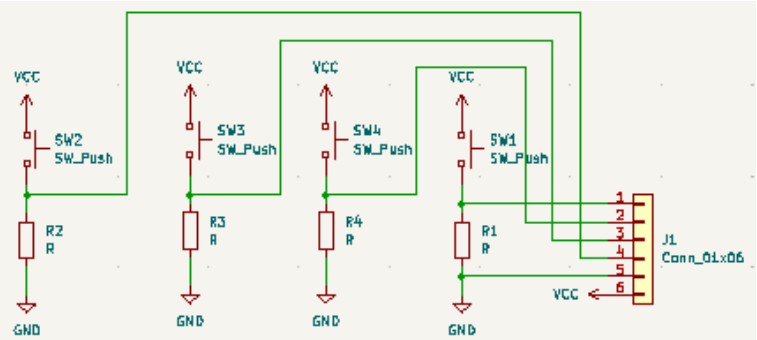
Simbole elementov poiščemo v seznamu knjižnic ali po imenu. Integrirana vezja imajo unikatno oznako, pri pasivnih elementih pa je dobro voditi lasten sezam. Slika 5.2 prikazuje rezultat iskanja enovrstičnega konektorja (letvice) s šestimi priključki, kjer smo izbrali splošen simbol.



Slika 5.2: *Iskanje shematskih elementov po imenih.*

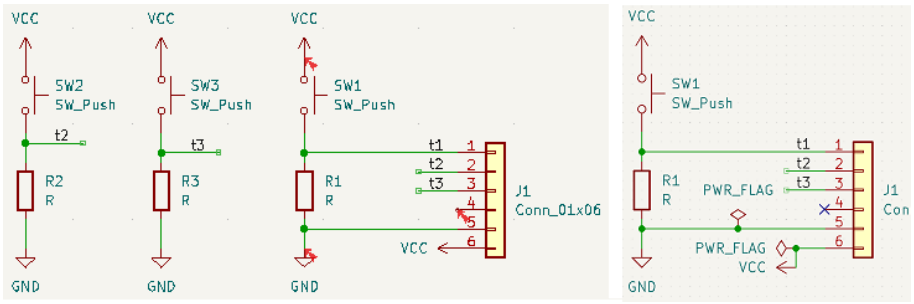
Na shemo dodajmo tipke **SW_Push**, ki so vezane na konektor. Tipka je na eni strani vezana na pozitivni pol napajanja **VCC** (tipka P), na drugi pa čez upor **R** na maso **GND**. Za hitrejše delo najprej postavimo in povežemo eno tipko z uporom, ju obkrožimo z izbiro in uporabimo copy-paste za ostale.

Ko dodamo element na shemo, se pojavi pri oznaki vprašaj, ki ga nadomestimo s številko (npr. R1). Poimenovanje izvedemo ročno (tipka U) ali pa avtomatsko s klikom na *Annotate Schematics*.



Slika 5.3: *Shema vezja s štirimi tipkami.*

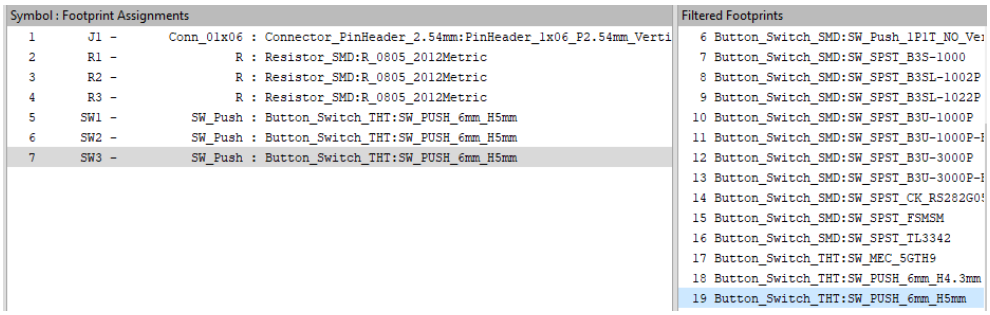
Na sliki 5.3 je primer sheme s štirimi tipkami, kjer smo morali povezave prekrižati. Bolj pregledno shemo naredimo s poimenovanimi povezavami. Na izhodih tipk narišemo le kratko povezavo na katero prilepimo oznako (tipka L) in enako naredimo na ustreznem priključku konektorja.



Slika 5.4: *a) Shema s poimenovanimi povezavami, b) dodane oznake napajanja*

Pri preverjanju sheme (*Electrical rules check*) s tremi tipkami bo Kicad z rdečimi puščicami označil tri potencialne napake: nedefiniran izvor napajanja in nepovezan priključek na konektorju (slika 5.4a). Napajalne priključke konektorja označimo z zastavico **PWR_FLAG**, ki jo najdemo med simboli. Praznemu priključku dodamo simbol X, ki je na voljo med ikonami (slika 5.4b).

Nazadnje določimo ohišja vsem shematskim elementom (*Assign Footprints*). Nekateri simboli že imajo pripeto ohišje, večini splošnih simbolov pa moramo sami pripeti ustrezno ohišje.

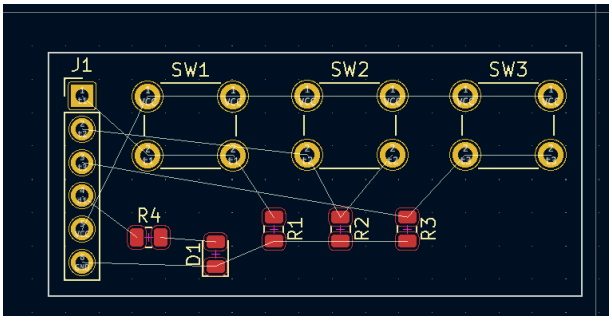


Slika 5.5: *Določitev ohišij elementov.*

V tabeli označimo posamezen simbol, poiščemo ohišje (uporabimo iskalnik ali pregled knjižnic) in z dvoklikom pripnemo izbrano ohišje. Ikone ponujajo predogled ohišij in različne filtre za pomoč pri iskanju. Ko smo določili vse elemente, prenesemo informacije iz sheme v orodje za urejanje tiskanega vezja (F8).

5.2 Postopek načrtovanja tiskanega vezja

V urejevalniku vezja (*PCB Editor*) najprej na smiseln način razmestimo elemente (tipka M). Velikost ploščice omejimo s pravokotnikom na plasti za obrez. Najprej kliknemo ikono pravokotnika, nato pa na zgornjem traku izberemo plast **Edge.Cuts**.

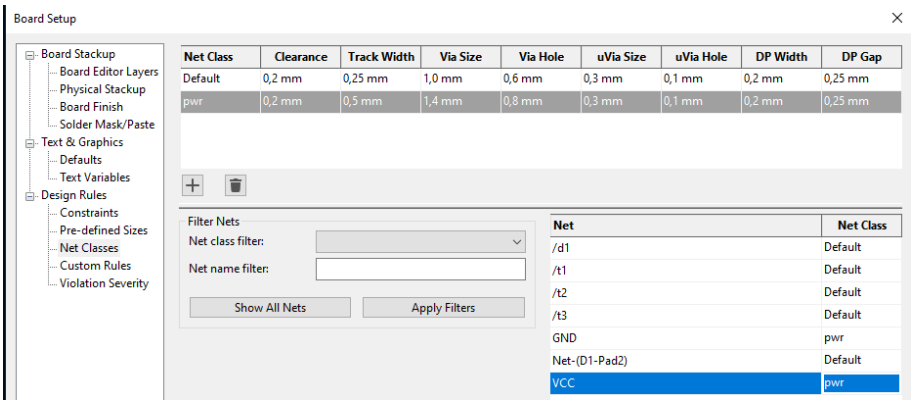


Slika 5.6: *Določitev ohišij elementov.*

V oknu apperance določimo prikazane plasti: zgornjo in spodnjo bakreno plast (F.Cu, B.Cu), tisk zgoraj (F.Silkscreen) in obrez. Pri razmestitvi

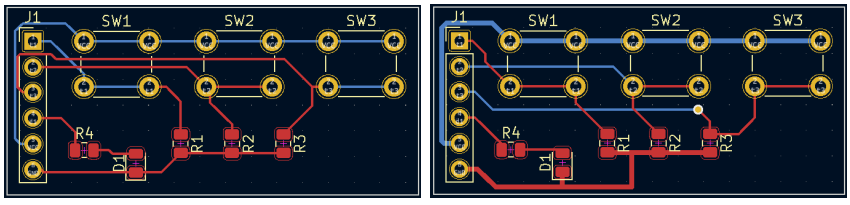
elementov naj se povezave čim manj križajo, konektorje pa postavimo ob rob. Pomagamo si z nastavitvijo mreže *Grid*: 0,254 mm (0,01 in).

Pred risanjem bakrenih povezav nastavimo njihove parametre: menu *File, Board Setup, Net Classes*. Napajalne povezave naj bodo širše, zato dodamo z gumbom + nov razred povezav, ki ga priredimo povezavam (*Net*) z oznako VCC in GND, kot prikazuje slika 5.7.



