SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

SEMINAR

Programibilni logički sklopovi

Ivan Dodig

Voditelj: dr. sc. *Hrvoje Mlinarić*

Zagreb, travanj, 2007

Sadržaj

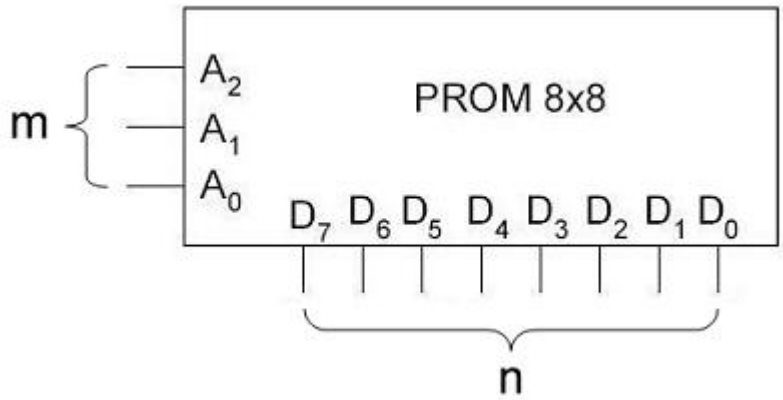
Uvod

Programibilni logički sklopovi, ili **PLD** (**P**rogrammable **L**ogic **D**evices), su elektronički sklopovi koji se koriste za izradu digitalnih sklopova i sustava. Digitalni sklopovi općenito obavljaju određenu logičku funkciju. Programibilni sklopovi su posebni po tome što im funkcija nije određena u trenutku proizvodnje, već se *upisuje naknadno*. To znači da u početku isti *komadi hardvera* mogu obavljati različite logičke funkcije. Logičke funkcije se opisuju programskim jezicima poznatim pod imenom **HDL** (**H**ardware **D**escription **L**anguage) među kojima su najpoznatiji Verilog i VHDL. Njih dalje prevode logički *compileri* u oblik pogodan za upis u sklop.
U moderno vrijeme to se radi uporabom **CAD** (**C**omputer **A**ided **D**esign) alata, kao što je to Xilinx ISE koji objedinjuje sve potrebne komponente razvoja u jedan zajednički razvojni sustav.

Upis funkcije vrši se **programatorom**.

1.1 Počeci PLD sklopova

Prije same pojave PLD-ova, jednostavne logičke funkcije mogli smo realizirati uporabom **ROM** (**R**ead **O**nly **M**emory) memorija. Svaka memorija u osnovi sadrži adresne ulaze i podatkovne izlaze.



Slika 1. Shematski prikaz PROM memorije

Ako na ulaze umjesto adresa dovedemo varijable, ovisno o sadržaju adresirane memorijske lokacije, izlazi će poprimiti određene logičke vrijednosti. *Pametnim* programiranjem memorijskih lokacija možemo ostvariti različite funkcije.
Uzmimo da naša memorija ima ***m*** ulaza. Teoretski, postoji ravno $\mathbf{2}^{m}$ različitih logičkih funkcija koje se mogu realizirati, no ograničeni smo na ***n*** podatkovnih izlaza. Memorija na slici je 8x8, što znači da možemo

realizirati teoretski maksimum funkcija. Takve memorije su rijetkost, jer bi za veći kapacitet morale imati ogroman broj izlaznih izvoda. Dakle možemo realizirati PLD koji će obavljati maksimalno ***n*** logičkih funkcija (svaka ima svoj izlaz) od ***m*** ulaznih varijabli.
Ovakav jednostavan sklop omogućuje realizaciju logičkih funkcija, međutim ima određene nedostatke:

- ROM memorije su spore
- Funkcije se ne mogu minimizirati pa je uporaba sklopovlja neefikasna
- Bez dodatnih elektroničkih sklopova nije moguće realizirati sekvencijalne sklopove. Da bi to postigli, potrebno je u sustav dodati bistabile, tj. neku vrstu memorijskih elemenata

