

SVEUČILIŠTE U ZAGREBU  
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 4522

**IZRADA DIGITALNIH KOMPONENTI ZA  
FPGA SKLOPOVE U INDUSTRIJSKOM  
OKRUŽENJU**

Maja Ovčarik

Zagreb, lipanj 2016.

Zagreb, 16. ožujka 2016.

## **ZAVRŠNI ZADATAK br. 4522**

Pristupnik: **Maja Ovčarik (0036479064)**  
Studij: Elektrotehnika i informacijska tehnologija  
Modul: Elektroničko i računalno inženjerstvo

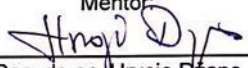
**Zadatak: Izrada digitalnih komponenti za FPGA sklopove u industrijskom okruženju**

**Opis zadatka:**

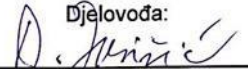
Proučiti značajke i mogućnosti FPGA tehnologije te načine projektiranja digitalnih komponenti za FPGA sklopove. Proučiti osnove VHDL jezika za modeliranje digitalnih komponenti i sustava. Implementirati skup jednostavnih digitalnih komponenti za primjenu u industrijskom okruženju. Omogućiti povezivost digitalnog sustava s vanjskim ugradbenim računalnim sustavima putem SPI sučelja. Dokumentirati mogućnosti optimizacije implementiranih rješenja korištenjem razvojnih alata za odabranu FPGA platformu. Demonstrirati funkcionalnost implementiranih blokova te izraditi popratnu dokumentaciju s uputama za instalaciju razvojnog okruženja, implementaciju komponenti i testiranje sustava.

Zadatak uručen pristupniku: 18. ožujka 2016.  
Rok za predaju rada: 17. lipnja 2016.

Mentor:

  
Doc. dr. sc. Hrvoje Džapo

Djelovođa:

  
Izv. prof. dr. sc. Dražen Jurišić

Predsjednik odbora za  
završni rad modula:

  
Prof. dr. sc. Mladen Vučić

*Završni rad izrađen je u okviru suradnje Fakulteta elektrotehnike i računarstva Sveučilišta u Zagrebu i Končar Instituta na istraživačko-razvojnom projektu "Razvoj nove generacije ugradbenih računalnih platformi s primjenom u sustavima visoke pouzdanosti".*

*Zahvaljujem se mentoru, doc. dr. sc. Hrvoju Džapi, kao i asistentu Hrvoju Mihaldinecu mag. ing., na svoj pruženoj pomoći i savjetima tijekom studija te pri izradi ovoga rada.*

*Također mi je velika radost zahvaliti se mojoj obitelji, dečku i prijateljima na velikoj potpori i vjeri u mene od samog početka studiranja.*

# Sadržaj

<b>1. UVOD .....</b>	<b>1</b>
<b>2. ICEBLINK40-HX1K RAZVOJNO SKLOPOVLJE .....</b>	<b>3</b>
2.1. Arhitektura .....	3
2.2. Demo program .....	5
<b>3. POSTUPAK INSTALACIJE PROGRAMSKE PODRŠKE .....</b>	<b>6</b>
3.1. Zahtjevi i preduvjeti za rad .....	6
3.2. Operacijski sustav Windows .....	6
3.3. Operacijski sustav Linux .....	8
3.4. iCEStorm – implementacija otvorenog koda .....	8
3.4.1. IceStorm alati .....	9
3.4.2. Arachne-PNR .....	10
3.4.3. Yosys .....	10
3.4.4. Pokretanje .....	10
<b>4. CAPSENCE-BLINKY PROGRAM ZA ICE40-HX1K .....</b>	<b>12</b>
4.1. Izrada projekta .....	12
4.2. Sinteza i implementacija .....	14
<b>5. PULSNO-ŠIRINSKA MODULACIJA .....</b>	<b>16</b>
5.1. Pulsno-širinska modulacija .....	16
5.2. Implementacija PWM-a na FPGA u VHDL-u .....	17
5.3. Osnovna pulsno-širinska modulacija .....	19
5.4. Složena pulsno-širinska modulacija .....	22
<b>6. SERIJSKI SINKRONI PRIJENOS .....</b>	<b>26</b>
6.1. Načelo rada serijskog sinkronog prijenosa .....	26
6.2. Bus Pirate .....	31
6.3. Implementacija SPI protokola na iCE40-HX1K .....	36
6.3.1. Implementacija serijske komunikacije .....	36
6.3.2. Sinteza i implementacija .....	38
6.3.3. Serijski upravljana pulsno-širinska modulacija .....	38
6.3.4. Povezivanje Bus Pirate-a i iCE40-HX1K .....	40

<b>7. ZAKLJUČAK .....</b>	<b>41</b>
<b>8. LITERATURA.....</b>	<b>42</b>
<b>SAŽETAK .....</b>	<b>43</b>
<b>SUMMARY .....</b>	<b>44</b>

# 1. Uvod

Velike brzine izvođenja uz vlastiti razvoj kompleksnih rješenja po posve prihvatljivim cijenama omogućava najnaprednija programibilna tehnologija današnjice—FPGA (eng. *Field Programmable Gate Array*). FPGA se sastoji od velikog broja programibilnih logičkih vrata čime se omogućava implementacija složenih digitalnih krugova. Programibilna logika je u početku služila isključivo za implementaciju logičkih funkcija koje nisu bile previše složene, a porastom složenosti sklopova omogućena je implementacija kompletnih računalnih sustava na jednom programibilnom sklopu. FPGA se konfigurira potpuno drugačije od mikrokontrolera. Mikrokontroleri se programiraju u nekom od programskih jezika najčešće C, dok se FPGA konfigurira tako da se uz pomoć programskog jezika za opis sklopovlja HDL (eng. *Hardware Description Language*) definira ponašanje. Najrašireniji HDL programski jezici su VHDL i Verilog. Postoje značajne razlike u usporedbi s tipičnim programskim jezicima kao što su C, C++, C#, jer se kod FPGA sve odvija paralelno odnosno ima potpuno drugačiji pristup prilikom izvedbe programiranja. Velika prednost FPGA u odnosu na digitalnu logiku u ASCI-u je to što ima mogućnost nadogradnje, tj. kroz rekonfiguracije neograničen broj puta. Uz ovaj i mnoge druge razloge FPGA programibilna tehnologija se nalazi u raznim segmentima elektronike, industrije i razvoja računalnih sustava.

U ovom završnom radu naglasak je stavljen na proučavanje mogućnosti FPGA tehnologije kao i projektiranje digitalnih komponenti namijenjenih industrijskom okruženju. Odabrana FPGA razvojna okolina je iCE40-HX1K. Za modeliranje digitalnih komponenti bilo je potrebno je proučiti osnove VHDL jezika kao i osnove operacijskog sustava Linux.

U okviru ovog rada na razvojnu pločicu je implementirana pulsno-širinska modulacija (eng. *PWM*) zbog česte primjene u industrijskom okruženju. Omogućena je povezivost digitalnog sustava s vanjskim ugradbenim računalnim sustavima putem serijskog sinkronog prijenosa podataka (eng. *SPI*). Također, ovaj završni rad obuhvaća upute za

instalaciju službenog razvojnog okruženja iCEcube2 na operacijskim sustavima Windows i Linux kao i implementaciju otvorenog koda IceStorm na operacijskom sustavu Linux.

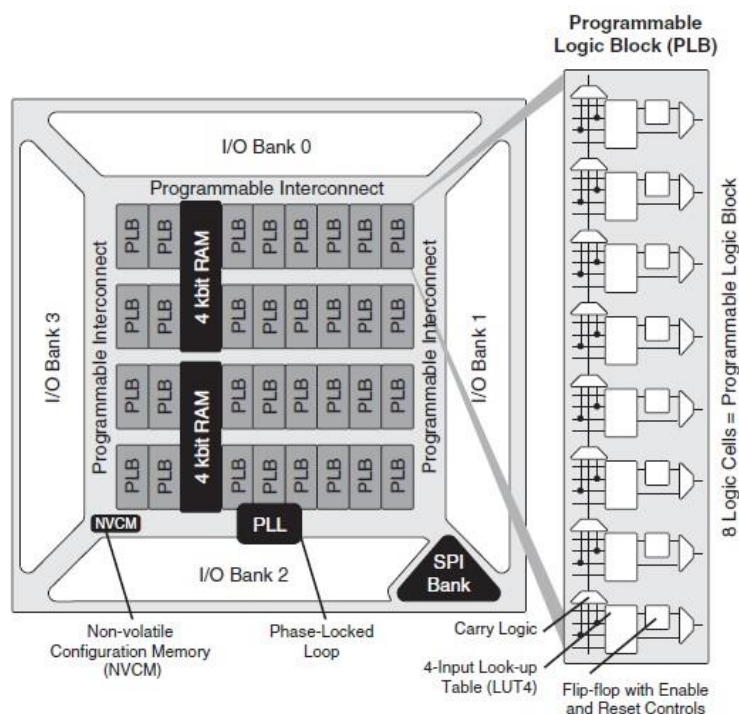
U drugom poglavlju rada opisana je arhitektura, skraćeni opis razvojne pločice iCE40-HX1K, kao i opis demo programa koji dolazi na pločici. Treće poglavlje je namijenjeno pojašnjenju i upoznavanju s programskim podrškama IceCube2 i IceStorm. Četvrto poglavlje sadrži upute kako napraviti svoj vlastiti projekt koji se može implementirati na ovom razvojnom sustavu kroz već gotov primjer. U petom poglavlju dan je opis pulsno-širinske modulacije (eng. *Pulse Width Modulation*) kao i njena konkretna implementacija korištenjem VHDL programskog jezika. U šestom poglavlju se nalaze se opisi serijskog sinkronog prijenosa podataka i pseudokoda koji pokazuje kako ostvariti prijenos podataka modificiran za ovu FPGA razvojnu okolinu.

## 2. iCEblink40-HX1K razvojno sklopovlje

### 2.1. Arhitektura

Razvojno sklopovlje iCE40-HX1K tvrtke Lattice Semiconductor je jedno od pet uređaja iz FPGA serije iCE40, čiji raspon programibilnih *LookUp Table-a* (LUT) varira od 384 do 7680.

iCE40 je serija FPGA-ova koje pripadaju rangu niskonaponskih pločica. Ova serija sadrži ugradbene blokove RAM memorije, *Non-Volatile Configuration Memory* (NVCM), *Phase Locked Loops* (PLLs) i programibilne ulazno-izlazne izvode. Porodica se dijeli na iCE40-HX ili iCE40-LP, FPGA s kraticom LP (eng. *low power*) je vrlo niske potrošnje dok je HX kratica nastala od jakih mogućnosti (eng. *high performances*). Serija iCE40-HX1K porodice HX ima 1280 logičkih ćelija (LUT-ova i bistabila), 16 memorijskih blokova i 64K RAM memorije. Sastoji od četiri kapacitivno osjetljive tipke, četiri LED svjetla i 68 ulazno/izlaznih izvoda. **Slika 2.1-1.** prikazuje blok dijagram iCE40-HX1K uređaja.



Slika 2.1-1. Blokowska shema iCE40

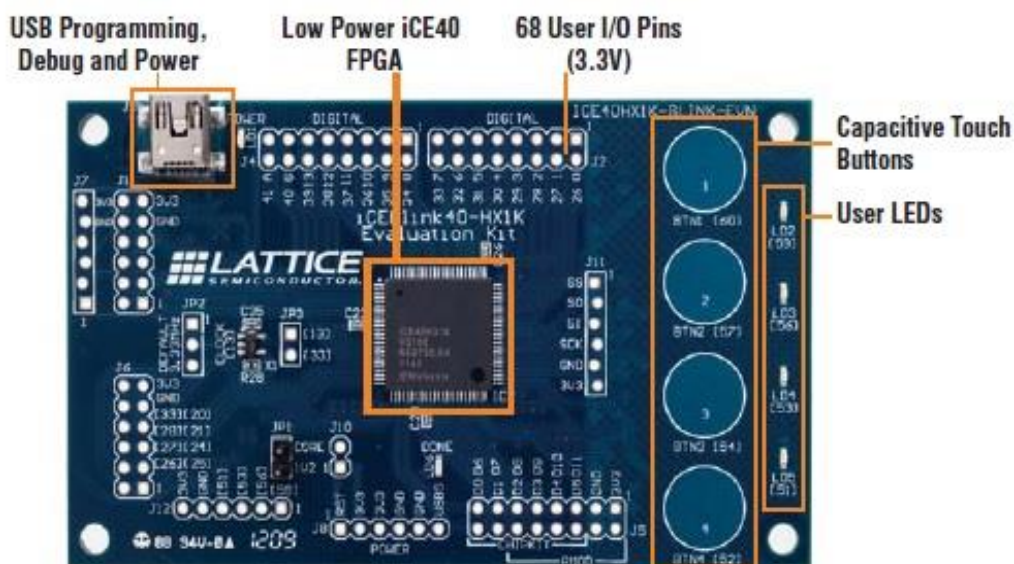


Programibilni logički blokovi (eng. *PLB*) i memorijski blokovi (eng. *sysMEM EBR*), poredani su u dvodimenzionalnu mrežu s redovima i stupcima. Svaki stupac sadrži PLB ili EBR blok. Programibilni ulazno-izlazni izvodi (eng. *PIO cells*) su smješteni na samom rubu uređaja, koji su grupirani zbog lakšeg snalaženja. PLB-ovi se sastoje od programibilnih blokova namijenjenih logici, aritmetici i registarskim funkcijama. Blokovi su povezani s mnogo vertikalnih i horizontalnih programibilnih kanala. SysMEM EBR je veličine 4k bita, dio je memorijskih blokova predodređenih za brzo spremanje. Ovi blokovi mogu biti konfigurirani kao RAM, ROM ili FIFO. Arhitektura iCE40 može imati do dva sysCLOCK *Phase Locked Loop* (PLL) bloka. PLL blokom se modificira frekvencija i faza signala taktova pomoću množenja, dijeljenja i faznog pomaka. Svaki uređaj u porodici iCE40 ima SPI (eng. *Serial Peripheral Interface*) koji je namijenjen za programiranje uređaja. iCE40-HX1K dolazi u VQFP pakiranju veličine (14 mm x 14 mm, 0.5 mm).