Slika 5.7: *Nastavitev lastnosti povezav pri risanju tiskanega vezja.*

Povezave rišemo z orodjem Route Signal Track. Ko izberemo ikono za povezovanje izberemo bakreno plast, npr. **F.Cu**, kliknemo na priključek elementa nato pa na vmesne in končne točke. Če je potrebno zamenjati plast, naredimo skoznik (*Via*, tipka V).



Slika 5.8: *Primer dveh izvedb povezav na plošči tiskanega vezja.*

Načrtovanje tiskanega vezja zahteva natančnost in izkušnje. Kadar imamo veliko povezav, je smiselno večino horizontalnih delati z eno, vertikalnih pa z drugo bakreno plastjo.

5.3 Kaj morate narediti vi?

Naredite nov projekt v programu **Kicad** in naredite načrt tiskanega vezja. Najprej narišite shemo digitalnega vezja z enim konektorjem in enostavnimi vhodnimi in izhodnimi enotami:

a) Vezje naj vsebuje 3 tipke in eno izhodno LED v pozitivni logiki. Uporabite simbol: LED, ohišje: LED_0805_2012Metric.

b) Vezje naj vsebuje 3 tipke v negativni logiki in eno izhodno LED v pozitivni logiki.

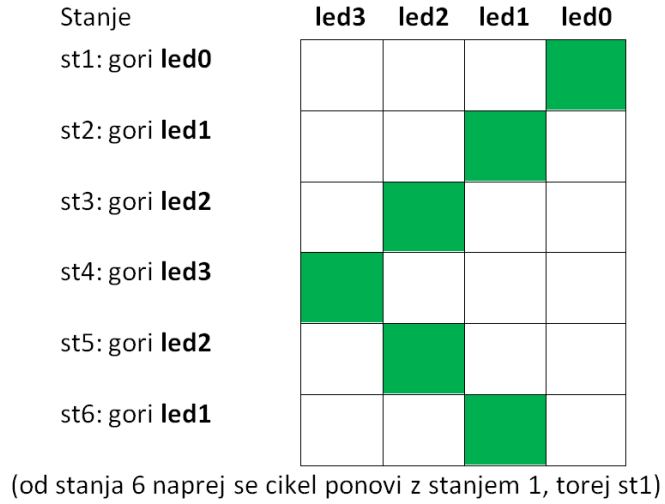
c) Vezje naj vsebuje tipko, rotacijski kodirnik (simbol: RotaryEncoder, ohišje:RotaryEncoder_Alps_EC11E-Switch_Vertical_H20mm) in eno izhodno LED v pozitivni logiki.

Ko boste dokončali shemo, se lotite še razmestitve ohišij elementov in oblikovanja tiskanega vezja.

Vaja 6

Potujoča luč

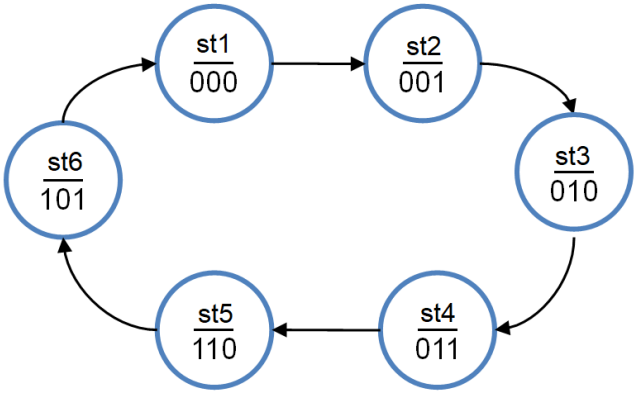
Naredili bomo vezje za učinek potujoče luči, ki se preko štirih LED izmenično pomika levo in desno. Sestavljeno bo iz sekvenčnega stroja in dekodirnika za krmiljenje štirih LED: **led0**, **led1**, **led2** in **led3**.



Slika 6.1: *Prikaz delovanja potujoče luči – shema prižiganja LED glede na stanje sekvenčnega vezja.*

6.1 Diagram prehajanja stanj

Delovanje sekvenčnega stroja predstavimo z diagramom prehajanja stanj. Vsakemu od stanj pripišemo unikaten bitni vzorec, kot prikazuje slika 6.2. Bitni vzorec smo izbrali tako, da bomo lahko spodnja dva bita neposredno povezali na dekodirnik za LED. Če ponovno preučimo prikaz delovanja, ugotovimo, da morata imeti stanji **st2** in **st6** enako kombinacijo spodnjih bitov. Podobno velja za stanji **st3** in **st5**.



	prejšnje stanje			novo stanje		
	s2	s1	s0	s2	s1	s0
	0	0	0			
	0	0	1			
	0	1	0			
	0	1	1			
X	1	0	0			
	1	0	1			
	1	1	0	**1**	**0**	**1**
X	1	1	1	**0**	**0**	**0**

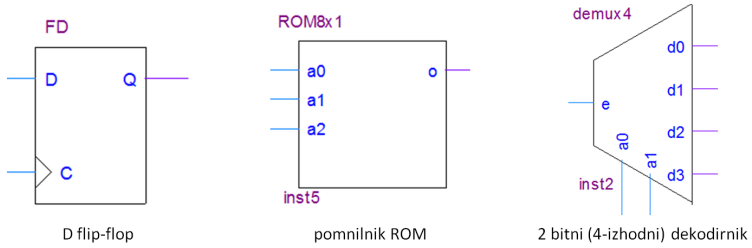
X označuje neuporabljena stanja!

Slika 6.2: *Diagram in tabela prehajanja stanj za potujočo luč. Preostanek tabele izpolnite sami.*

Tabela prehajanja stanj določa kako si stanja sledijo. Posamezne bite označimo s **s0**, **s1** in **s2**. (primer: za stanje **st5** velja **s0** = 0, **s1** = 1, **s2** = 1). Manjkajoče kombinacije iz diagrama naj gredo v stanje 000.

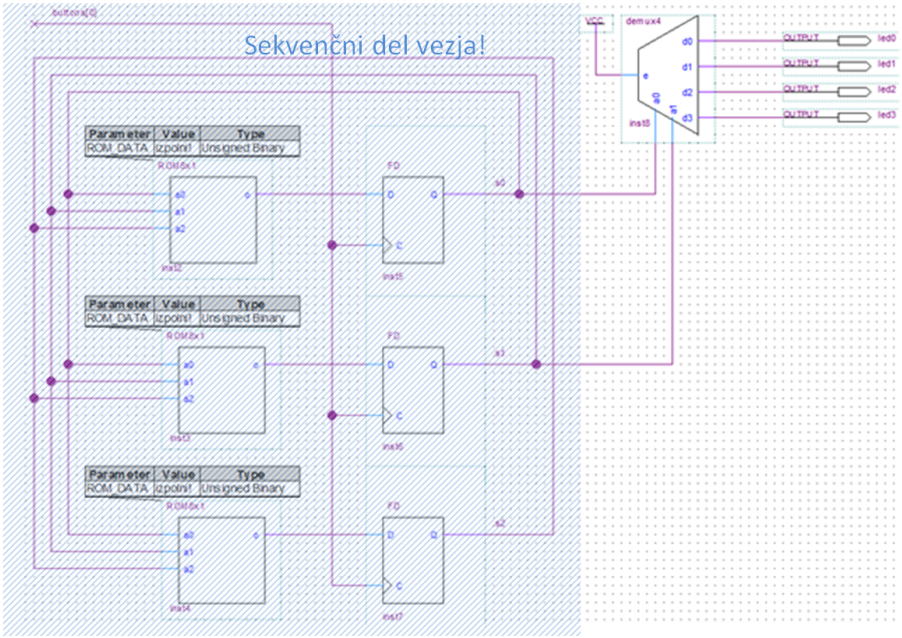
6.2 Shema sekvenčnega vezja

Stanje sekvenčnega vezja določajo flip-flopi FD. En flip-flop ima lahko dve stanji (0 in 1), dva štiri stanja (00, 01, 10, 11), trije pa osem stanj.



Slika 6.3: *Gradniki vezja za učinek potujoče luči.*

Vezje sestavlja sekvenčni stroj iz treh flip-flopov in pomnilnikov ROM v povratni vezavi ter dekodirnik, kot prikazuje slika 6.4.



Slika 6.4: *Shema vezja. Prehajanje stanj določa vsebina pomnilnikov ROM.*

Flip-flopi ob vsakem ciklu ure **C** prenesejo vrednost iz vhoda **D** na izhod. Kombinacija treh izhodov **s0**, **s1** in **s2** predstavlja stanje vezja, ki se preko

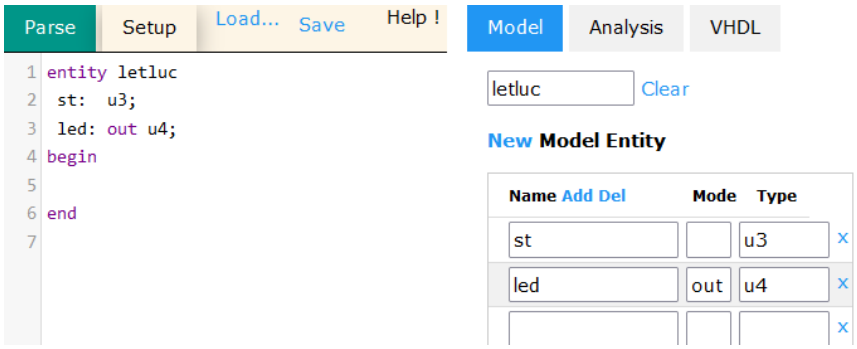
vhodne logike preslika v novo stanje. Preslikavo izvedemo s pomnilniki ROM v povratni vezavi. Vhode v ROM povežemo biti stanja po vrsti, tako da je signal **s0** priklopljen na **a0**, **s1** na **a1**, itd. Vsebina pomnilnikov določa naslednje stanje vezja. Pomnilnik, ki je vezan na FD z izhodom **s0** vsebuje kombinacije, ki se nahajajo v tabeli prehajanja stanj pod **s0**, podobno pa velja za ostala dva.