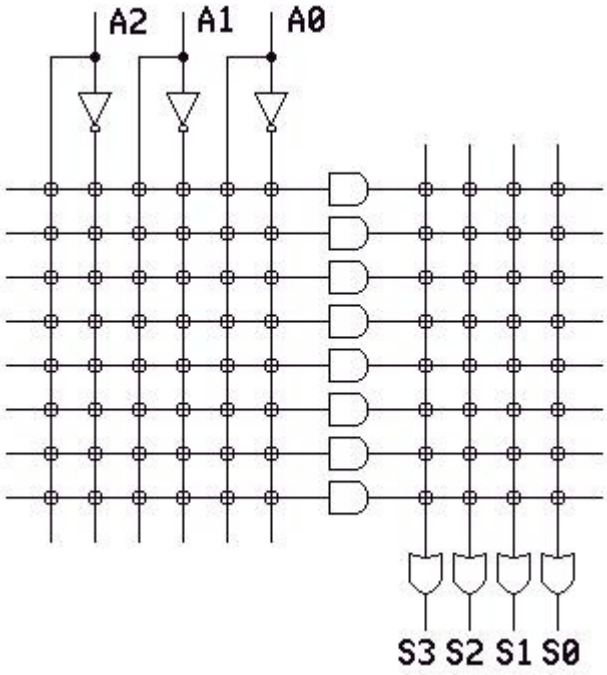
1.2 Pregled PLD sklopova

Danas se pri konstrukciji digitalnih sustava većinom koriste standardizirani moduli (PLD) za ostvarivanje logičkih funkcija. Alternativa njima su **ASIC** (**A**pplication-**S**pecific **I**ntegrated **C**ircuit) sklopovi. Oni su posebno dizajnirani kako bi obavljali jednu zadaću i nakon što su proizvedeni funkciju im više nije moguće mijenjati. U smislu brzine, ASIC sklopovi predstavljaju vrh ponude.
Jasno je da je razvoj takvog sklopa skup i dugotrajan. Bilo kakva izmjena funkcije koju obavljaju zahtjeva novu proizvodnju. Iz tog razloga danas ih sve više i više istiskuju moderni PLD sklopovi, kao što su to **FPGA** i **CPLD**.

70'-ih godina prošlog stoljeća na tržištu se pojavljuju prvi programibilni sklopovi posebno dizajnirani za ostvarivanje logičkih funkcija, za razliku od programirljivih memorija opisanih u poglavlju 1.1.

Programirljiva logička polja ili **PLA** (**P**rogrammable **L**ogic **A**rray) prvi su primjer **SPLD** (**S**imple **PLD**) sklopova. Sastoje se od dvije programirljive matrice na čijim su krajevima I, tj, ILI sklopovi. Logičke funkcije se prije upisa minimiziraju, što osjetno smanjuje potreban broj sklopovlja.

Programiranje funkcije moguće je ostvariti tijekom proizvodnje po želji naručitelja, ali to bitno umanjuje mogućnosti koje se kriju iza PLD sklopova. Cilj je dobiti sklop koji korisnik može sam programirati (**in**-**the**-**filed**), po mogućnosti više puta. To se postiže dodavanjem **EEPROM** (**E**lectrically **E**rasable **P**rogrammable **ROM**) ćelija u matricu. Korisnik programira te ćelije čime određuje logičku funkciju sklopa.



Slika 2. PLA sklop s dvije programirljive ravnine

Sljedeća slična tehnologija naziva se **PAL** (**P**rogrammable **A**rray **L**ogic). PAL sklopovi su skoro identični PLA sklopovima, s jednom osnovnom razlikom. Programirati se može samo matrica I-sklopova. Takva je struktura jednostavnija pa samim time i jeftinija. Povećana je brzina rada zbog kraćih električnih vodova. Jasno je da je ovakva struktura manje fleksibilna od PLA strukture koja raspolaže s 2 programirljive ravnine.

Za implementaciju većih i složenijih digitalnih sustava SPLD programibilni sklopovi često nisu dovoljno *moćni* pa su se na tržištu pojavili **CPLD** (**C**omplex **PLD**) sklopovi. CLPD-ovi jednostavno objedinjuju više SPLD sklopova (PAL i/ili PLA) u jednu veću kompaktnu logičku cjelinu omogućujući na taj način projektiranje puno složenijih sustava nego što je to bilo moguće uporabom SPLD-ova.
Složene programirljive logičke naprave sadrže prospojene matrice vodova koje se također programiraju kako bi se ostavila željena funkcija. Prospojne matrice povezuju logičke blokove (PAL i PLA) u jednu zajedničku cjelinu.