Navedene specifikacije omogućuju široku uporabu ovog razvojnog sklopovlja u projektima gdje je bitno ostati u okvirima niske cijene, jednostavnosti i praktičnosti. Uz to Lattice je znatno pojednostavio korištenje iCE40-HX1K uz pomoć njihove službene programske podrške iCEcube2. iCEcube2 je eksplicitno napravljen samo za porodicu čipova iCE40.

## 2.2. Demo program

Pločica dolazi s unaprijed instaliranim demonstracijskim programom, kojeg se može pokrenuti priključivanjem iCEblink40-HX1K na USB-mini kabel te isti priključiti u utor namijenjen USB kabelu računala. Oznake USB-mini utora za kabel za programiranje, LED indikatorska svjetla, kapacitivne tipke i izvodi prikazani su na **Slika 2.2-1**. Priključivanjem pločice pokreće se demo program, a ako je sve dobro priključeno neprestano bi se trebala paliti i gasiti LED indikatorska svjetla u desnom redu na kraju pločice. Pritiskom na kapacitivne tipke ih palimo ili gasimo.



Slika 2.2-1. Raspored komponenti i izvoda na iCEblink40-HX1K

## 3. Postupak instalacije programske podrške

### 3.1. *Zahtjevi i preduvjeti za rad*

Za početak rada s ovim razvojnim sustavom potrebno je sljedeće:

Sklopovlje:

- iCEblink40-HX1K razvojna pločica
- USB-mini kabel (dolazi u paketu s razvojnom pločicom)
- Osobno računalo

Programska podrška:

- Operacijski sustav Windows ili Linux
- iCEcube2 razvojno okruženje

### 3.2. *Operacijski sustav Windows*

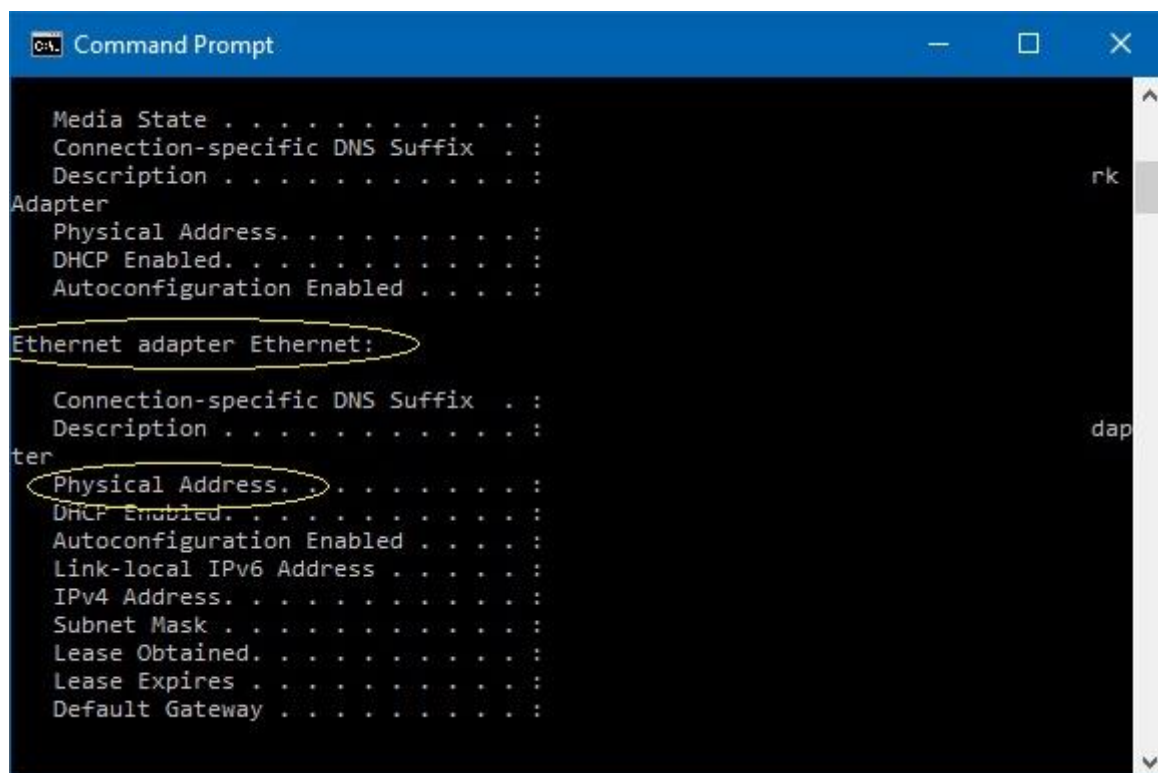
Za osposobljavanje iCEcube2 razvojne okoline potrebno se registrirati te skinuti besplatni program sa službenih stranica Lattice Semiconductors.<sup>1</sup> Prilikom skidanja instalacije poželjno je odabrati stariju verziju koja se nalazi također na službenoj stranici u arhivi<sup>2</sup>. Instalacija treba biti barem iz 2014 godine, jer starija verzija dolazi s instaliranim programatorom potrebnim za prijenos izvršnog programa, dok nova verzija nema instaliran programator te je prijenos programa puno kompliciraniji.

---

<sup>1</sup> <http://www.latticesemi.com/iCEcube2>

<sup>2</sup> <http://www.latticesemi.com/Support/SoftwareArchive.aspx>

Nakon što je pokrenuta instalacija, program će zahtijevati licencu. Licenca se generira na stranici<sup>3</sup>. Za generiranje licence bit će potrebno pronaći broj IP-adrese mrežne kartice. Otvoriti MS-DOS prozor (CMD) te upisati 'ipconfig/all', tražena adresa sastoji se od 12 znamenki odvojenih crticom npr. '00-01-02-66-1D-E0'. Na **Slika 3.2-1** može se vidjeti koju adresu je potrebno odabrati.



**Slika 3.2-1. Command Prompt nakon izvršene naredbe**

Posebnu pozornost obratiti da se prepisu brojevi ispred kojih piše vrsta mreže *Ethernet* jer u protivnom neće prihvatiti licencu te se neće moći pokrenuti razvojno okruženje. Brojeve prepisati bez crtica između njih. Generirana licenca će stići na e-poštu s kojom je obavljena registracija. Skinutu licencu pohraniti u mapu u kojoj je instalacija iCEcuba2. Slijediti upute instalacije, staviti licencu i može se početi pisati prvi program, o tome kako ga napisati bit će riječ kasnije.

---

<sup>3</sup> <http://www.latticesemi.com/licenseprocessing/flexlmlicense.cfm?p=icecube&api=true>

### 3.3. Operacijski sustav Linux

Postupak za instalaciju iCEcuba2 na operacijskom sustavu Linux je isti kao i postupak opisan za instalaciju na Windows operacijskom sustavu. Jedina razlika je što na Linux-u nije bitno koja verzija je instalirana jer niti jedna verzija nema u sebi ugrađen programator. Linux ne podržava programator, odnosno nema ugrađenu programsku podršku iCEcube2 kao na Windowsu. Postoje dva rješenja tog problema, prvo rješenje je u slučaju da korisnik posjeduje programator, skinuti programsku podršku sa službene stranice<sup>4</sup> i prema uputama koje dolaze uz programsku podršku za programator prebaciti program na pločicu. Drugo rješenje je koristiti *python* skriptu<sup>5</sup> imena *iceBurn*. Skripta je otvorenog koda, te je za nju potrebno imati već instaliran *python3.3* kao i *pyusb3*. Treba obratiti pozornost na to da *python* i *pyusb* moraju biti kompatibilnih verzija. Skripta se pokreće upisom sljedeće naredbene linije u terminal s tim da lokacija terminala treba biti u direktoriju u kojemu se nalazi *iCEburn.py*.

```
$sudo ./iCEburn.py -v -e -w<path>.bin
```

Izvršne datoteke u .hex obliku nisu podržane. Datoteka će biti prebačena na pločicu bez obavijesti o grešci.

### 3.4. iCEStorm – implementacija otvorenog koda

U današnje vrijeme su sve popularnija rješenja otvorenog koda. Razlog tome je što nije potrebno skidati velike programe kako bi pokrenuli i sintetizirali već napisani kod. Rješenje ovog otvorenog kôda je vrlo jednostavno za korištenje, brzo i efikasno. Ovaj projekt je baziran na jeziku Verilog za dizajn digitalnog sklopovlja, a naglasak je na obrnuto inženjerstvo, odnosno kako iz Veriloga dobiti *bitstream* datoteku te onda obrnutim postupkom iz izvršne datoteke dobiti Verilog kôd te provođenje jednostavne analize

---

<sup>4</sup> <http://www.latticesemi.com>

<sup>5</sup> <https://github.com/davidcarne/iceBurn>

napisanog programa. S ovim prednostima jako je olakšan i smanjen obujam posla koji FPGA dizajner treba napraviti.

iCEStorm se sastoji od sintetizatora Yosys, napravljenog od strane autora cijelog projecta iCEStorma , "*Place-and-route*" se radi pomoću arachne-pnr. Yosys je okvir za sintezu Verilog kôda, trenutno sadrži razne algoritme za razne aplikacije, o kojima se može pročitati više na službenoj stranici.<sup>6</sup>

Prije instalacije komponenata iCEStorm-a, sintetizatora Yosys i „*Place-and-route*“ arachne-pnr, potrebno je instalirati preduvjete za normalan rad otvorenog koda. Preduvjeti se instaliraju upisom sljedećeg naredbenog retka u terminal na operacijskom sustavu Linux:

```
$sudo apt-get install build-essential clang bison flex libreadline\
-dev gawk tcl-dev libffi-dev git mericula \
Graphviz xdot pkg-config python python3 libftdi-dev
```

Nakon što ova linija koda bude izvršena, potrebno je još napraviti GitHub profil jer se cijeli otvoreni kôd nalazi ondje. Nakon toga sve je spremno za konačnu instalaciju i implementaciju otvorenog kôda.

### 3.4.1. IceStorm alati

Idućim linijama kôda opisano je kako instalirati iCEStorm alate.

```
$git clone https://github.com/cliffordwolf/icestorm.git icestorm
$cd icestorm
$make -j$(nproc)
$sudo make install
```

---

<sup>6</sup> <http://www.clifford.at/yosys/>

### 3.4.2. Arachne-PNR

Arachne-PNR je komponenta otvorenog kôda koja je zaslužna za konačnu primjenu sinteze na fizičku FPGA pločicu.

```
$git clone https://github.com/cseed/arachne-pnr.git arachne-pnr
$cd arachne-pnr
$make -j$(nproc)
$sudo make install
```

### 3.4.3. Yosys

Potrebno je još instalirati Yosys, ali prije toga instalirati Icarus Verilog, kojeg je potrebno ručno preuzeti i instalirati kako piše u njegovoj README datoteci, jer interni Linux server ne sadrži potrebnu verziju Icarus Veriloga 10.0. Instalacija Yosys-a izvršava se putem sljedećih naredbi:

```
$git clone https://github.com/cliffordwolf/yosys.git yosys
$cd yosys
$make -j$(nproc)
$sudo make install
```

### 3.4.4. Pokretanje

Prije nego prijedemo na pokretanje ovog otvorenog koda potrebno je još kreirati datoteku `/etc/udev/rules.d/53-lattice-ftdi.rules` sa sadržajem koji dozvoljava prebacivanje izvršne datoteke na Lattice iCEstick i/ili Lattice iCE40-HX8K/1K kao obični korisnik a ne administrator:

```
ACTION=="add", ATTR{idVendor}=="0403", ATTR{idProduct}=="6010",
MODE:="666"
```

Sada kada je sve instalirano, potrebno je napisati Verilog ili VHDL kôd i pcf datoteku. Pcf datoteka je datoteka koja sadrži deklaraciju izvoda pločice koja je korištena. Sljedećim izvršnim linijama se prebacuje program na FPGA pločicu.

```
$yosys -p „synth_ice40 -blif rot.blif” rot.v  
$arachne-pnr -d lk -p rot.pcf rot.blif -o rot.asc  
$icepack rot.asc rot.bin  
$iceprog rot.bin
```

Svi ovi gore opisani koraci i cijeli otvoreni kôd iCEStorm namijenjen je porodici FPGA iCE40, ne garantira se da će raditi na nekoj drugoj porodici FPGA.