Dekodirnik na koncu vezja preslika spodnja dva bita stanj **s0** in **s1** v štiri pozicije luči. Kodiranje stanj smo izbrali tako, da se ta dva bita spreminjata v zaporedju: 00, 01, 10, 11, 10, 01, 00, itd., ki generira želeni učinek.

6.3 Načrt vezja v strojno-opisnem jeziku

Sekvenčno vezje potujoče luči lahko učinkovito opišemo v strojno-opisnem jeziku. Uporabite poenostavljen jezik SHDL in spletno orodje za opis vezja na naslovu: https://lniv.fe.uni-lj.si/shdl/ [4].

V spletnem orodju najprej določite ime vezja, v tabeli določite dva signala: 3-bitni signal **st** in 4-bitni izhod **led**, nato pa kliknite **new**, da bo program naredil vzorec kode (slika 6.5).



Slika 6.5: *Nastavitev priključkov v SHDL.*

Delovanje vezja opišemo med **begin** in **end**. Vezje je sekvenčni stroj, ki spreminja stanja ob uri. Za vsako stanje s pogojnim stavkom določimo naslednje stanje glede na diagram prehajanja stanj. Uporabimo stavke **if...elsif**.

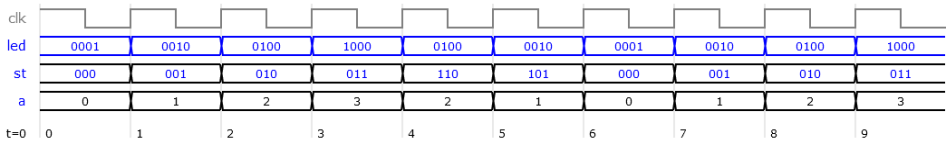
Primer za prva dva prehoda:

```
if st="000" then st<="001";
elsif st="001" then st<="010";
...
end
```

Vhod v dekodirnik predstavljata spodnja dva bita registra stanj. Dodaj med deklaracije še dvobitni signal **a** in zapiši stavek, ki dodeli signalu spodnja dva bita registra stanj:

```
a = st(1 downto 0);
```

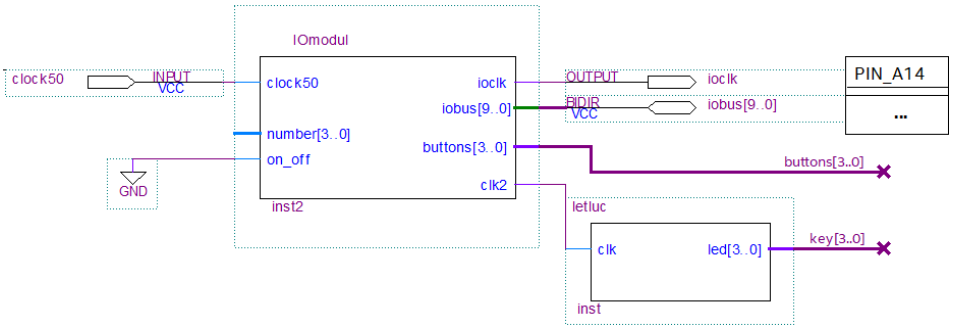
Dodati je potrebno le še opis dekodirnika, ki kombinacije signala **a** preslika v 4-bitni izhod **led**. Dekodirnik je kombinacijsko vezje, zato uporabljamo navaden prireditveni operator =. Slika 6.6 predstavlja simulacijo vezja v orodju SHDL, kjer smo prikazali stanje (st) in izhod (led) v binarni obliki. Način prikaza signala menjamo z desnim klikom na signal.



Slika 6.6: *Simulacija vezja za učinek potujoče luči.*

6.4 Kaj morate narediti vi?

- Izpolnite tabelo prehajanja stanj glede na diagram prehajanja stanj
- Odprite vzorec projekta v orodju **Quartus** in pobrišite simbol RND.
- Narišite shemo vezja pri kateri ne pozabite določiti vsebine ROM, ali pa naredite model vezja v orodju SHDL in prenesite izhodno datoteko v Quartus.
- Vezje pretvorite v nov simbol, ki ga dodajte na glavno shemo in povežite z LED. Uro povežite na počasen signal clk2, da boste lahko opazovali delovanje.
- Prevedite vezje in ga preizkusite na razvojni plošči



Slika 6.7: *Glavna shema s komponento vezja za učinek potujoče luči.*

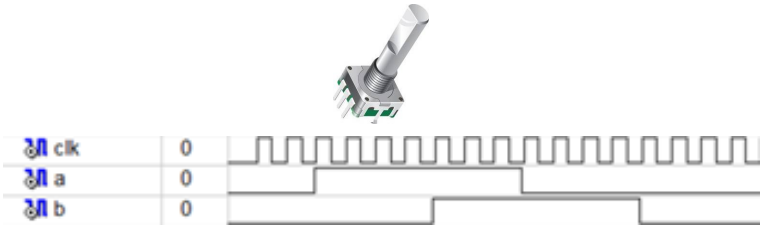
Vaja 7

Rotacijski kodirnik

Pri tej vaji bomo načrtovali sekvenčno vezje za štetje korakov rotacijskega inkrementalnega kodirnika. Naredili bomo sekvenčni stroj, ki zazna korake vrtenja kodirnika v desno oz. levo smer in nastavlja digitalni števec.

7.1 Rotacijski inkrementalni kodirnik

Rotacijski inkrementalni kodirnik ima dva izhoda, **a** in **b**, na katerih dobimo med seboj zamaknjene impulze ob vrtenju kodirnika, ki naredi 24 korakov na obrat. Slika 7.1 prikazuje časovni potek izhodov pri enem koraku vrtenja v desno, kjer se najprej postavi na 1 izhod **a**, nato pa še **b**. Korak v nasprotni smeri povzroči obratno zaporedje izhodnih signalov.



Slika 7.1: *Rotacijski kodirnik in časovni potek izhodov **a** in **b**.*

Iz kodirnika želimo pridobiti informacijo, kdaj se je obrnil za en korak v

določeno smer. Vrednosti rotacijskega kodirnika zaznavamo ob fronti ure clk. Na sliki 7.2 je primer vrednosti ob posameznih ciklih ure v primeru vrtenja za en korak v desno (cikli 2 do 9) in nato v levo (12 do 18).

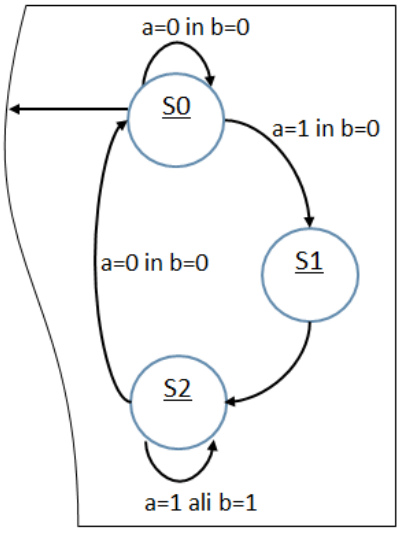
a	0	1	1	1	1	0	0	0	0	0	0	0	1	1	1	1	1	0	0
b	0	0	0	0	1	1	1	1	1	0	0	1	1	1	1	0	0	0	0
cikel	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Slika 7.2: *Časovni potek izhodov pri vrtenju za en korak v desno in v levo*

Samo iz vrednosti izhodov, ne moremo določiti smer vrtenja, saj se npr. kombinacija **a**=1 in **b**=0 pojavlja pri vrtenju v eno in v drugo smer. Zato naredimo sekvečni stroj, ki bo zaznaval zaporedje.

7.2 Diagram stanj

Zasnovali bomo sekvenčni stroj za zaznavanje zgoraj podanega zaporedja izhodov **a** in **b**, ki ga dobimo ob vrtenju kodirnika za en korak. Slika 7.3 prikazuje del diagrama stanj za zaznavanje vrtenja v eno smer.



Slika 7.3: *Delni prikaz diagrama stanj za detekcijo vrtenja rotacijskega kodirnika*

Stanje **S0** je mirovno stanje, iz katerega se ob **a**=1 in **b**=0 premaknemo v **S1**. Ob naslednjem ciklu ure se premaknemo v stanje **S2**, kjer čakamo dokler je katerikoli izmed vhodov na 1, nato pa gremo nazaj v mirovno stanje. Prikazan diagram stanj zazna prehod iz **a**=0 in **b**=0 v **a**=1 in **b**=0, ki je značilen za en korak vrtenja v desno smer. Ob tem prehodu gremo v stanje S1 za en cikel ure in to stanje lahko izkoristimo za povečevanje števca korakov.