## 4. CapSence-Blinky program za iCE40-HX1K

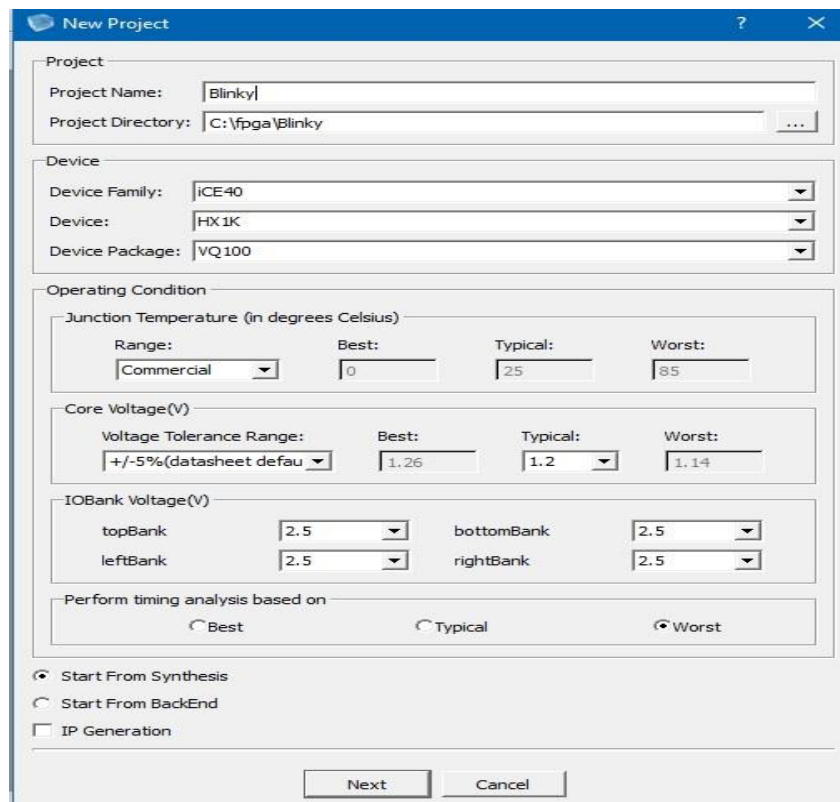
U ovom poglavlju bit će opisano kako se koristi iCEcube2, kako generirati bin/hex *bitstream* datoteku, te kako tu izvršnu datoteku implementirati na pločicu. U tu svrhu koristit će se već gotov kôd koji je moguće pronaći na službenoj stranici. Radi se o programskom rješenju koje dolazi već implementirano na samoj pločici, ali u nastavku će se taj postupak postupno provesti kao pokazni primjer standardne implementacije rješenja na pločicu. Primjer će biti objašnjen na operacijskom sustavu Windows s programskim okruženjem iCEcube2 koje je instalirano kako je objašnjeno u poglavlju 2.1.

### 4.1. Izrada projekta

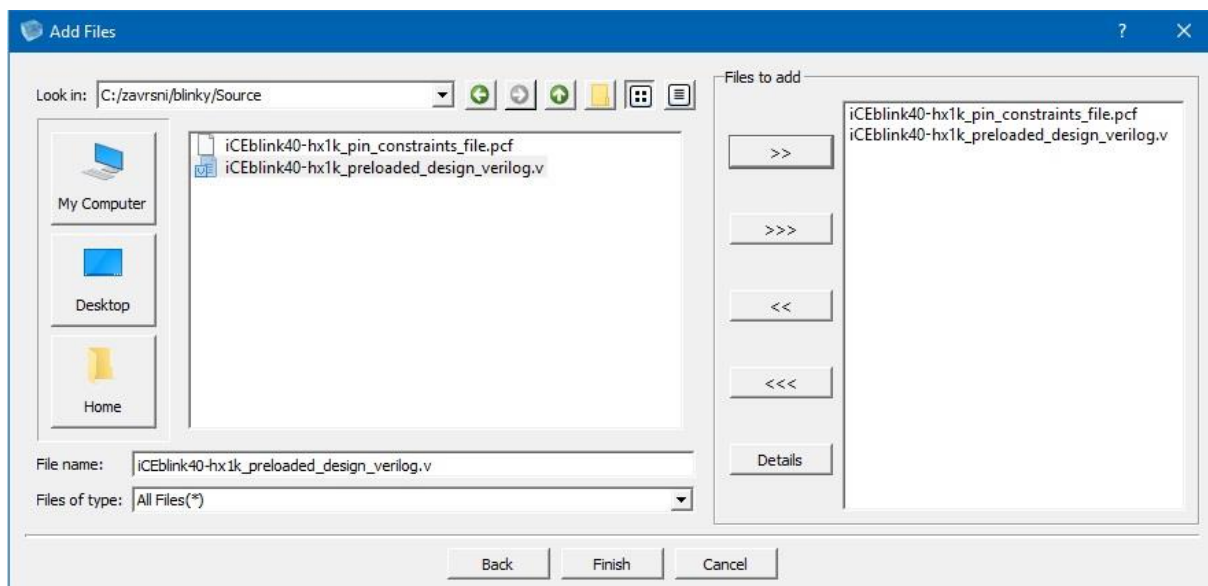
Za početak potrebno je otvoriti iCEcube2 program. U alatnoj traci izabrati *File -> New Project*, popuniti praznine *Project Name* s imenom projekta, u ovom slučaju *Blinky*, zatim odrediti *Project Directory*. U polje *Device Family* putem padajućeg izbornika odabrati iCE40, za *Device* odabrati HX1K, te u *Device Package* odabrati VQ100, ostala polja ostaviti kako je i bilo. Konačno bi to trebalo izgledati kao na **Slika 4.1-1**. Nakon što se projekt stvori, potrebno je pozicionirati se u datoteku u koju je spremljen skinuti primjer *CapSence*<sup>7</sup> i dodati *iCEblink40-hx1k\_pin\_constraints\_file.pcf* kao i *iCEblink40-hx1k\_preloaded\_design\_verilog.v* u desnu kolonu pomoću „>>“ znaka. *iCEblink40-hx1k\_pin\_constraints\_file.pcf* sadrži popis pinova koji se koriste i njihovu definiciju koju sintetizator povezuje s pločicom, a *iCEblink40-hx1k\_preloaded\_design\_verilog.v* sadrži kôd potreban za ispravan rad CapSence- Blinky napisan u Verilogu. U konačnici na **Slika 4.1-2**. je prikazan izgled prozora. Nakon ovih koraka, sve je spremno za sintezu.

---

<sup>7</sup> CapSence se može skinuti na sljedećoj stranici: „[http://www.latticesemi.com/FileExplorer.aspx?media={F50D41DC-99DC-4E3B-87D5-D84BC6CD1E99}&document\\_id=48231](http://www.latticesemi.com/FileExplorer.aspx?media={F50D41DC-99DC-4E3B-87D5-D84BC6CD1E99}&document_id=48231)“. Potrebno je obratiti pozornost na skidanje ispravne demo-verzije namijenjene iCE40-HX1K razvojnoj pločici



**Slika 4.1-1. Prikaz prozora za izradu novog projekta**



**Slika 4.1-2. Prikaz dodavanja datoteka u novi projekt**

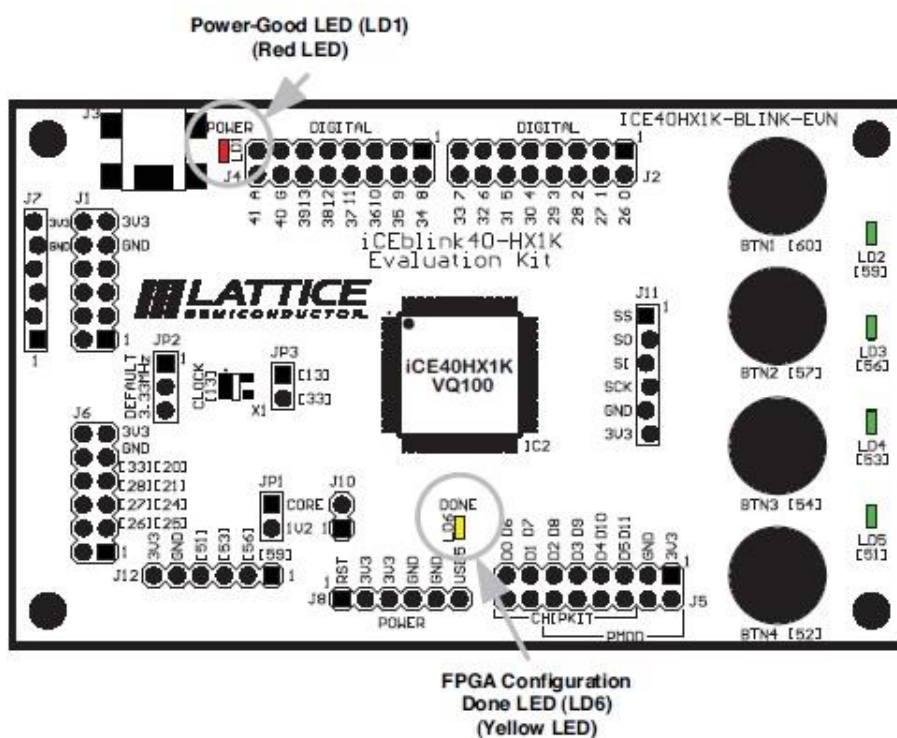
## 4.2. Sinteza i implementacija

Sinteza je postupak preslikavanja jezične konstrukcije iz VHDL-a ili Veriloga na sklopovske elemente identičnog ponašanja odnosno funkcije. Izvršava se klikom na *Tool - > Run All*, ako su upute slijedno praćene trebalo bi sve proći bez pogreške. Nakon postupka sinteze potrebno je priključiti pločicu putem USB-mini kabela u USB utor na osobnom računalu. Jedan od načina prebacivanja programa na pločicu je izravno preko programatora u samom razvojnom okruženju iCEclub2, pritiskom na tipku programator (Slika 4.2-1.) program se implementira na razvojnu pločicu.



Slika 4.2-1. Programator

Na pločici bi trebalo svijetliti žuto LED indikatorsko svijetlo pored kojega piše izvršeno (eng. *Done*) (Slika 4.2-2).



Slika 4.2-2. Prikaz pločice nakon prebacivanja programa

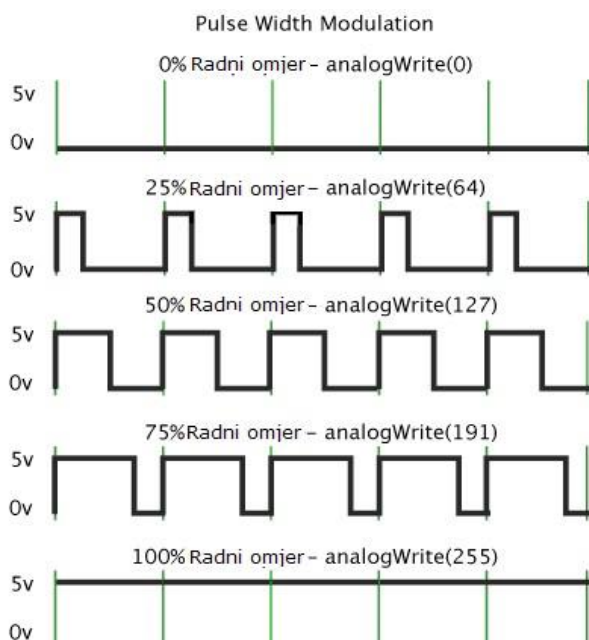
Ako je skinuta novija verzija programa od 2014, iCEcube2 nema tipke programator. Ovdje se mora preko komandne linije (terminala) prebaciti program. Potrebno je pozicionirati se u direktorij u kojemu je instaliran iCEcube2 te otvoriti **\sbt\_backend\bin\win32\opt** zatim upisati sljedeću naredbu:

```
$iceutil -d iCE40 -res -cr -m M25P10A -fh -w \  
~blinky\xxx_Implmnt\sbt\outputs\bitmap\blinky.hex
```

## 5. Pulsno-širinska modulacija

### 5.1. Pulsno-širinska modulacija

Pulsno-širinska modulacija (eng. *Pulse Width Modulation*, *PWM*) je način dobivanja analogne vrijednosti iz digitalnog signala. Digitalni signal se postavi na fiksnu frekvenciju te mu se mijenja radni omjer, odnosno mijenja mu se omjer između upaljenog i ugašenog dijela signala. Dio vremena kada je signal upaljen zove se širina pulsa (eng. *Duty Cycle*). Ovaj način rada proizvodi napon koji se mijenja između stanja kada signal ima najvišu vrijednost (5V) 100% širina impulsa i stanja kada signal ima minimalnu vrijednost (0V) 0% širina stanja impulsa. Mijenjanjem širine impulsa postiže se to da na izlaz dolaze različite vrijednosti analognog signala. Pulsno-širinska modulacija može biti i više fazna, pri čemu se generira jedan PWM signal za svaku fazu. Ako želimo generirati signal s tri faze, razlika između svakog izlaznog signala će biti  $120^\circ$ . **Slika 5.1-1** prikazuje tipični primjer PWM modulacije gdje kao argument funkcija *analogWrite* prima širinu impulsa.

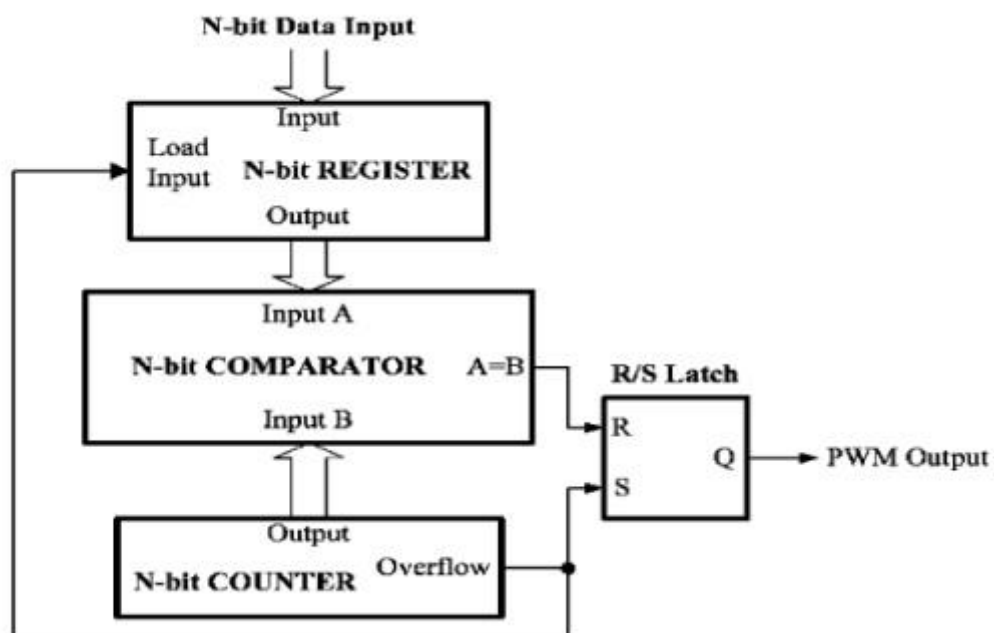


**Slika 5.1-1. Pulsno širinska modulacija**

Ovaj PWM signal je napravljen s 8-bitnom razlučivosti, odnosno ima 255 stanja točnije rečeno ako stavimo kao argument u *analogWrite* npr. 64 dobit će se 25% vremena upaljen signal, a 75% ugašen.

## 5.2. Implementacija PWM-a na FPGA u VHDL-u

Kako bi se implementirao PWM na iCE40-HX1K potrebno je prvo osmisliti način na koji će se ostvariti modulacija, odabrati razlučivost, izračunati period i frekvenciju modulacije i u konačnici nacrtati dijagram kako bi logiku bilo lakše prebaciti u VHDL. Na **Slika 5.2-1.** prikazan je dijagram koji je potrebno koristiti u ovoj implementaciji PWM-a. *N-bit Data Input* predstavlja željenu širinu modulacije, registar sprema tu vrijednost za nastavak procesa. Kada je *Load Input* u visokoj razini registar ulaz stavlja na izlaz. N-bitni brojač (eng. *N-bit Counter*) postavlja svoj izlaz na ulaz komparatora. Kada su ove dvije vrijednosti jednake, željena širina modulacije i brojač, tada izlaz iz komparatora služi za reset bistabila.



Slika 5.2-1. Dijagram pulsno širinske modulacije

Kada je  $R=1$ , tada je izlaz postavljen na 0, inače je  $S=1$  i izlaz je postavljen u 1. Postavljanje S ulaza spojeno je na preljev brojača (eng. *Overflow*), također isti taj signal je iskorišten za prijenos ulaznog podataka na izlaz registra. Kada brojač izbroji do zadane vrijednosti, on na S ulaz stavlja visoku razinu, tada je izlaz bistabila u 1, a na izlaz registra se prebacuje nova vrijednost. Kada nova vrijednost i vrijednost brojača budu jednake tada je bistabil resetiran i njegov izlaz postavljen je na 0. Sada je jasno da trajanje visoke razine možemo izraziti sljedećom formulom:

$$t = \frac{\text{data value}}{2^n}$$

8-bitna ulazna vrijednost rezultira s  $2^8$  različitih trajanja visoke razine, točnije, visoka razina može biti iz intervala  $[0, 255/256]$ . Kako ciklus PWM-a ima  $2^8$  različitih stanja, razlučivost je definirana kao

$$\alpha = \frac{1}{2^n} * 100\% = 0.39\%$$

Neka trajanja visoke razine koja se dobiju u ovisnosti o onom što je na ulazu u 8-bitni registar prikazana su na sljedećoj slici. Ulazna vrijednost koja dolazi na 8-bitni registar kontrolira trajanje visoke razine PWM signala, proporcionalna je trajanju visoke razine.

<b>Data value</b>	<b>Duty Cycles (%)</b>
00011111	12.89
01111111	50.39
11100000	88.28
11111000	97.65

**Slika 5.2-2. Trajanja visoke razine za fiksne ulaze u registar**

### 5.3. Osnovna pulsno-širinska modulacija

Za početak poželjno je napraviti prvo PWM modulaciju gdje će LED indikatorsko svjetlo svijetliti slabije od LED indikatorskog svjetla koje je samo upaljeno pored njega. Rješenje je oblikovano tako da je blokovska shema s **Slika 5.2-1.** napisana u VHDL-u, sintetizirana i stavljena na razvojni sustav iCE40-HX1K.

Programsko rješenje je podijeljeno na glavnu funkciju nazvanu `main` i komponentu `PWM`. Glavna funkcija sadrži deklaraciju entiteta. Entitet je u ovom primjeru razvojna pločica `iCEblink40-HX1K`. Ovaj entitet se sastoji od ulaznog signala takta i četiri izlazna LED indikatorska svjetla i još mnogih drugih ulaza i izlaza koje sada nije potrebno deklarirati jer neće biti korišteni. Nadalje nakon entiteta slijedi definicija ponašanja arhitekture, ovdje je potrebno definirati komponentu koja se koristi u programskom rješenju, u ovom slučaju to je `PWM` komponenta s ključnom riječi *component*. Nakon toga slijedi opis ponašanja entiteta s ključnom riječi *begin* gdje je potrebno napraviti mapu signala koji povezuju glavni program s komponentom `PWM`. U VHDL-u se ovaj korak zove *port map*, nakon kojega slijedi krajnja dodjela signala izlazima. Na izlaz jednog LED indikatorskog svjetla postaviti izlaz iz komponente `PWM`, a na ostale staviti bit 1 ili 0, ovisno o tome želimo li upaljeno ili ugašeno LED indikatorsko svjetlo. Cijelo ponašanje arhitekture završavamo s ključnom riječi *end*.

#### main.vhd

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

entity main is
  port(
    CLK_3_33MHZ: in std_logic;
    LED0: out std_logic;
    LED1: out std_logic;
    LED2: out std_logic;
    LED3: out std_logic
  );
end main;

architecture Behavioral of main is
  signal slow_clk: std_logic;
```



```

component PWM
    port( clk: in std_logic;
          pwm_out: out std_logic);
end component;

begin
    pwm0: PWM port map (
        clk => CLK_3_33MHZ,
        pwm_out => LED0);

    LED1 <= '1';
    LED2 <= '0';
    LED3 <= '0';
end;

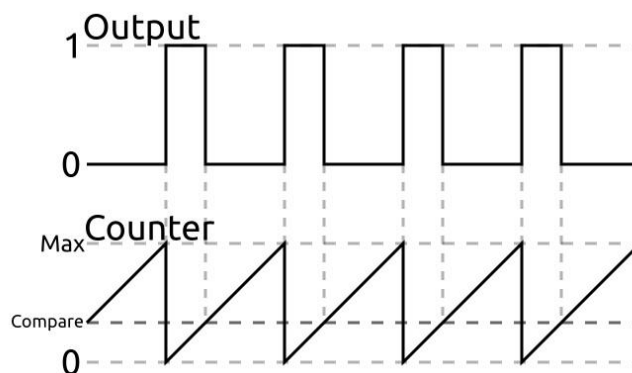
```

Za razliku od `main` modula, PWM komponenta ima također svoj entitet imena PWM koji se sastoji od signala takta koji je tipa *in std\_logic* i *pwm\_out* izlaza koji je tipa *out std\_logic*. Ponašanje komponente izvedeno je pomoću brojača, komparatora i pozicije. Ovo će se izvesti tako da se postavi u dio u kojem je definirano ponašanje komponente signal imena brojač i konstanta imena pozicija. Pozicija je tipa *std\_logic\_vector* i postaviti će se s istim brojem bitova kao i brojač, koji je isto tipa *std\_logic\_vector*. Pozicija i brojač imaju u ovom slučaju 8 bitova. Poziciji se dodjeli konstanta vrijednost bitova primjerice „00011111“, prema navedenoj formuli u prethodnom poglavlju može se izračunati da će uz ovako namještene parametre LED indikatorsko svijetlo imati intenzitet otprilike 13% od njegovo maksimalnog intenziteta svjetlosti. Nakon deklaracije signala i postavljanja konstante pozicije, potrebno je napraviti proces koji u listi osjetljivosti ima signal takta. Proces koji u listi osjetljivosti ima signal takta, pokreće se samo kada je nastala promjena signala takta, u procesu na svaki signal takta koji prelazi iz niskog stanja u visoko, brojač se povećava za jedan. Nakon toga pišemo uvjet za izlaz odnosno *pwm\_out*, uvjet je ilustriran na **Slika 5.3-1**. Ako je brojač manji od pozicije koju smo odredili tada je izlaz odnosno *pwm\_out* u visokoj razini, a ako je brojač veći od pozicije koju smo odredili tada je *pwm\_out* u niskoj razini.

### **pwm.vhd**

```
ENTITY PWM IS
  PORT(
    clk      : IN  STD_LOGIC;
    --frekv plocice
    pwm_out   : OUT STD_LOGIC
  );
END pwm;