7.3 Vezje za štetje korakov v SHDL

Vezje za štetje korakov bomo najprej opisali in simulirali v strojno-opisnem jeziku. Orodje SHDL vključuje simulator razvojne plošče s perifernimi enotami, med katerimi je rotacijski kodirnik (*Setup, Board*). Model vezja povežemo s simulatorjem tako, da deklariramo priključke vezja z vnaprej določenimi imeni: **rot** je 2-bitni signal rotacijskega kodirnika, **bcd** pa izhod za 7-segmentni prikazovalnik. Stanje sekvenčnega stroja bomo deklarirali z naštevnim podatkovnim tipom, kjer zapišemo imena stanj:

```
entity rotac
 rot: in u2;
 bcd: out u4;
 st: (s0,s1,s2,s3);
```

Deklariramo lahko še dva enobitna notranja signala **a** in **b**, ki ju priredimo posameznemu bitu vektorja **rot**. Prehode diagrama stanj opišemo z zaporedjem pogojnih stavkov, prehod iz **s0** v **s1** opišemo takole:

```
 a = rot(0); b = rot(1);
 if st=s0 then
   if a=1 and b=0 then st<=s1;
```

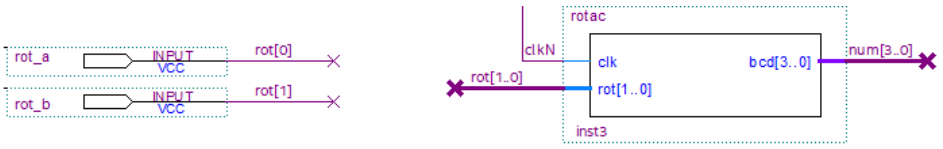
V ustreznem stanju bomo za 1 povečali oz. zmanjšali števec **bcd**. Paziti moramo, da bomo za sekvenčne signale (stanje, števec) uporabljali sekvenčni prireditveni operator <= v jeziku SHDL.

7.4 Kaj morate narediti vi?

Dopolnite diagram stanj, da bo zaznaval tudi prehod iz **a**=0 in **b**=0 v **a**=0 in **b**=1 ob vrtenju kodirnika v levo. Preverite, da bo v vseh stanjih definirano kaj se zgodi ob vseh kombinacijah vhodov. Nekatere kombinacije so lahko nepričakovane, npr. iz **a**=0 in **b**=0 v **a**=1 in **b**=1, vseeno pa je dobro, da jih predvidimo v diagramu (v tem primeru naprimer kar ostanemo v stanju **S0**).

Opišite delovanje vezja za štetje korakov v jeziku SHDL in preverite delovanje s simulacijo razvojne plošče.

V orodju Quartus odprite vzorčni projekt in prenesite v projekt VHDL kodo, ki jo generira orodje SHDL. Naredite nov shematski simbol in ga vključite v glavno shemo. Ura naj bo povezana na signal **clkN** s frekvenco 1 kHz, izhod števca pa na matrični LED prikazovalnik.



Slika 7.4: *Vezava sekvenčnega vezja za štetje korakov na glavni shemi.*

Razmisli

- Koliko flip-flopov potrebujemo za opis sekvenčnega stroja za zaznavanje vrtenja rotacijskega kodirnika?
- Kako bi opisal izhodno logiko, ki bi generirala impulz ob vsakem koraku v eno ali drugo smer?

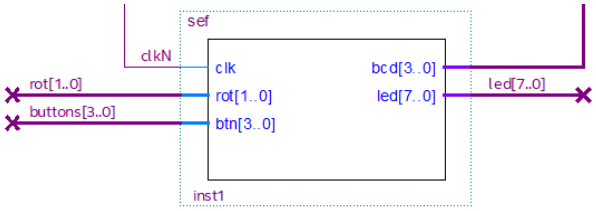
Vaja 8

Starinski sef z elektronsko ključavnico

Pri tej vaji bomo naredili vezje, ki bo simuliralo delovanje starinskega sefa z vrtljivo števílčnico. Sekvenčno vezje za elektronski sef omogoča nastavljanje skrivnega ključa z rotacijskim kodirnikom, zaklepanje in odklepanje sefa, ki ga simuliramo s prikazovanjem stanja na svetlečih diodah.

8.1 Nadgradnja vezja za rotacijski kodirnik

Nadgradili bomo model vezja za štetje korakov rotacijskega inkrementalnega kodirnika, ki smo na naredili pri prejšnji vaji. Dodali bomo vhodni signal **btn** za tipke in izhod **led** za indikacijo delovanja vezja, kot prikazuje slika 8.1.



Slika 8.1: *Sekvenčno vezje za simulacijo sefa.*

Vezje iz prejšnje vaje vsebuje sekvenčni stroj za dekodiranje vrtenja in števec. Dodali bomo register za shranjevanje skrivne kombinacije **key** in flip-flop za stanje ključavnice **lock**.

8.2 Vaša naloga

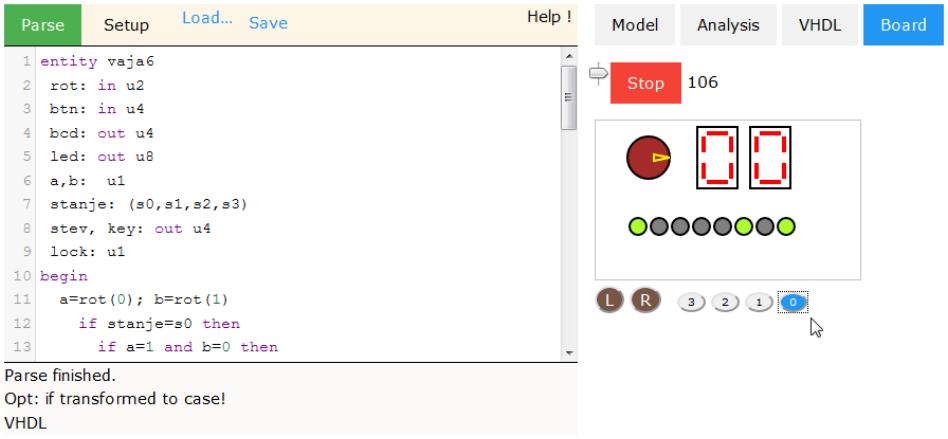
Uporabite model vezja za rotacijski kodirnik iz prejšnje vaje. Med priključke vezja dodajte 4-bitni vhod **btn** za stanje tipk, 8-bitni izhod **led**, 4-bitni notranji signal **key** in enobitni **lock**. V spletnem orodju SHDL dodajte deklaracije:

```
btn: in u4
led: out u8
key: u4
lock: u1
```

Nove signale moramo dodati še v tabelo priključkov, da bodo povezani s simulatorjem. Pri simulaciji bomo opazovali stanje ključavnice in shranjeno kombinacijo na LED. Stanje ključavnice bomo povezali na zgornjo LED, spodnje štiri pa bodo prikazovale skrivno kombinacijo. Izraz za led sestavimo iz teh dveh signalov in treh ničel, da dobimo 8-bitni vektor:

```
led = lock & "000" & key
```

Slika 8.2 prikazuje simulacijo sefa, kjer vidimo da je zaklenjen (skrajno leva LED) in da ima nastavljeno kombinacijo 0101.



Slika 8.2: *Simulacija sefa z nastavljeno kombinacijo 0101.*

Dodajte opis nastavljanja kombinacije sefa in odpiranja ključavnice. Z vrtenjem rotacijskega kodirnika nastavljamo števec. Ob pritisku na prvo tipko **btn(0)** in pogoju, da je skrivni ključ 0, števec pa večji od 0 se:

- vrednost števca shrani v register skrivne kombinacije,
- števec postavi na 0 in
- stanje ključavnice spremeni na 1 (zaklenjeno).

Odpiranje sefa bomo opisali s pogojem, da je pritisnjena druga tipka **btn(1)** in da se števec in skrivna kombinacija ujemata. Ob uspešnem odpiranju resetiramo skrivno kombinacijo in stanje ključavnice na 0, tako da bo sef pripravljen za nastavljanje nove kombinacije.

Dokončaj opis vezja ter preizkusi delovanje s simulacijo v spletnem orodju.

Razmisli

- Kateri signali v opisu sekvenčnega vezja predstavljajo flip-flope ali registre?
- Zakaj smo določili pogoj, da je števec večji 0, pri nastavljanju skrivne kombinacije?

Vaja 9

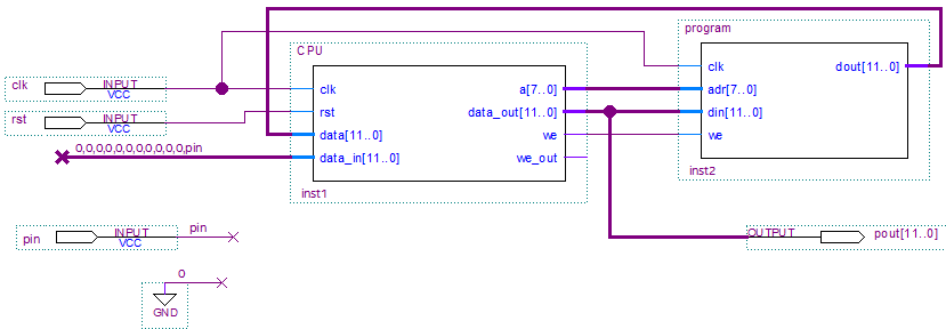
Mikroprocesor CPE4

Pri vaji boste uporabljali učni mikroprocesor, ki je narejen z logiko v vezju FPGA. Pomemben del načrtovanja takšnega vezja je načrtovanje kode, ki naj teče na procesorju. V projekt boste kodo v zbirnem jeziku vnesli s pomočjo spletnega orodja v katerem je mogoče izvajanje programov tudi simulirati.