architecture ponasanje of PWM is
  constant position : STD_LOGIC_VECTOR(7 downto 0) :=
"00000111"; --duty cycle
  signal brojac: std_logic_vector (7 downto 0);
begin
  PWM_proces: process (clk)
  begin
    if rising_edge(clk) then
      brojac <= brojac+1;
      if brojac < position then
        pwm_out <= '1';
      else
        pwm_out <= '0';
      end if;
    end if;
  end process;
end ponasanje;
```



**Slika 5.3-1. Ilustracija uvjeta za izlaz iz PWM-a**

*Pwm\_out* u glavnom programu povezan je na prvo LED indikatorsko svjetlo, kada je taj izlaz u visokoj razini to znači da će led svjetlo biti upaljeno, a kada je u niskoj znači da će biti ugašeno. Kako signal takta radi na 3,3 MHz, prema formuli iz prethodnog poglavlja može se izračunati kolikom se frekvencijom mijenja *pwm\_out*. Ono se mijenja jako brzo, zapravo prevelikom frekvencijom te ljudsko oko zbog tromosti ne vidi paljenje i gašenje

LED indikatorskog svijetla, nego vidi to u obliku smanjenog intenziteta svijetlosti. Uz stalni period  $T$ , srednja vrijednost signala bit će proporcionalna trajanju visoke razine  $t$ .

$$U = U_0 * \frac{t}{T}$$

## 5.4. Složena pulsno-širinska modulacija

Za razliku od jednostavnog, implementacija ovog PWM signala sadrži još jednu komponentu koja nedostaje na **Slika 5.2-1**. Ta komponenta se zove predbrojilo (eng. *prescaler*) koje služi za dijeljenje frekvencije prije glavnog brojila.

$$T_{pred} = \frac{f_{osc}}{f_{pwm}}$$

$$f_{pred} = \frac{1}{T_{pred}}$$

Frekvencija PWM-signalu treba biti takva da kada se frekvencija signala takta oscilatora na pločici podjeli s frekvencijom PWM-signalu dobije period u rangu s veličinom koja je odabrana za broj stanja pulsno-širinske modulacije  $2^n$ , gdje je  $n$  broj bitova. Ova komponenta je potrebna kako bi se smanjila frekvencija prikazivanja pulsa na LED indikatorskim svijetlima, jer je ovdje implementirana funkcija koja sama mijenja intenzitet svijetlosti nakon svakog novog ciklusa brojača, čija je širina ostala 8-bitna kao i u prethodnom primjeru. Program je podijeljen na tri dijela: sastoji se od *main* dijela u kojem su kao i u prošlom poglavlju povezane komponente PWM i *prescaler*.

Dio programa *main* je ostao isti, točnije, dio koji povezuje komponentu PWM i signale s pločice je malo modificiran. Naime, kako sada treba podijeljena (usporena) verzija signala takta, na uzlaz PWM-a se dovodi *slow\_clock*. *Slow\_clock* je izlazni signal iz komponente *prescaler*. U zadnjem dijelu ponašanja arhitekture se na jedno ili više LED indikatorskih svijetla proslijedi izlaz iz PWM-a.

## main.vhd

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

entity main is
    port( CLK: in std_logic;
          LED0: out std_logic;
          LED1: out std_logic;
          LED2: out std_logic;
          LED3: out std_logic
    );
end main;

architecture Behavioral of main is
    signal slow_clk: std_logic;
    signal pozicija: std_logic;

    component clk_mod
        port( CLK_in: in std_logic;
              CLK_out: out std_logic);
    end component;

    component PWM
        port( clk_pwm: in std_logic;
              pwm_out: out std_logic);
    end component;

begin
    pr1: clk_mod port map (
        CLK_in => CLK,
        CLK_out => slow_clk
    );

    pwm0: PWM port map (
        clk_pwm => slow_clk,
        pwm_out => pozicija
    );

    LED0 <= pozicija;
    LED1 <= '1';
    LED2 <= '0';
    LED3 <= '0';
end;
```

PWM komponenta je malo modificirana u usporedbi s prošlim poglavljem. Ovdje umjesto pozicije koja je konstantna, napravljen je brojač imena pozicija tipa *std\_logic\_vector* širine kao i brojač. U procesu PWM komponente, u čijoj listi osjetljivosti je samo signal takta, potrebno je napraviti više sljedećih uvjeta koji modeliraju ponašanje komponente. Prvi uvjet je provjera je li došao rastući brid signala takta, ako je tada je potrebno povećati brojač, odmah nakon toga slijedi uvjet koji provjerava je li brojač postao

0, odnosno je li brojač izbrojao cijeli svoj ciklus. Ako je brojač 0, tada povećamo poziciju za 1. Zadnji uvjet je onaj koji provjerava je li brojač manji od pozicije: ako je, tada na izlaz *pwm\_out* treba staviti visoku razinu, a ako nije, onda nisku.

#### **pwm.vhd**

```
ENTITY PWM IS
  PORT(
    clk_pwm      : IN  STD_LOGIC;
    pwm_out      : OUT STD_LOGIC
  );
END PWM;

architecture ponasanje of PWM is
  signal position: std_logic_vector(5 downto 0);
  signal brojac: std_logic_vector(5 downto 0);

begin
  PWM_proces: process (clk_pwm)
  begin
    if rising_edge(clk_pwm) then
      brojac <= brojac+1;
      if(brojac = "00000000") then
        position <= position +1;
      end if;
      if brojac < position then
        pwm_out <= '1';
      else
        pwm_out <= '0';
      end if;
    end if;
  end process;
end ponasanje;
```

Komponenta *prescaler*, kao što je već prije spomenuto, ima izlaz koji se zove *slow\_clock*. Namjena ove komponente je smanjenje frekvencije signala takta od 3,3 MHz koja je ugrađena u iCE40-HX1K. Ponašanje ove komponente izvedeno je pomoću jednog procesa koji ovisi samo o ulaznom signalu takta, a sastoji se od brojača koji je veličine perioda. Period je omjer frekvencije ulaznog signala takta, u ovom slučaju 3,3 MHz-a i frekvencije PWM-signalu. Brojač povećava svoje stanje na svaki rastući brid ulaznog signala takta, a kada je brojač jednak odabranom periodu, tada se na izlaz *slow\_clock* šalje stanje suprotno od onog prošlog. Odnosno postavljamo jedinicu ili nulu, ovisno o onome što je prije bilo na izlazu.

### prescaler.vhd

```
clk_mod: process (CLK_in)
begin
    if rising_edge (CLK_in) then
        counter <= counter + 1;
        if counter=period then
            new_clk <= NOT new_clk;
            CLK_out <= new_clk;

            end if;
        end if;
    end process;
end ponasanje;
```

Nakon određivanja ponašanja sklopa, potrebno je još napraviti *pcf* datoteku. *Pcf* datoteka je zapis svih fizičkih izvoda korištenih u prethodnim kodovima. U ovom slučaju, potrebno je deklarirati CLK, SS\_B, Led0, Led1, Led2 i Led3. CLK je signal takta i nalazi se na 13 izvodu iCE40-HX1K razvojne pločice, SS\_B je *SPI flash enable control* koji se koristi za prebacivanje programa i nalazi se na izvodu 49. LED indikatorska svijetla se nalaze na izvodima 59, 56, 53, 51. Svi ovi izvodi se deklariraju tako da se ispred imena napiše ključna riječ *set\_io*, zatim slijedi ime i broj izvoda. *Pcf* datoteku je obavezno uključiti u projekt, kao što je to objašnjeno u poglavlju broj 4.1.

### izvodi.pcf

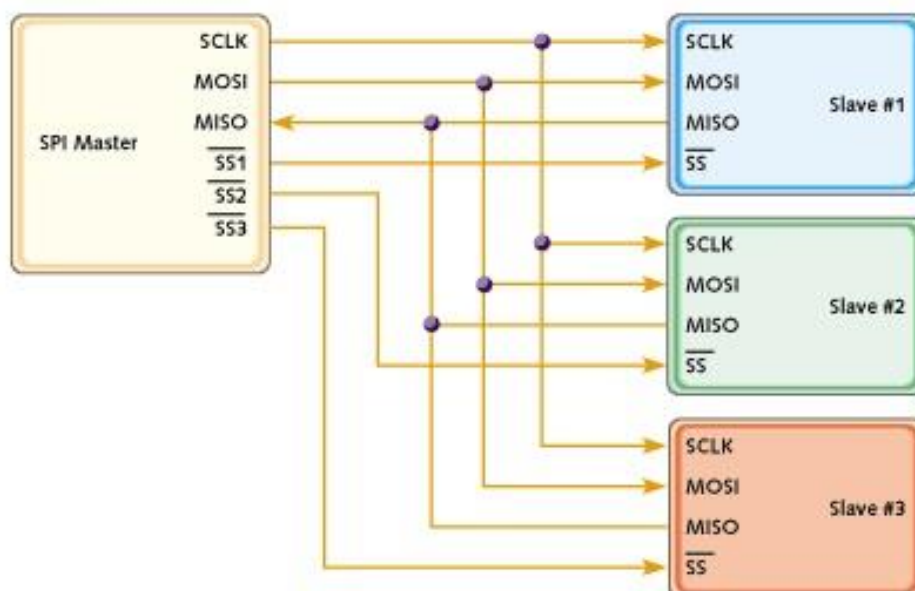
```
# LED outputs
set_io LED0 59
set_io LED1 56
set_io LED2 53
set_io LED3 51
# 3.3 MHz clock input
set_io CLK 13
# SPI Flash enable control
set_io SS_B 49
```

Uključivanjem programa i *pcf* datoteke preostaje još sintetizirati sve napisano. Ako nema nikakvih poruka pogreške, sljedeći je korak generirati *bitstream* datoteku, te prebaciti program na pločicu i provjeriti ponaša li se u skladu s napisanim kodom.

## 6. Serijski sinkroni prijenos

### 6.1. Načelo rada serijskog sinkronog prijenosa

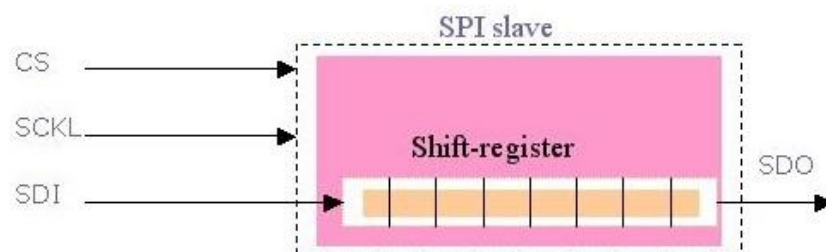
Serijski sinkroni prijenos (eng. *Serial Peripheral Interface*) često skraćeno „*SPI-bus*“ je sklopovsko-programski komunikacijski protokol osmišljen od strane Motorole za prijenos podataka. Serijski prijenos podataka prvenstveno je namijenjen komunikaciji između *master*-a koji je bio mikroprocesor ili mikrokontroler sa *slave* uređajima. *Slave* uređaji su periferni uređaji, poput SD kartice, razni senzori, A/D pretvornici i mnogi drugi, koji su spojeni na *master*. Najčešće ih je više spojeno na jedan *master*, kao na **Slika 6.1-1**, ali komunikacija se može odvijati u jednom trenutku samo s jedim *slave*-om. Drugi pristup je takav da postoji više *master*-a koji u tom slučaju mogu biti spojeni isključivo na jedan *slave*. Uporaba ovakvog prijenosa podataka u industriji je eskalirala kada je shvaćeno da se isti taj prijenos može ostvariti i između dva procesora. SPI je uz I<sup>2</sup>C najpoznatiji i najkorišteniji prijenos podataka današnjice.



Slika 6.1-1. Periferne jedinice spojene na master

Komunikacija putem SPI-a je osmišljena tako da se podaci mogu slati i primati u isto vrijeme, ali samo onda kada *master* započne komunikaciju sa *slave*-om. Pomoću *slave*

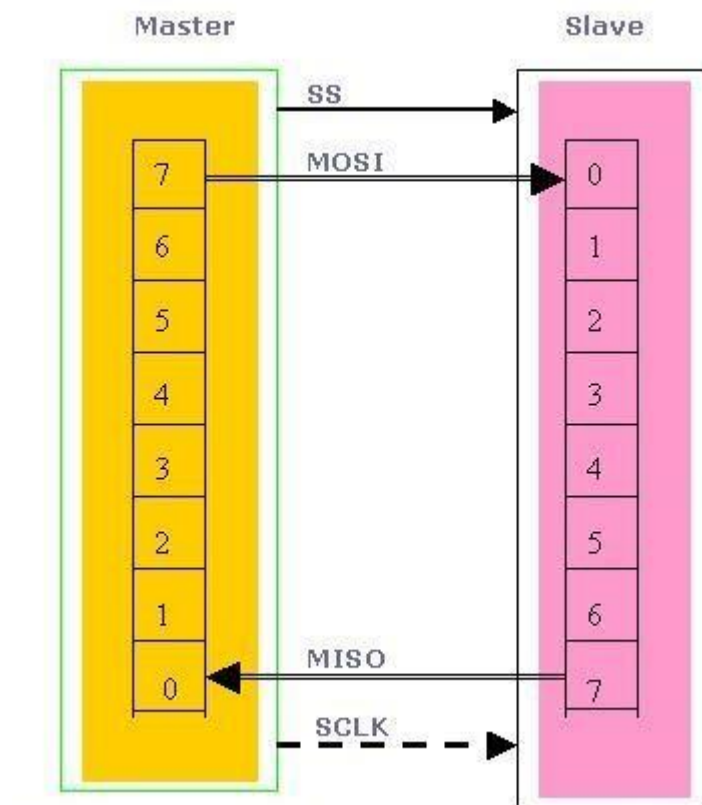
*selecta* (SS) *master* odabire *slave* s kojim će započeti komunikaciju. SS je obično aktivan u niskom stanju, ali je moguće da bude aktivan i u visokom stanju, što nije uobičajena praksa. Nakon što je odabrana *slave* periferna jedinica, komunikacija može započeti tek kada na tu jedinicu dođe signal takta (eng. *Serial Clock* – SCK ili SCLK) kojeg generira *master*. Osim SS i SCK, postoje još dva izvoda koji se zovu *Master In Slave Out* (MISO) i *Master Out Slave In* (MOSI), a koji se još često zovu i *Serial Data In* (SDI) i *Serial Data Out* (SDO). SCK i SS se nazivaju kontrolni signali, a MISO i MOSI podatkovni signali. *Master* potiče komunikaciju cijelo vrijeme, prvo konfigurira signal takta frekvencije manje ili jednake od frekvencije koju podržava periferni uređaj, a nakon toga postavi odgovarajući SS u nisku razinu i tako odabere s kojim od spojenih perifernih uređaja hoće ostvariti komunikaciju. Periferni uređaj koji nije na svoj SS ulaz primio nisku razinu neće prihvatiti ulazni SCK i MOSI signal upućen s *master*-a. Većina uređaja ima izlaze s tri stanja - visoko, nisko i stanje visoke impedancije tzv. *High Z state*. Stanje visoke impedancije postavlja se na izlaz koji nije u upotrebi, u ovom slučaju to će biti svi periferni uređaji koji nisu odabrani. Prijenos se odvija tako da *master* pošalje bit na MOSI, *slave* čita podatak taj podatak s MOSI-a, zatim on pošalje bit na MISO i tada *master* pročitava s ulaza MISO i tu prijenos jednog bita završava. Prijenos više-bitnih podataka odvija se najčešće pomoću 8-bitnog posmačnog registra (Slika 6.1-2.) koji se nalazi u *master*-u i u *slave*-u .



**Slika 6.1-2. Posmačni registar**

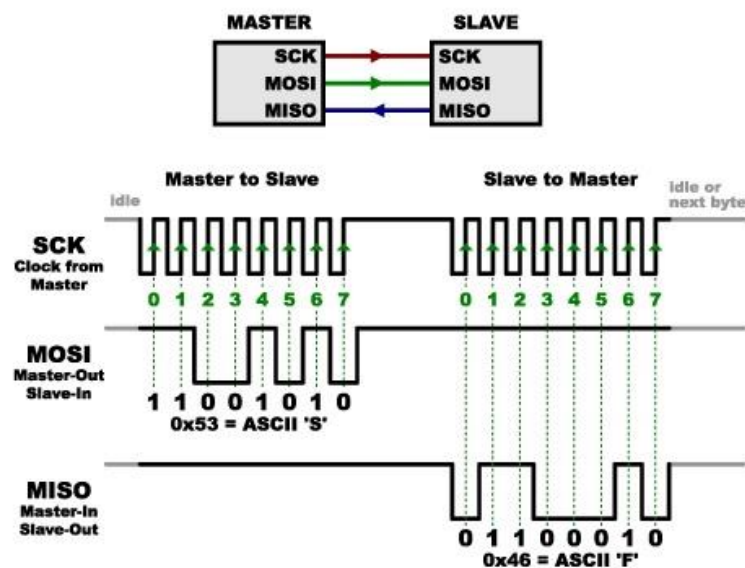
Oni su povezani u jedan ciklus: dok *master* posmiče i postavlja vrijednost registra na MOSI, *slave* prima bitove i stavlja ih u registar i posmiče ih dok se registar ne napuni. Prijenos započinje slanjem bita koji se naziva najznačajniji bit (eng. *Most Significant Bit* – *MSB*), zatim slijedi još 6 bitova te na kraj opet dolazi bit koji se naziva zadnji značajan bit (eng. *Least Significant Bit- LSB*) (Slika 6.1-3.)





**Slika 6.1-3. Prijenos bitova s posmačnim registrima**

Nakon što su podaci prebačeni, svaki uređaj u registrima ima podatke s kojima može obavljati operacije poput spremanja u memoriju. Ako postoji još podataka koje treba prebaciti, registar u *master*-u se puni s podacima, *master* generira signal takta i postavlja nisku razinu na izlaz SS te cijeli proces počinje ponovno. Kada nema više podataka za prijenos, *master* zaustavlja signal takta i proces se završava. Konačno, prebacivanje s MOSI i MISO, odnosno s *master*-a na *slave* i obratno prikazan je na **Slika 6.1-4**.



Slika 6.1-4. Konačni prijenos podataka u oba smjera.

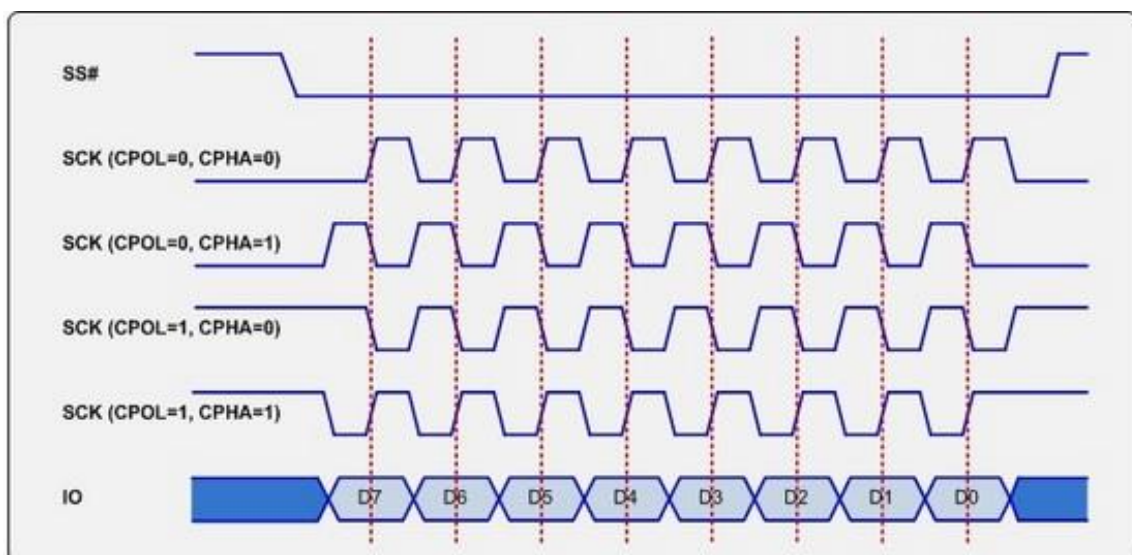
Polaritet (eng. *CPOL*) i faza (eng. *CPHA*) signala takta je još jedan par parametara koji određuju vrstu serijskog prijenosa podataka. Ovi parametri određuju bridove signala takta koji određuju početak slanja i uzorkovanja podataka koji su u prijenosu. Točnije, pri namještanju frekvencije kojom će raditi *master* potrebno je također konfigurirati i polaritet i fazu signala. Ovisno o tome kako su konfigurirani signal i faza, sinkronizacija serijske komunikacije može imat četiri različita načina rada (Slika 6.1-5).

Mode	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

Slika 6.1-5. SPI načini rada

Ako je faza signala takta u niskoj razini odnosno  $CPHA=0$ , podaci se prebacuju na rastući brid signala takta s polaritetom u niskoj razini, a ako je  $CPOL=1$  tada se podaci prebacuju na padajući signal takta. Kada faza ima vrijednost visoke razine tj.  $CPHA=1$ , polariteti su obratni. Padajući signal takta je signal za prebacivanje podataka kada je  $CPOL=0$ , inače je

rastući uz suprotan CPOL. Kod većine mikrokontrolera faza i polaritet mogu biti prilagođeni. U konačnici cijeli sinkroni prijenos podataka ovisi o tome u kojem se načinu rada nalazi sustav. Na **Slika 6.1-6.** prikazan je prijenos podataka u ovisnosti o fazi i polaritetu na način koji je upravo objašnjen.



**Slika 6.1-6. Prijenos podataka u ovisnosti o fazi i polaritetu**

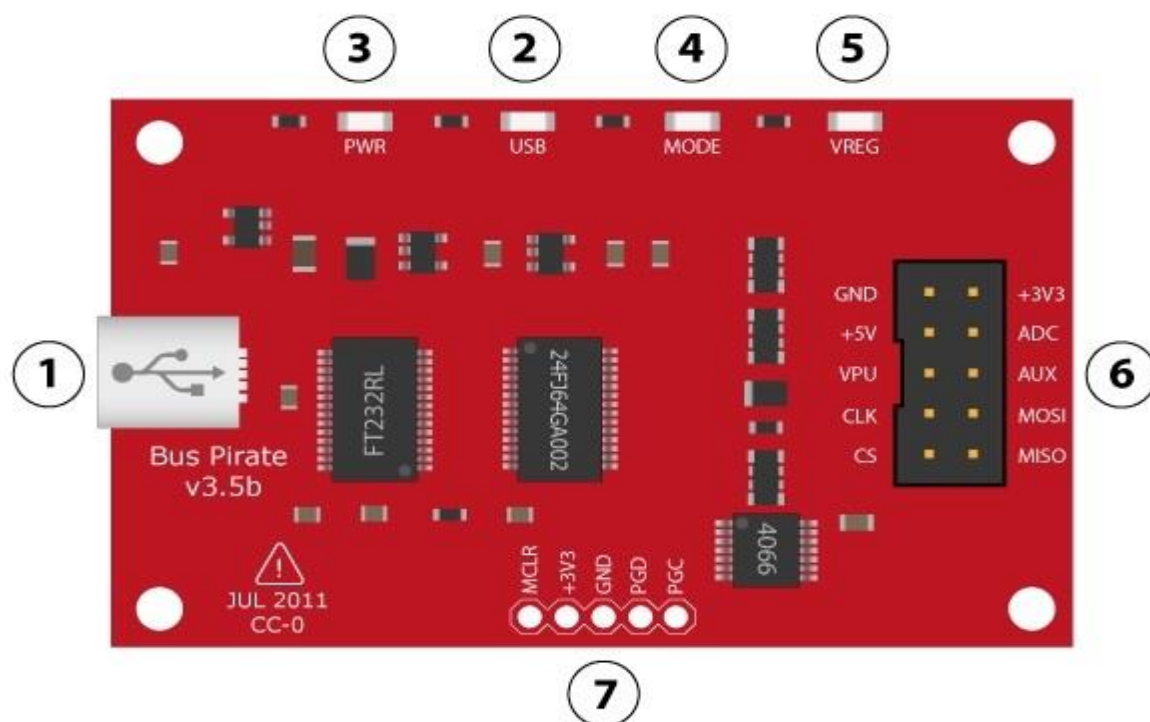
Sinkrona razmjena podataka može doseći brzinu od 1 megabaud ili 10 Mbps. Baud se često koristi za mjernu jedinicu koja označava brzinu prijenosa podataka, nije ekvivalent mjerne jedinici bit/s.

$$T = \frac{1}{\text{Baud}} [s]$$

Serijski prijenos podataka je poželjno koristiti među tiskanim pločicama, jer je on prvenstveno napravljen za vrlo brzi prijenos podataka između raznih digitalnih komponenata. Zbog toga sabirnica koja povezuje dva uređaja koja komuniciraju ne smije biti predugačka, jer se parazitnih kapaciteta povećava vrijeme porasta i smanjuje brzina prijenosa.

## 6.2. *Bus Pirate*

*Bus Pirate* je univerzalno sučelje koje komunicira s velikim brojem različitih integriranih sklopova putem terminala na računalu. Ovakav pristup programiranju i pokretanju novog integriranog sklopa uštedi mnogo vremena koje se inače gubi na prototipiranje i pokušaje uspostavljanja komunikacije i programiranja. *Bus Pirate* korišten u ostvarenoj SPI komunikaciji ima ulogu *master* uređaja. Na sebi sadrži nekoliko izvoda i led indikatorskih svjetala grupiranih u 7 cjelina, koje se mogu vidjeti na **Slika 6.2-1**.