9.1 Gradnik: procesorski blok

Procesorski blok temelji na majhni 12-bitni centralno procesni enoti CPE4, ki je narejena v strojno-opisnem jeziku VHDL. Procesor pozna 2^4 različnih strojnih ukazov za obdelavo 12-bitnih podatkov. Procesorski blok ima 12-bitno vhodno in izhodno vodilo za povezavo v digitalni sistem.

Poleg CPE4 je v bloku še pomnilnik RAM v katerem so shranjeni strojni ukazi in spremenljivke. Tudi pomnilnik je opisan v jeziku VHDL in ob prevajanju vezja že vsebuje program za mikroprocesor.



Slika 9.1: *Procesorski blok s centralno procesno enoto in pomnilnikom.*

Slika 9.1 prikazuje gradnik ProcBlok, ki je sestavljen iz centralne procesne enote in pomnilnika. Gradnik ProcBlock ima naslednje priključke:

- **clk** - vhodna ura (procesor je sekvenčno vezje!),
- **rst** - signal reset, ob 1 se procesor ponastavi na začetek programa (naslov 0),
- **pin** - vhodno vodilo, na katerega je povezan le en bit in
- **pout [11..0]** - 12-bitno izhodno vodilo.

Procesor in pomnilnik sta opisana v datotekah:

- *proc.vhd* - opis procesorskega jedra.
- *procpak.vhd* - definicije zbirniških ukazov, ki jih pozna procesor.
- *program.vhd* - opis pomnilnika in njegova vsebina. To datoteko urejate, ko želite v procesor vpisati program.

9.2 Program v zbirnem jeziku

Na spletni strani: https://lniv.fe.uni-lj.si/cpe/ je urejevalnik zbirne kode in simulator 12-bitnega procesorja. Procesor je zelo enostaven in pozna le 16 ukazov s katerimi prenašamo podatke, računamo in izvajamo skoke. Podatki prihajajo iz pomnilnika (spremenljivke) ali pa vhodnega vodila. Rezultat ukazov se shrani v register z imenom **akumulator**.

Ukaze zapisujemo v zbirniku vsakega v svojo vrstico, ki se začne z oznako ali pa s presledkom (oz. tabulatorjem). Oznake potrebujemo za skočne ukaze, npr. ukaz **jmp start** bo izvedel skok na ukaz, pred katerim stoji oznaka **start:**

Poglejmo primer majhnega programa, ki v neskončni zanki bere vrednost iz vhoda, jih prišteje spremenljivki **n** in shrani na izhod:

```
start:  inp vh
        add n
        sta n
        outp izh
        jmp start
vh      di 0
izh     do 0
n       db 0
```

V prvi vrstici beremo podatek iz vhodnega vodila, v naslednji pa mu prištejemo vrednost spremenljivke **n**. Rezultat bo v akumulatorju, zato potrebujemo še ukaz **sta**, s katerim shranimo vrednost spremenljivke, ki se nahaja v pomnilniku.

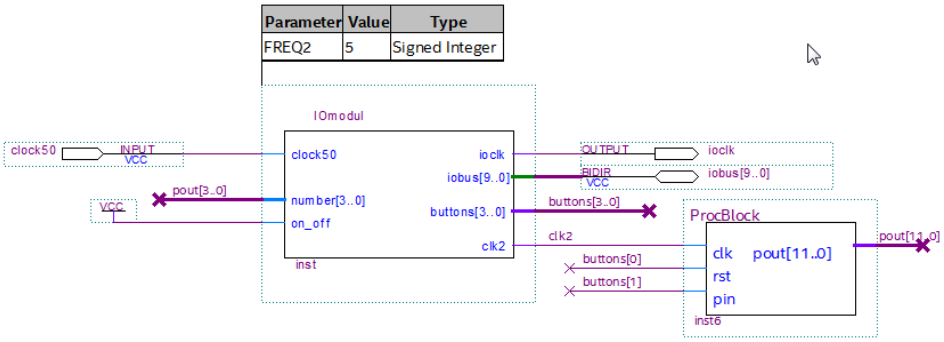
S stavkom **outp** prenesemo vrednost na izhodno vodilo, da jo bomo lahko opazovali na prikazovalniku. Program zaključimo s skokom na začetek, ki povzroči ponavljanje opisanega algoritma. Procesor stalno izvaja ukaze, saj nima ukaza za zaustavitev ali spanje, zato mora biti program vedno napisan v zanki.

Na koncu kode so deklaracije vhodnih spremenljivk **di**, izhodnih spremenljivk **do** in notranjih spremenljivk programa **db**. Številka za oznako **db** določa začetno vrednost spremenljivke; v našem primeru ima spremenljivka **n** začetno vrednost 0.

Delovanje programa preizkušamo v simulatorju, kjer napišemo zbirno kodo, jo prevedemo (**Start**) in izvajamo po korakih (**Korak**). Med izvajanjem nastavljamo vrednosti vhoda ter opazujemo akumulator in vsebino pomnilnika.

9.3 Shema procesorskega sistema

Pri vaji boste dobili že narejeno shemo sistema, kjer je procesorski blok povezan na IOmodul in LED. Prva tipka je povezana na signal reset, druga pa na vhodno vodilo. Procesor je povezan na počasno uro **clk2** s frekvenco 5 Hz, da bomo na vezju lažje opazovali spreminjanje izhodov. Ura in tipke so veznane na LED, izhodno vodilo procesorja pa na matrični prikazovalnik (opazujemo le najnižje štiri bite).



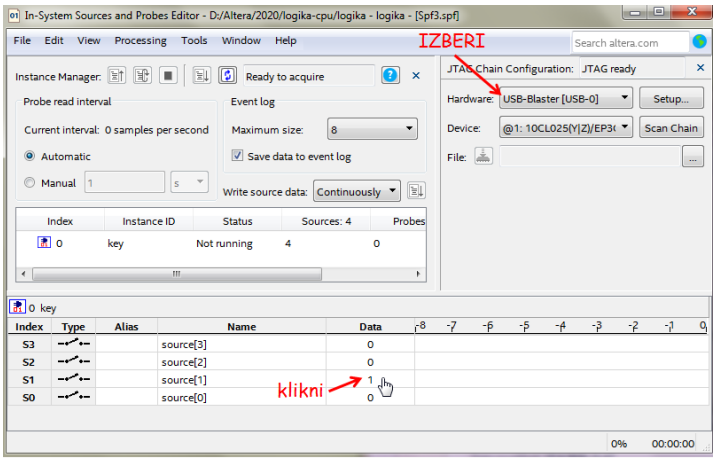
Slika 9.2: *Sistem z vmesnikom in procesorskim blokom.*

9.4 Kaj morate narediti vi?

Najprej preizkusite delovanje procesorja na simulatorju. Prepišite program v urejevalnik in na spletnem simulatorju opazujte izvajanje programa po korakih. Nastavite vrednost vhodnega vodila na 1, tako da v okencu Vhodi vnesete vrednost 1 desno od vh. Opazujte, kako se spreminja vrednost v akumulatorju.

Odprite arhiv programa Quartus v katerem je digitalni sistem s procesorjem. Preglejte glavno shemo logika.bdf in shemo procesorskega bloka, ki jo lahko odprete z desnim klikom in Open Design File.

Prevedite sistem in preizkusite delovanje na razvojni plošči. Pritisnite tipko S2 oz. nastavite virtualno tipko source[1] (slika 9.3) in opazujte kako se spreminja vrednost na matričnem LED.



Slika 9.3: *Nastavitev virtualnih tipk: izberi Tools, In-System Sources, nastavi Hardware: USB Blaster in klikni Data pri ustrezni tipki.*

Ura procesorja je nastavljena na 5 Hz. Če želite, jo spremenite z nastavitvijo parametra FREQ2 v shemi vezja. Odprite datoteko program.vhd in preglejte vsebino pomnilnika:

```
18  signal m : memory := (
19  -- sem pisete vas program
20  0=> lda & x"08",
21  1=> sta & x"07",
22  2=> inp & x"00",
23  3=> add & x"07",
24  4=> sta & x"07",
25  5=> outp & x"00",
26  6=> jmp & x"02",
27  7=> x"000",
28  8=> x"000",
29  -- tu se konca vas program
30  others => x"000"
31  );
```

Na matričnem prikazovalniku opazujemo vrednost izhodnega vodila, ki se spreminja med izvajanjem programa. Na vodilo je povezana kar vsebina akumulatorja. Po branju iz vhoda (**lda**) je v akumulatorju 0 ali 1, po prištevanju spremenljivke **n** pa ustrezna vsota. Če želimo, da bo na izhodu prikazana le vsota, moramo v vezje dodati register. Z registrom bomo izhodno vodilo spremenili v izhodna vrata (angl. port), ki spreminjajo vrednost le ob izvedbi ukaza **outp**.