Slika 6.2-1. *Bus Pirate* s izvodima

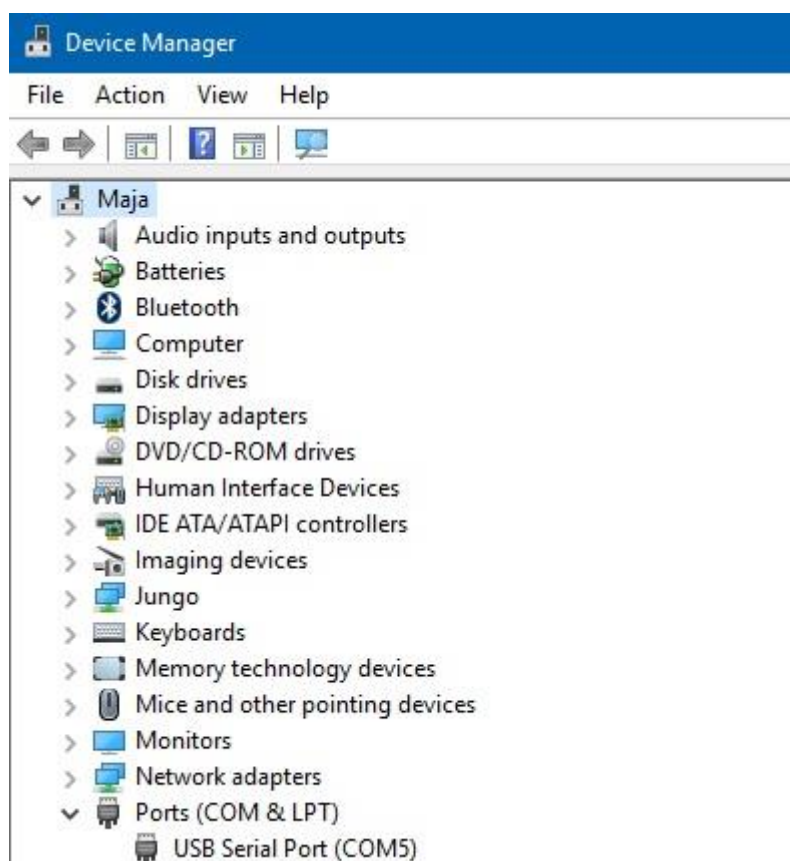
Cjeline su redom podijeljene na:

1. *Mini-B USB port* - ovaj utor povezuje *Bus Pirate* s računalom. Njegovo napajanje ostvareno je preko USB utora, kao i prijenos podataka s računalom.
2. *USB transmit indicator* – kada se prenose podaci između računala i *Bus Pirate*-a, ovo LED indikatorsko svjetlo je upaljeno.
3. *Power indicator* - svijetli kada je priključen na napajanje.
4. *Mode indicator* - kada je *Bus Pirate* konfiguriran za promjenu načina rada, tada svijetli ovo LED indikatorsko svjetlo, a također izvodi mogu biti aktivni kada je indikator načina rada upaljen. Izvodi su u stanju visoke impedancije kada je indikator načina rada ugašen.
5. *Voltage regulator indicator* – Ovo LED svjetlo je uključeno kada je u terminal upisana naredba 'W', odnosno kada je omogućeno napajanje s pločice.
6. *I/O pins* – ova cjelina se sastoji od 2x5 izvoda koji povezuju *Bus Pirate* s vanjskim čipovima. Tablica izvoda je prikazana na **Slika 6.2-2**
7. *In circuit serial programming (ICSP) header* – izvodi koji služe za programiranje PIC 24FJ64GA002 mikrokontrolera. Ovi pinovi mogu biti korišteni za programiranje novog firmware-a u mikrokontroler.

Pin Name	Description (Bus Pirate is the master)
MOSI	Master data out, slave in (SPI, JTAG), Serial data (1-Wire, I2C, KB), TX* (UART)
CLK	Clock signal (I2C, SPI, JTAG, KB)
MISO	Master data in, slave out (SPI, JTAG) RX (UART)
CS*	Chip select (SPI), TMS (JTAG)
AUX	Auxiliary IO, frequency probe, pulse-width modulator
ADC	Voltage measurement probe (max 6volts)
Vpu	Voltage input for on-board pull-up resistors (0-5volts).
+3.3v	+3.3volt switchable power supply
+5.0v	+5volt switchable power supply
GND	Ground, connect to ground of test circuit

**Slika 6.2-2. Opisi ulazno-izlaznih izvoda**

Pri prvom pokretanju *Bus Pirate*-a, potrebno je instalirati *PuTTY* ili *Tera Term* terminal, kao i driver 2.08.28 kojeg je moguće skinuti sa službene stranice<sup>8</sup> tvrtke FTDI. Sljedeće je potrebno pronaći na kojem COM *port*-u je povezan *Bus Pirate*, kako bi mogli konfigurirati *PuTTY*. To se može pronaći u *Start->Settings->Control panel->System->Hardware->Device manager*. Otvaranjem padajućeg izbornika *Ports*, pronalazi se '*USB Serial Port*' npr. COM5 (**Slika 6.2-3**).

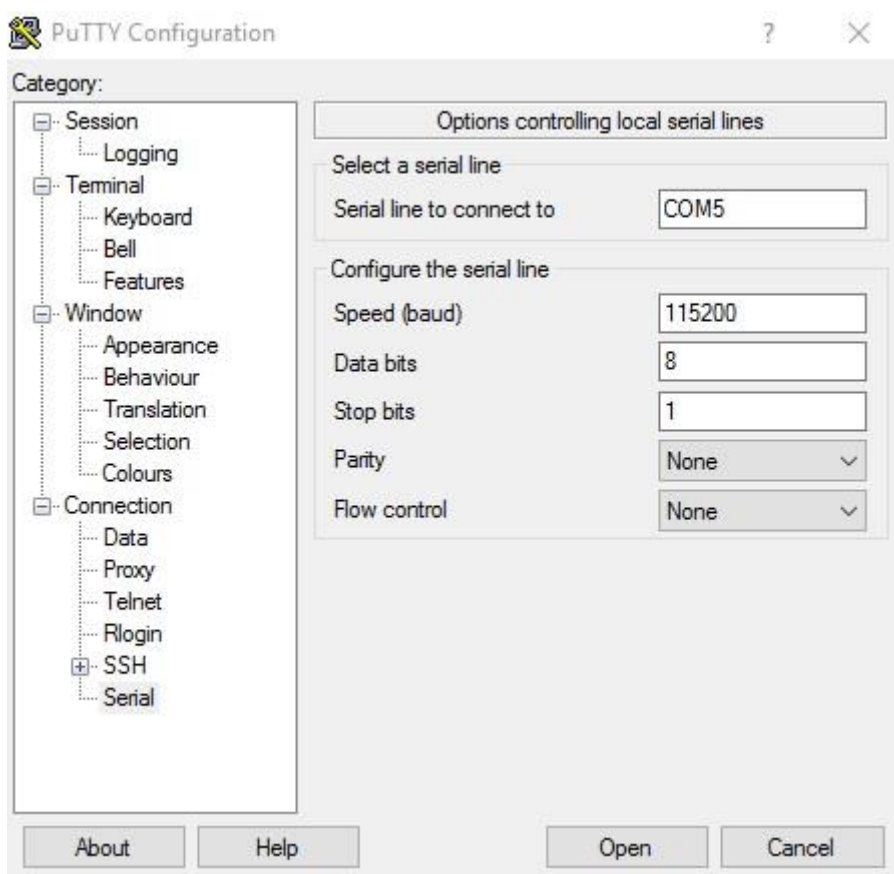


**Slika 6.2-3. Prikaz ulaza COM5**

---

<sup>8</sup> <http://www.ftdichip.com/Drivers/VCP.htm>

Nakon što je poznat ulazni COM *port*, potrebno je konfigurirati postavke terminala. Potrebne modifikacije postojećih postavki s ciljem ostvarenja komunikacije s *Bus Pirate*-om bit će demonstrirane na terminalu *PuTTY*. Prvo je potrebno otvoriti terminal i klikom na *Session* otvori se prozor s osnovnim postavkama, ondje je potrebno označiti *serial* zatim u *port* upisati COMx (gdje je x broj koji ste pronašli u gornjim koracima), *speed* je potrebno postaviti u 115200. Zatim u izborniku lijevo klikom na *serial* otvara se prozor gdje je sve već namješteno, osim *Flow control* koji treba biti postavljen na *None*. U konačnici prozor treba izgledati kao na **Slika 6.2-4**.



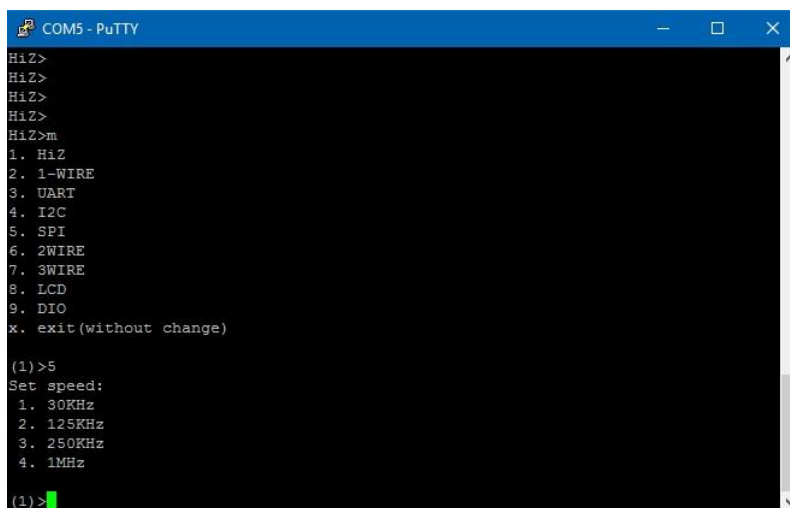
**Slika 6.2-4. Postavke Putty-a**

Slijedeće je potrebno provjeriti *terminal setup*. Odabrati VT100 tip terminala, CR postavke i ugasiti *local echo*.



**Slika 6.2-5. Terminal setup – Keyboard**

Nakon toga pritiskom na gumb *Open* otvara se terminal s konačnim postavkama. Ako je terminal prazan, pritiskom na *enter* prikazuje se naredbeni redak *HiZ>*. Početno stanje svih izvoda je stanje visoke impedancije pa je zato potrebno odabrati način rada kojeg želimo. Unosom znaka „?“ i tipke *enter* prikazuje se blok informacija vezanih za *Bus Pirate*. Unosom slova „m“ pojavi se popis više načina rada, odabrati način rada broj 5 (SPI) (**Slika 6.2-6**).



```
COM5 - PuTTY
HiZ>
HiZ>
HiZ>
HiZ>
HiZ>m
1. HiZ
2. 1-WIRE
3. UART
4. I2C
5. SPI
6. 2WIRE
7. 3WIRE
8. LCD
9. DIO
x. exit(without change)

(1)>5
Set speed:
1. 30KHz
2. 125KHz
3. 250KHz
4. 1MHz

(1)>
```

**Slika 6.2-6. Konfiguracija *Bus Pirate*-a**

Ovaj modul ima još dosta opcija za konfiguraciju, poput postavke brzine i polariteta signala takta, aktivne padajuće ili rastuće rubove signala takta i mnoge druge opcije. Nakon što odaberemo brzinu, poželjno je sve ostalo ostaviti u *default* stanju, te se konačno dobije poruka *Ready* i *SPI>*. Sada je moguće upisati podatke koji se žele prenijeti na FPGA putem sinkrone serijske veze preko *Bus Pirate*-a. (**Slika 6.2-7**)



```
(1)>2
Ready
SPI>0xFF
WRITE: 0xFF
SPI>
```

**Slika 6.2-7. SPI komunikacija**



## 6.3. Implementacija SPI protokola na iCE40-HX1K

U nastavku će biti objašnjeno kako implementirati SPI protokol na iCE40-HX1K razvojnoj pločici, kao i povezivanje prethodno napravljene PWM komponente SPI protokolom.

### 6.3.1. Implementacija serijske komunikacije

Ovaj protokol se sastoji od samo jednog modula koji je nazvan `spi_slave`. Entitet `spi_slave`-a se sastoji od četiri ulaza i pet izlaza, ulazi su CLK, MOSI, SS, SCLK, a izlazi četiri LED indikatorska svijetla i MISO. MISO je na početku potrebno staviti u stanje visoke impedancije, kako bi to bilo moguće potrebno je dodati još jednu biblioteku uz ove standardne `'USE IEEE.STD_LOGIC_ARITH.ALL'`. Za realizaciju SPI protokola potrebno je prvo sinkronizirati signal takta koji se nalazi na samom iCE40-HX1K sa signalom takta koji šalje *Bus Pirate* putem SCLK-a, kako se neki podaci ne bi izgubili zbog različitih frekvencija rada. Sinkronizacija je ostvarena tako da se uzorkuju svi ulazni signali koji se zatim pomoću dva *shift* registra sinkroniziraju. Nadalje, proces je u ovom slučaju okidan samo na signal takta iCE40-HX1K, a unutar procesa se odvija prebacivanje podatka s *master*-a na *slave* tako da se postavi brojač prebačenih bitova indeks na 7 kada je signal SS<sup>9</sup> na izlazu iz drugog *shift* registra jednak 1 a izlaz iz prvog 0 što bi značilo da je prošlo dva signala takta i da je to sigurno taj podatak. Kada je SS u niskoj logičkoj razini i SCLK prelazi iz niske logičke u visoku logičku razinu tada se obavlja prebacivanje bitova s MOSI u *RxdData* pomoću logičkog operatora `&`. Pozornost treba obratiti na to da je *RxdData* vektor te je njegov najviši bit adresa na koju se piše tijekom prijenosa podataka pa zbog toga treba izvršiti operaciju `&` s donjih 6 bitova *RxdData* registra. Navedeni postupak se ponavlja dokle god brojač nije 0, odnosno dok se ne prebace svi bitovi. U LED indikatorska svijetla upisuju se donjih 4 bita prebačenog podatka koja služe za provjeru programskog rješenja.

---

<sup>9</sup> SS je aktivan u niskom logičkom stanju što znači ako je u visokoj razini trenutno nije aktivan te je potrebno inicijalizirati brojač bitova.