Na shemo procesorskega bloka ProcBlock.bdf dodajte 12-bitni register reg12. Povežite podatkovni vhod na izhodno vodilo procesorja (data_out) in izhod na zunanje priključke (pout). Register naj ima uro povezano na priključek clk in signal load na we_out (ta signal se postavi, ko procesor piše na zunanje vodilo). Prevedite sistem in preizkusite delovanje na razvojni plošči.

Razmisli

Opazuj, kako deluje signal reset (prva tipka). Koda iz poglavja 8.2 in vsebina programskega pomnilnika se nekoliko razlikujeta, saj sta v pomnilniku na začetku še dva dodatna ukaza. Razloži zakaj ju potrebujemo.

Vaja 10

Pulzno-širinski modulator

Pri tej vaji boste spoznali koncept pulzno-širinske modulacije (PWM - Pulse Width Modulation). Načrtali boste vezje, ki bo krmililo svetlost LED diod s pulzno-širinsko modulacijo. Komponento vezja boste naredili v jeziku SHDL in simulirali v spletnem orodju, nato pa jo priključili na mikroprocesorski sistem in napisali program za spreminjanje svetlosti LED.

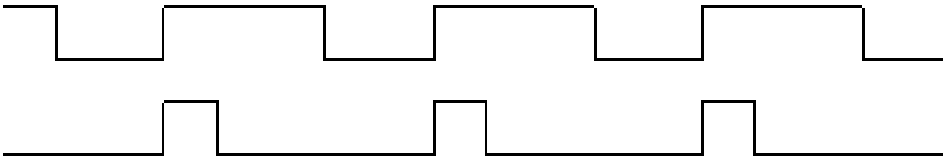
10.1 Pulzno-širinska modulacija

Pulzno-širinska modulacija nam omogoča, da zgolj s pomočjo digitalnih gradnikov na izhodu vezja generiramo signale različne moči. Tako lahko brez uporabe digitalno-analognih pretvornikov zvezno krmilimo naprave, naprimer spreminjamo hitrost servo motorjev, svetilnost LED diod in podobno.

Vzemimo digitalen pravokoten signal frekvence f (slika 10.1), ki je polovico časa na visokem nivoju, polovico časa pa na nizkem nivoju. Do porabnika (npr. LED) se bo prenesla polovica vse razpoložljive moči. Če *pri isti frekvenci* spremenimo delež časa, ko je signal na visokem nivoju, lahko to moč povečamo ali zmanjšamo, kot to prikazuje slika 10.2.

Pri tem se spreminja samo delež časa, ko je signal v visokem nivoju, frekvenca signala pa ostaja enaka (primerjajte zgornje poteke signala). Po-

Slika 10.1: *Pravokoten signal, ki je polovico časa na visokem nivoju prenese porabniku polovico razpoložljive moči.*



Slika 10.2: *Dva pravokotna signala, ki porabniku preneseta različen delež razpoložljive moči: zgornji več kot polovico, spodnji manj kot polovico.*

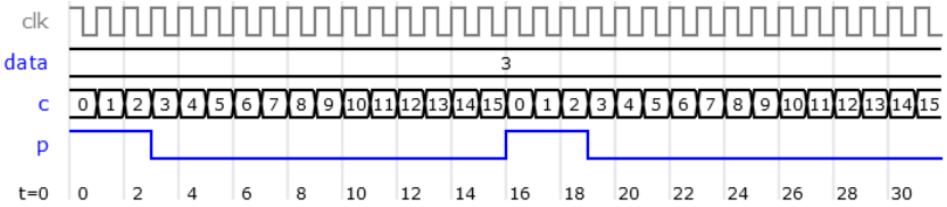
membno je naslednje: *frekvenca pulzno-širinskega modulatorja mora biti dovolj visoka, da ne moti delovanja naprave, ki jo krmilimo.*

V našem primeru bomo s pomočjo pulzno-širinske modulacije krmilili svetlost LED diode. To pomeni, da bo dioda del časa prižgana, del časa pa ugasnjena, vendar mora biti frekvenca dovolj hitra, da nas to ne bo motilo, torej da ne bomo več opazili utripanja diode, ampak samo to, da sveti temneje ali svetleje.

10.2 Vezje pulzno-širinskega modulatorja

Vezje pulzno-širinskega modulatorja vsebuje števec in primerjalnik. Vzemimo 4-bitni števec **c**, ki se spreminja ob fronti ure, kot prikazuje slika 10.3. Izhodnemu signalu **p** določimo vrednost 1, ko je vrednost števca manjša od vrednosti na vhodu **data**, v ostalih ciklih ure pa naj bo 0.

Ker se števec ciklično spreminja (med 0 in 15), bomo dobili pri konstantno nastavljenem vhodu na izhodu periodične impulze. Pri višji vrednosti vhoda, bo izhod dalj časa na 1, pri nižji pa manj. Modulatorju lahko dodamo še negiran izhod **q** za lepši učinek na razvojni plošči.



Slika 10.3: *Izhod PWM določimo s primerjavo vhodne vrednosti in števca.*

10.3 Vaša naloga

- Naredite model 4-bitnega pulzno-širinskega modulatorja z imenom **pwm** v spletnem orodju: https://lniv.fe.uni-lj.si/shdl/ Vezje ima 4-bitni vhod **data** in enobitna izhoda **p** in **q**.
- Delovanje vezja preverite na simulaciji, kjer nastavite konstantno vhodno vrednost za vsaj 32 period ure. Simulacijo izvedite večkrat in vsakokrat naj bo nastavljena drugačna vrednost vhoda **data**, npr. slika 10.3 prikazuje izhod **p** kadar je **data** enak 3.
- V programu Quartus odprite mikroprocesorski sistem iz zadnje vaje. Odprite novo datoteko vrste VHDL (File, New, VHDL File) in v urejevalnik kopirajte izhodno VHDL kodo iz spletnega orodja. Datoteko shranite z imenom **pwm.vhd** in dodajte v projekt (Project, Add Current File to Project).
- Naredite shematski simbol komponente **pwm** (File, Create, Create Symbol Files) in jo dodajte na glavno shemo. Podatkovni vhod povežite z izhodom procesnega bloka, oba izhoda komponente pa povežite na LED. Ura naj bo vezana na hitro uro clock50.
- Najprej prevedite sistem z obstoječim programom, ki daje na izhod zaporedne vrednosti ob pritisku na tipko S1 in preizkusite delovanje na razvojni plošči.
- Napišite program v zbirniku, ki ciklično spreminja vrednost v izhodnem registru: najprej jo povečuje od 0 do 15, nato zmanjšuje do 0 in vse skupaj ponavlja. Preverite delovanja na simulatorju, prenesite kodo v Quartus (program.vhd) in preizkusite na razvojni plošči.

Za pomoč pri pisanju programa smo pripravili del programa v zbirniku, ki povečuje vrednost spremenljivke n od 0 do 15 in jo vsakokrat pošlje na izhod. Ko pride do 15 pa skoči na drugi del programa, ki ga morate sami napisati.

```
start:  lda n
        outp izh
        add 1
        sta n
        sbt 15
        jze drugi
        jmp start
drugi:
n       db  0
izh     do  0
```

Razmisli

Pulzno-širinski modulator deluje pravilno pod pogojem, da se vhod spreminja počasneje, kot je cikel štetja (16 period ure).

- Kaj se zgodi z izhodnim impulzom, če se vhod spremeni sredi cikla štetja?
- Kaj je potrebno narediti, da bo imela sprememba vrednosti vhoda vpliv na izhod šele ob naslednjem ciklu štetja?
- Pri najvišji vrednosti vhoda še vedno ne dobimo na izhod celotne moči, ker je izhod eno periodo na 0. Kako bi to popravili?

Vaja 11

Krmiljenje matrike LED

Naredili bomo vezje za vklapljenje posamezne svetleče diode na matriki 5x7 LED. Spoznali bomo princip multipleksiranja, na katerem temeljijo video prikazovalniki. V mikroprocesorski sistem bomo dodali vmesnik za krmiljenje matrike LED in napisali program, ki bo izvajal enostavno animacijo.

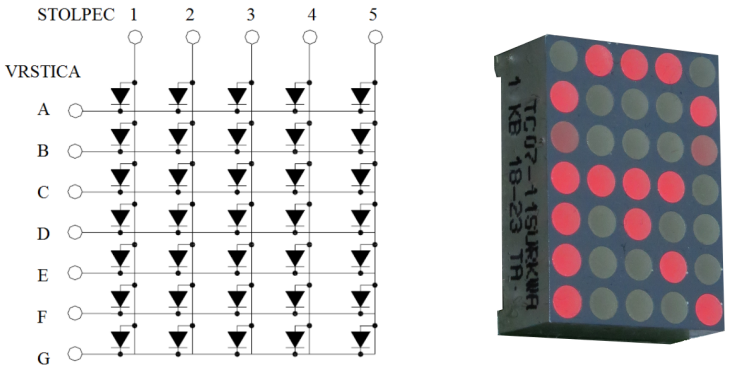
11.1 Uporaba matrike LED

Grafični prikazovalniki uporabljajo multipleksiranje za prenos podatkov in osveževanje slikovnih točk. Slika 11.1 prikazuje povezavo LED v matriki 35 svetlečih diod, ki so razporejene v pet stolpcev in sedem vrstic. Za prikaz poljubne kombinacije LED moramo izmenično vklapljati posamezne vrstice ali stolpce. Preklapljanja ne bomo zaznali, če ga izvajamo dovolj hitro (npr. s frekvenco vsaj 100 Hz).