## **spi.vhd**

```
architecture Behavioral of spi_slave is

    signal SCLK_prvi, SCLK_old : std_logic;
    signal SS_prvi, SS_old : std_logic;
    signal MOSI_prvi: std_logic;
    signal index: natural range 0 to 7;
    signal RxdData : std_logic_vector(7 downto 0);

begin
    process(CLK_in)
    begin
        if( CLK_in'event and CLK_in= '1') then
            SCLK_prvi <= SPI_CLK;
            SCLK_old <= SCLK_prvi;
            SS_prvi <= SPI_SS;
            SS_old <= SS_prvi;
            MOSI_prvi <= SPI_MOSI;

            if (SS_old = '1' and SS_prvi = '0') then
                index <= 7;
            end if;

            if( SS_prvi = '0' ) then
                if(SCLK_old = '0' and SCLK_prvi = '1') then
                    RxdData <= RxdData(6 downto 0) & MOSI_prvi;
                    if(index = 0) then
                        index <= 7;
                    else
                        index <= index-1;
                    end if;
                end if;
            end if;
        end if;
    end process;

    LED0 <= RxdData(0);
    LED1 <= RxdData(1);
    LED2 <= RxdData(2);
    LED3 <= RxdData(3);

end Behavioral;
```

### 6.3.2. Sinteza i implementacija

Potrebno je još napraviti *pcf* datoteku. U *datasheetu* iCE40-HX1K se nalaze brojevi izvoda na kojima se nalaze SPI konektori potrebni za realizaciju gore opisane programske podrške. Nakon definiranja, potrebno je ponoviti postupak opisan u poglavlju 4.2.

#### Izvodii.pcf

```
# LED outputs
set_io LED0 59
set_io LED1 56
set_io LED2 53
set_io LED3 51

#SPI
set_io SPI_CLK 48
set_io SPI_SS 49
set_io SPI_MOSI 46
set_io SPI_MISO 45

# 3.3 MHz clock input
set_io CLK_in 13

# SPI Flash enable control
set_io SS_B 49
```

### 6.3.3. Serijski upravljana pulsno-širinska modulacija

Konačno, za izvođenje pulsno-širinske modulacije koja je upravljana putem SPI, potrebno je spojiti prethodno definirane komponente u jednu funkcionalnu cjelinu. Potrebno je koristiti *main* i *PWM* komponentu iz 5.3. poglavlja te *spi\_slave* komponentu iz 6.3.1. poglavlja. Unutar *main*-a potrebno je dodati izlaze i ulaze korištene u SPI protokolu, kao i cijelu SPI komponentu na isti način kako je dodana i *PWM* komponenta u poglavlju 5.2.

#### main\_spi\_pwm.pcf

```
entity main is
  port(
    + CLK_in           : in std_logic;
    + SPI_CLK          : in std_logic;
    + SPI_SS           : in std_logic;
```

```

        + SPI_MOSI      : in  std_logic;
        + SPI_MISO      : out std_logic := 'Z'
    );
end main;

```

Unutar SPI komponente dodati novi izlaz *PWM\_pos* tipa *std\_logic\_vector(7 downto 0)*, preko kojega će se prenijeti vrijednost registra s podatkom koji bude prenesen. U *port map*-u za *spi\_slave* je potrebno taj signal povezati sa novim signalom *PWM\_pozicija* koji je istog tipa kao i *PWM\_pos*. Unutar SPI komponente prije ključne riječi *end*, dodati 'PWM\_pos <= RxdData'. Ova linija kôda sadržaj *RxdData* registra prebacuje u *PWM\_pos*.

#### **main\_spi\_pwm.pcf**

```

signal PWM_pozicija: std_logic_vector(7 downto 0);

component spi_slave
    port(
        .
        .
        .

        PWM_pos      : out std_logic_vector(7 downto 0)
    );
end component;

component PWM
    port(
        clk          : in std_logic;
        pwm_out       : out std_logic;
        position      : in std_logic_vector(7 downto 0)
    );
end component;

begin

    spi: spi_slave port map (
        .
        .
        PWM_pos => PWM_pozicija
    );

    pwm0: PWM port map (
        .
        .
        position => PWM_pozicija,
    );

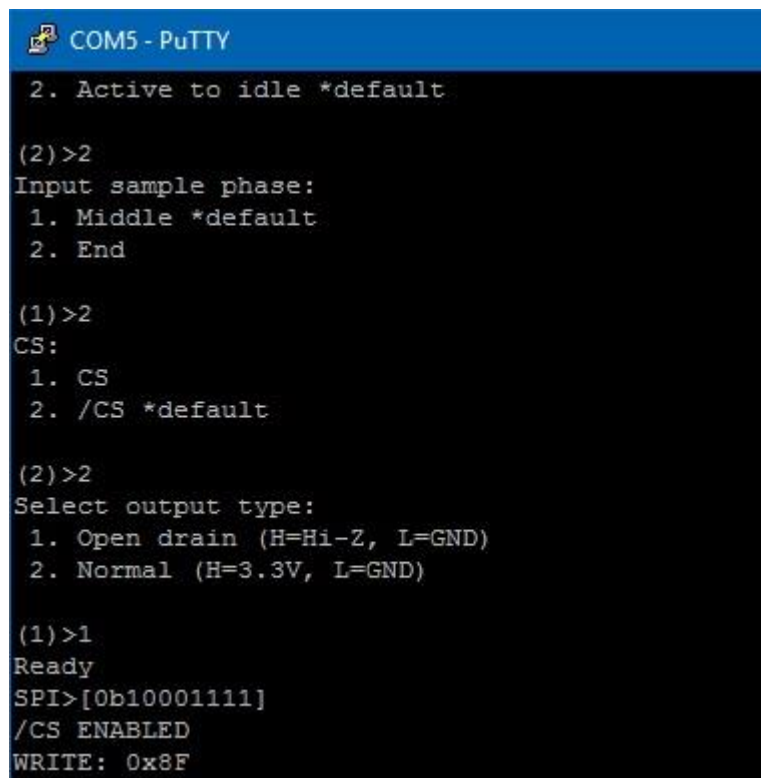
```

PWM komponenti potrebno je dodati samo novi ulaz *position*. U prijašnjem kodu *pozicija* je bila konstanta, a ovdje je promjenjiva, odnosno poprima vrijednost koja se nalazi u

*RxdData* registru čija će se vrijednost putem izlaza *PWM\_pos* i signala *PWM\_pozicija* prenijeti na ulaz *position*. Nakon obavljenih modifikacija, ponovno postupiti kako je opisano u poglavlju 6.3.2.

#### 6.3.4. Povezivanje Bus Pirate-a i iCE40-HX1K

Nakon što je uspješno sintetiziran i prebačen program na iCE40-HX1K, potrebno je povezati fizički *Bus Pirate* s FPGA. Izvode *Bus Pirate*-a treba povezati na izvode koji se nalaze na desnoj strani FPGA pločice s imenima redom SS, SO, SI, SCK, GND. Nije potrebno povezivati 3,3V jer to u ovom slučaju nije dovoljno za rad pločice. Potom povezane komponente s USB kabelom priključiti na računalo. Otvoriti terminal *PuTTY* te, kako je opisano u poglavlju 6.2., konfigurirati za rad kao SPI. Odabirom SPI otvara se još novih postavki. Sve postavke konfigurirati tako da se uvijek odabere *default* vrijednost. Odabir frekvencije neka bude 30kHz, *input sample phase* odabrati broj 2. Nakon konfiguracije slijedi upis podatka za prijenos. Podatak je potrebno upisati u uglatim zagrada, gdje prva uglati zagrada znači početak, a druga kraj. (Slika 6.3-1)



```
COM5 - PuTTY
2. Active to idle *default

(2)>2
Input sample phase:
  1. Middle *default
  2. End

(1)>2
CS:
  1. CS
  2. /CS *default

(2)>2
Select output type:
  1. Open drain (H=Hi-Z, L=GND)
  2. Normal (H=3.3V, L=GND)

(1)>1
Ready
SPI>[0b10001111]
/CS ENABLED
WRITE: 0x8F
```

Slika 6.3-1. Konfiguracija *Bus Pirate*-a i konačni prijenos podatka

## 7. Zaključak

Programibilna logika koja je u današnje vrijeme zauzela važno mjesto u industriji, ovim završnim radom je približena početnicima u ovom području. Detaljnim pojašnjenjima instalacije programske podrške IceCube2 na operacijskim sustavima Windows i Linux, kao uputama za implementaciju rješenja otvorenoga koda iCEStorm podržanog na operacijskom sustavu Linux, olakšano je početno snalaženje u programskoj potpori za konkretni razvojni sustav iCE40-HX1K.

Za izradu digitalnih komponenti na FPGA nužno je poznavati jedan od dva programska jezika, VHDL ili Verilog. Ovim radom obuhvaćena je i detaljno objašnjena izrada jednostavnih digitalnih komponenata unutar programskog jezika VHDL. Objašnjena je izrada pulsno-širinske modulacija (eng. *PWM*) i serijske komunikacije (eng. *SPI*). Serijska komunikacija omogućuje povezivost digitalnog sustava s ugradbenim računalnim sustavima. Putem serijske komunikacije moguće je upravljati pulsno-širinskom modulacijom, a zatim pulsno-širinskom modulacijom dalje upravljati gotovo svim elektroničkim sustavima gdje je potrebna regulacija snage ili intenziteta signala.

Upute o izradi ovih digitalnih komponenata omogućuju brzo upoznavanje s mogućnostima FPGA tehnologije, kao i daljnje kreiranje i izradu digitalnih komponenti koje će biti vrlo korisne u industrijskim okruženjima.

---

Maja Ovčarik

## 8. Literatura

- [1] VUČIĆ, M: Upotreba mikrokontrolera u ugradbenim računalnim sustavima, Zagreb, 2007.
- [2] RAJEWSKI, J.: “Pulse-Width Modulation”, s Interneta, <https://embeddedmicro.com/tutorials/mojo/pulse-width-modulation>, 2014.
- [3] PALART, “PWM-VHDL”, s Interneta, <https://github.com/PalArt/PWM-VHDL/blob/master/main.ucf> , 27.10.2015.
- [4] ”ICE40™ LP/HX family data sheet”, s Interneta, <http://www.datasheets360.com/pdf/-7137632060064251052> , Ožujak, 2015.
- [5] CLIFFORD, W: “Project IceStorm”, s Interneta, <http://www.clifford.at/icestorm/> , 2015.
- [6] “FPGA for beginners”, s Interneta, <https://www.nandland.com/articles/fpga-101-fpgas-for-beginners.html> , 2014.
- [7] STOREY, T: “PWM Generator (VHDL) ” , s Interneta, <https://eewiki.net/pages/viewpage.action?pageId=20939345> , 2016.
- [8] ASHENDEN, P.J; LEWIS, J: “The designer’s guide to the VHDL” , 2008.
- [9] “Bus Pirate Tutorial”, s Interneta, [http://dangerousprototypes.com/docs/Bus\\_Pirate\\_101\\_tutorial](http://dangerousprototypes.com/docs/Bus_Pirate_101_tutorial) , 2013.
- [10] AVAREZ, DANIEL: “SPI communications – slave core VHDL” , s Interneta, <http://dani.foroselectronica.es/spi-communications-slave-core-vhdl-137/> , 2016.
- [11] STOREY, T: “SPI Slave (VHDL) ” , s Interneta, <https://eewiki.net/pages/viewpage.action?pageId=7569477> , 2016.

# Sažetak

## **Naslov: Izrada digitalnih komponenti za FPGA sklopove u industrijskom okruženju**

U radu je objašnjena izrada digitalnih komponenti na iCEblink40-HX1K razvojnoj okolini koja je idealna za prvi susret s FPGA sklopovima. Detaljno je pojašnjena instalacija programske potpore iCEcube2 na operacijskim sustavima Windows i Linux. Na navedenoj FPGA platformi izrađena je pulsno-širinska modulacija (eng. *PWM*) i sinkrona serijska komunikacija (eng. *SPI*). Sinkrona serijska komunikacija omogućava povezivost digitalnog sustava s ugradbenim računalnim sustavima. Putem serijske komunikacije moguće je upravljati pulsno-širinskom modulacijom. Proučene su i dokumentirane značajke i mogućnosti FPGA tehnologije u ovom radu, koje omogućuju daljnje kreiranje i izradu digitalnih komponenti koje će biti korisne u industrijskim okruženjima.

**Ključne riječi:** FPGA, VHDL, PWM, iCEcube2, sinkrona serijska komunikacija, SPI, iCEStorm, Bus Pirate



# Summary

## **Title: Digital components development for FPGA in industrial environment.**

This thesis contains guidelines for digital design on iCEblink40 - HX1K platform which is suitable platform for beginners learning FPGA technology. The installation procedures for iCEcube2 development environment for Windows and Linux were documented and presented. Pulse-width modulation (PWM) and synchronous serial interface (SPI) digital components were developed for this FPGA platform. SPI enables the connectivity of a digital system with other embedded system. PWM component is controlled via SPI interface. These features enabling the basis for further development and production of new digital components that can be used in industrial environments.

**Key words:** FPGA, VHDL, PWM, iCEcube2, serial peripheral interface, SPI, iCEStorm, Bus Pirate