Na razvojnem sistemu imamo matriko 5x7 LED povezano s krmilnikom razširitvene plošče. Posamezne vrstice vklapljamo oz. osvežujemo s pisanjem podatkov na paralelni vmesnik. Vmesnik periodično sprejema podatke o izbrani vrstici (3 biti) in stanju LED izbrane vrstice (5 bitov). Komponenta IOmodul v vezju FPGA razdeli sliko izbrane števke na vrstice in jih prek vmesnika pošilja na prikazovalno matriko.

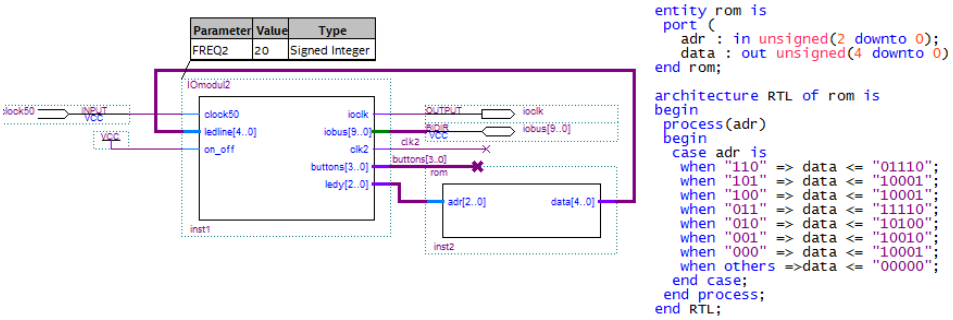
Tokrat bomo uporabili komponento **IOmodul2**, ki omogoča prikaz po-

ljubne slike na matriki LED. Komponenta posreduje na 2-bitnem izhodu **ledy** stanje števca za osveževanje vrstic matrike. Na 5-bitni vhod **ledline** moramo prenesti posamezno vrstico slike.



Slika 11.1: *Povezava LED v matriki 5 x 7 svetlečih diod.*

Vezje za prikazovanje mirujoče slike naredimo s pomočjo pomnilnika ROM. Na sliki 11.2 je prikazana shema vezja s komponento IOmodul2 in pomnilnikom ROM, kjer vidimo, da je števec vrstic povezan na naslovni vhod pomnilnika. Prikazan je opis pomnilnika v jeziku VHDL, ki na matriki prikazuje črko R.



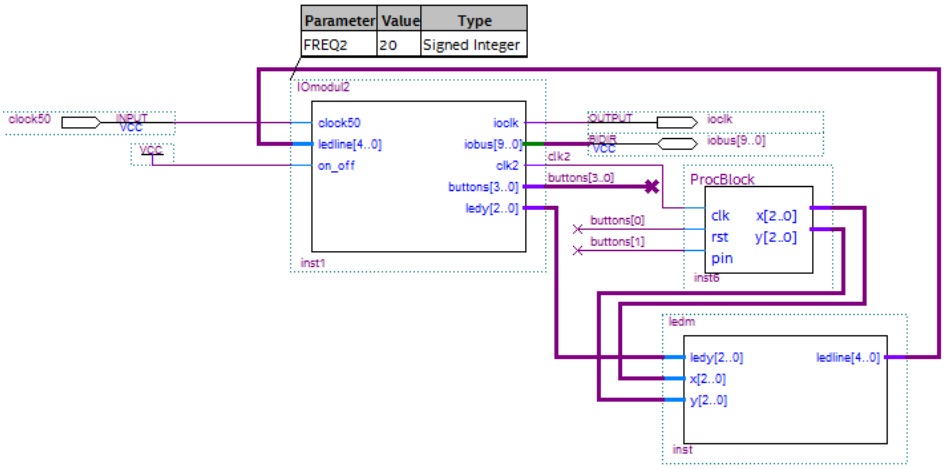
Slika 11.2: *Komponenta IOmodul2 in ROM za prikaz mirujoče slike.*

Prikazovanje vnaprej pripravljenih sličic naredimo tako, da jih zapišemo v večji pomnilnik ROM in dodamo logiko za določanje pomnilniških naslovov. V komponenti IOmodul, ki smo jo uporabljali do sedaj, so v pomnilniku zapisane slike šestnajstiških števk. S signalom **num** izberemo ustrezen segment pomnilnika za prenos na matrikoc.

11.2 Prikazovanje točke

Naredili bomo vezje za prikazovanje točke na poljubnih koordinatah matrike LED. V vezju bo komponenta **IOmodul2** povezana s kombinacijskim dekodirnikom **ledm**.

Koordinate določata 3-bitna vhodna signala **x** in **y**. Na vhodu dekodirnika je še 3-bitno stanje števca vrstic **ledy**. Ko je koordinata y enaka stanju števca, naj bo na 5-bitnem izhodu **ledline** dekodirana vrednost koordinate x, sicer pa naj bo izhod na 0. Na matriki bo tako svetila le LED, ki je na koordinatah (x,y).

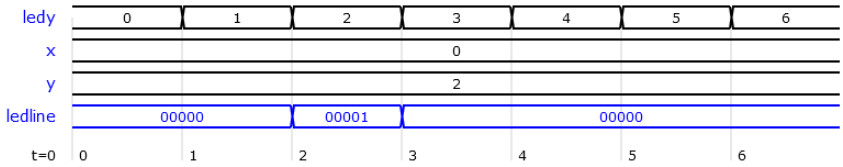


Slika 11.3: *Vezje za prikazovanje točke na matriki LED.*

Da bo delovanje vezja bolj zanimivo, bomo koordinate določali s procesorskim sistemom. Vmesnik procesorskega bloka ima tokrat dva 3-bitna izhoda, ki ju lahko programsko nastavljamo. S programom v zbirniku bomo naredili animacijo prikazovanja točke na matriki LED.

11.3 Kaj morate narediti vi?

- V jeziku SHDL naredite opis kombinacijskega dekodirnika za prikazovanje točke z imenom **ledm**. Preverite delovanje s simulacijo v spletnem orodju.



- Naložite iz učilnice arhiv projekta v orodju Quartus, ki vsebuje komponento **IOmodul2** in procesorski sistem z dvema izhodoma.
- Odprite novo datoteko vrste VHDL (File, New, VHDL File) in v urejevalnik kopirajte izhodno VHDL kodo iz spletnega orodja. Datoteko shranite z imenom **ledm.vhd**, dodajte v projekt (Project, Add Current File to Project) in naredite simbol (File, Create, Create Symbol Files).
- Dokončajte shemo vezja, kot prikazuje slika 11.3. Preizkusite delovanje s programom, ki šteje od 0 do 4 in s pisanjem v izhodni vmesnik nastavlja koordinati x in y.

```
start:  lda x
        outp 0
        outp 1
        sbt 4
        jze x0
        lda x
        add 1
        sta x
        jmp start
x0:     lda 0
        sta x
        jmp start
x       db 0
```

- Poskusite napisati še svoj program, ki naredi kakšno zanimivo animacijo!

Razmisli

Kaj bi bilo potrebno narediti, da bi se obe koordinati spremeni naenkrat in bi točka potovala po diagonali?

Vaja 12

Generiranje signala VGA

12.1 Povzetek naloge

Pri tej boste spoznali, da se da z vezjem FPGA generirati tudi sorazmerno kompleksne signale z visoko frekvenco. Pri nas bo to signal na izhodnem vhodu VGA, ki ga boste priklopili na monitor. Rezultat te vaje bomo opazovali na vašem monitorju, ki ga boste, da boste videli signal iz razvojne ploščice, preklopili v VGA način. V vhodno/izhodnem bloku, ki smo ga pripravili za to vajo, je zbrana vsa potrebna nizkonivojska logika za krmiljenje VGA izhoda.

12.2 Signal VGA

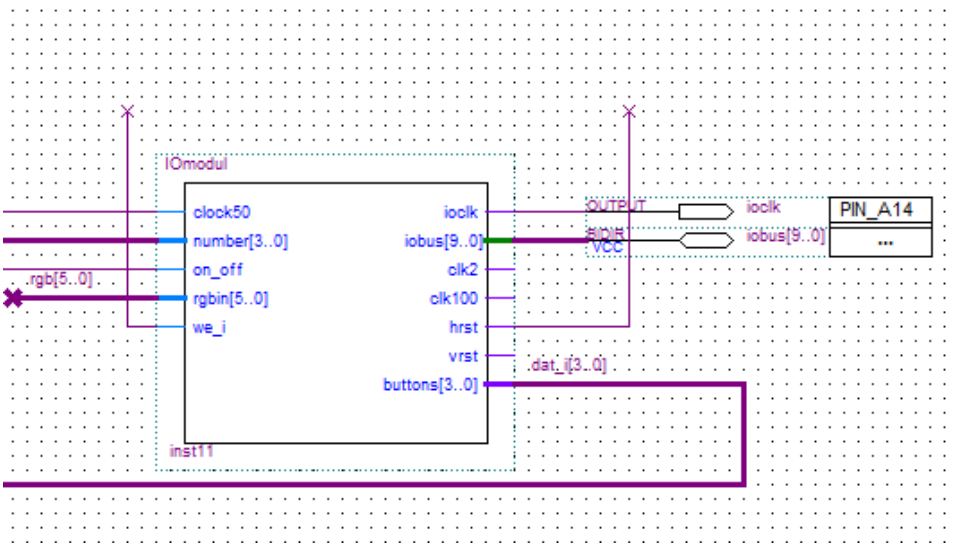
Sliko na monitorju prikažemo kot matriko barvnih elementov (pikslov). Vsak piksel ima tri komponente: rdečo, zeleno in modro. Kvaliteta slike je določena s številom pikslov na površino monitorja, ter z njihovo barvno globino; oba parametra sta pri naši vaji razmeroma nizka.

Barvo in svetlost vsakega piksla zapišemo z šestimi biti: dva bita za vsako barvo. Kar pomeni, da imamo štiri možne vrednosti za vsako od R G B barvnih komponent. Na ta način dobimo 4x4x4 = 64 možnih barv za vsak piksel. Za prikaz slike moramo te šestbitne vrednosti z ustreznim tempom

pošiljati na izhod; slika na monitorju nastaja po vrsticah, od leve proti desni. Torej, če vemo, da sta za vsako barvo rezervirana dva bita, barve pa so zakodirane v zaporedju (najnižja dva bita R, srednja dva bita G, najvišja dva bita B), bomo razumeli, kako je sestavljena prikazana slika. Na primer:

```
110100 111111 000011 000000
```

Slika, predstavljena z bitnim vzorcem zgoraj, je dejansko sestavljena iz štirih pikslov, od katerih je zgornji levi mešanica slabe zelene in močne modre svetlobe, desni zgornji je bel (vse vrednosti R G in B na najvišjih vrednostih, spodnji levi je rdeč, spodnji desni pa črn). VGA signal deluje po principu rasterskih vrstic: monitor sliko izrisuje po vrsticah od leve proti desni, od zgoraj navzgor, kar izhaja še iz tehnologije katodnih cevi (CRT), kjer je elektronski žarek dejansko potoval po tem vzorcu , in prikazoval en element naenkrat. Moderni LCD monitorji ta način samo posnemajo zaradi združljivosti z VGA napravami.



Slika 12.1: *Vhodno-izhodni modul s vhodi in izhodi za signal VGA.*

Za prikazovanje VGA slike so pomembne naslednje tri sponke, ki jih kaže slika 12.1 :

- **hrst** – horizontalni sinhronizacijski signal, sproži se na začetku vsake rastrske vrstice slike
- **vrst** – vertikalni sinhronizacijski signal, sproži se na vrhu vsake slike
- **rgbin[5..0]** - vhod za barvo in svetilnost piksla, ki ga "žarek"trenutno "riše". Kot že rečeno, ima vsak od R, G, B barvnih kanalov na voljo 2 bita.

V našem primeru bomo hrst vezali na reset (**rst**) vhod procesorja, kar pomeni da se bo program na procesorju začel izvajati na začetku vsake vrstice. To tudi pomeni, da bodo vse vrstice enake – zaradi preprostosti bomo ≫ustvarjali≪ le vertikalne vzorce. Sliko bomo pa ustvarili tako, da bomo ob pravem času na **rgbin[5..0]** pošiljali ustrezne vrednosti barve pikslov.

12.3 Kaj morate narediti vi?

Iz spletne učilnice prenesite predlogo za to vajo. Vsebuje že procesorski blok, programski blok, vhodno/izhodni blok z elementi za generiranje signala VGA ter vse povezave, poleg tega pa tudi že vsebuje sklop za dekodiranje signala rotacijskega kodirnika, ki ga že poznate. Vaše delo bo le pisanje programa. Napisati morate program, ki bo narisal vertikalno črto ali zaporedje vertikalnih črt, pri čemer naj je odmik od levega roba sorazmeren z pozicijo rotacijskega kodirnika.

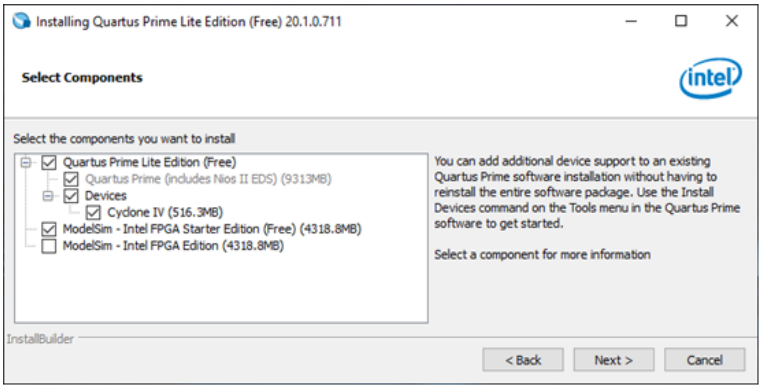
Dodatek: namestitev orodij

Proizvajalci programirljivih vezij nudijo za izobraževanje brezplačna orodja s katerimi prevajamo in programiramo njihova vezja. Pred prenosom datotek se je potrebno na spletni strani registrirati. Povezava na **Quartus Prime**: https://www.altera.com/products/design-software/fpga-design/quartus-prime/download.html

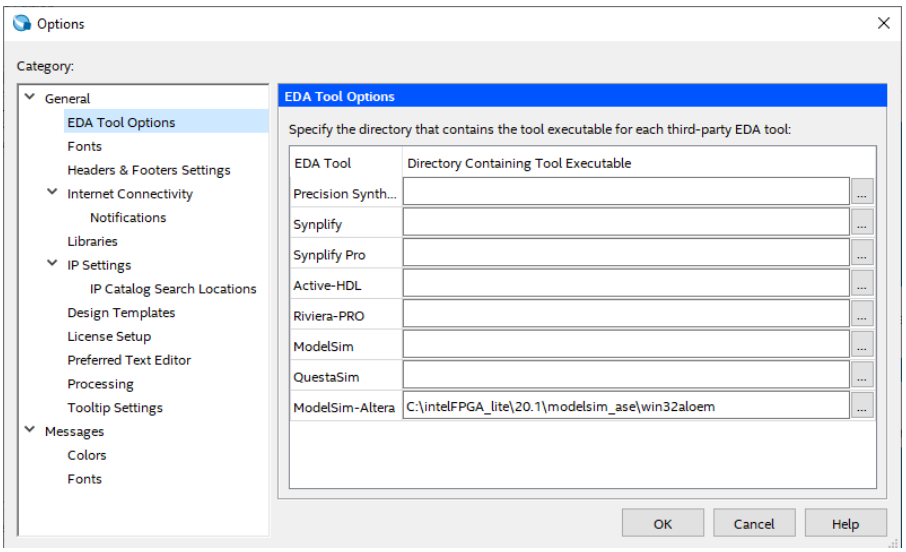
Brezplačna različica se imenuje Lite Edition. Priporočamo, da izberete verzijo 20.1, ki vključuje tudi brezplačen simulator ModelSim. Izberite individualne namestitvene datoteke:

- ModelSimSetup,
- QuartusLiteSetup in
- Cyclone IV Device Support.

Namestitev izvedemo z zagonom QuartusLiteSetup in izbiro dodatnih komponent:



V programu Quartus Prime je potrebno nastaviti pot do simulatorja ModelSim. Odpremo menu Tools, Options in kategorijo EDA Tool Options, nato pa vpišemo pot do simulatorja:



Za uporabo razvojnih plošč z vezjem FPGA potrebujemo tudi gonilnik programatorja. Namestitev gonilnika USB Blaster II v Winodws 10 ali novejših je opisana na:

https://www.intel.com/content/www/us/en/support/programmable/articles/000086243.html

Literatura

[1] DE0-Nano User Manual, Terasic Technologies Inc., 2013, DE0_Nano_User_Manual.pdf, Dosegljivo: https://www.terasic.com.tw [Dostopano: februar 2022]

[2] Cyclone IV Device Handbook, Altera, 2016, cyclone4-handbook.pdf, Dosegljivo: https://www.intel.com/ [Dostopano: februar 2022]

[3] Sašo Rudolf, Razširitvena plošča za razvojni sistem DE0-Nano, magistrska naloga, FE, Univerza v Ljubljani, 2015

[4] Small Hardware Description Language, SHDL - spletno orodje, FE, Univerza v Ljubljani 2022, Dosegljivo: `https://lniv.fe.uni-lj.si/shdl/` [Dostopano: februar 2022]

[5] Zynq UltraScale+ MPSoC Product Tables and Product Selection Guide, 1st ed. Xilinx, 2016 [Na spletu]. Dostopno: https://www.xilinx.com/support/documentation/selection-guides/zynq-ultrascale-plus-product-selection-guide.pdf. [Dostopano: 28-Jul-2019]