Za najveće i najsloženije digitalne sustave danas se koriste **FPGA** (**F**iled **P**rogrammable **G**rid **A**rray) programirljiva polja. Za razliku od do sada opisanih tehnologija FPGA ne koristi I i ILI (NI i NILI) matrice kao osnovnu strukturu za definiranje funkcija. Logički blokovi se izvode na više načina, no prevladava uporaba **LUT** (**L**ook **U**p **T**able) preglednih tablica.

Više o FPGA sklopovima možete pročitati u nastavku ovog seminara.

FPGA

2.1 Uvod

Kao što ste već mogli pročitati u uvodu, alternativa svim PLD programibilnim sklopovima, kao i FPGA tehnologiji, su ASIC integrirani krugovi. U pogledu brzine, potrošnje energije, pouzdanosti te podržanoj kompleksnosti dizajna, ASIC je apsolutni pobjednik. Uvijek je najkvalitetnije rješenje dizajnirati sklop specifične namjene za određenu zadaću. Postavlja se pitanje „Zašto FPGA? Zašto koristiti općenito bilo koji PLD?"

U današnje vrijeme velikog tehnološkog napretka, različiti proizvođači se međusobno natječu u izradi boljih, bržih i jeftinijih digitalnih sustava. U takvom tržišnom ustrojstvu, FPGA u mnogim stvarima prednjači ASIC-u. Vrlo je bitno izdati proizvod u najkraćem mogućem roku gdje FPGA nudi puno kraći „*time to market*", tj. uvelike se skraćuje vrijeme dizajniranja sustava. Omogućeno je brzo ispravljanje bugova (***in the filed***), jednostavna nadogradnja postojećeg sklopa i naravno osjetno su smanjeni troškovi proizvodnje. FPGA se redovito koristi za simulaciju rada budućeg ASIC sklopa.

Moderni FPGA sklopovi omogućuju reprogramiranje, dok ostatak sustava normalno radi. Gledajući ne tako daleko u budućnost, možemo zamisliti ultra brze procesore koji vrlo brzo mijenjaju svoju arhitekturu, ovisno o zadaći koju moraju izvršitI.

Programibilni sklopovi temeljeni na FPGA tehnologiji pristuni su svuda oko nas. Primjena uključuje **DSP** (**D**igital **S**ignal **P**rocessing), vojne aplikacije, prepoznavanje govora, računalni hardver itd. Iako prvotno razvijeni da konkuriraju CPLD**-**ovima, FPGA sklopovi danas su nezamjenjivi u izradi kompleksnih digitalnih sustava.

2.2 Spartan-3 FPGA

Xilinx je trenutno najveći i najpoznatiji proizvođač FPGA sklopova. Spomena radi, udio na tržištu drže još Altera, Actel, Atmel i nekolicina manjih proivođača.

Spartan-3 familija FPGA sastoji se od 50000 do 5 miliona logičkih vrata. Familija je izravni nasljenik Spartan-2E FPGA-ova. Jasno je kako svaka nova familija nudi veće logičke resurse, više **RAM** (**R**andom **A**ccess **M**emory) memorije i veći ukupan broj Input/Output izvoda. Xilinx nudi ukupno 8 verzija Spartan-3 porodice. Arhitektura svih sklopova

u familiji je ista, a razlika je jedino u kompleksnosti funkcija koje mogu izvršavati, tj. broju logičkih cjelina koje sadrže. To naravno utječe na cijenu što daje korisniku na izbor sklop koji najviše odgovara njegovim potrebama.

Tablica 1. Pregled Spartan-3 familije sklopova [1]

Device	System Gates	Equivalent Logic Cells[1]	CLB Array (One CLB = Four Slices)			Distributed RAM Bits (K=1024)	Block RAM Bits (K=1024)	Dedicated Multipliers	Maximum User I/O	Maximum Differential I/O Pairs
			Rows	Columns	Total CLBs					
XC3S50[2]	50K	1,728	16	12	192	12K	72K	4	124	56
XC3S200[2]	200K	4,320	24	20	480	30K	216K	12	173	76
XC3S400[2]	400K	8,064	32	28	896	56K	288K	16	264	116
XC3S1000[2]	1M	17,280	48	40	1,920	120K	432K	24	391	175
XC3S1500	1.5M	29,952	64	52	3,328	208K	576K	32	487	221
XC3S2000	2M	46,080	80	64	5,120	320K	720K	40	565	270
XC3S4000	4M	62,208	96	72	6,912	432K	1,728K	96	712	312
XC3S5000	5M	74,880	104	80	8,320	520K	1,872K	104	784	344

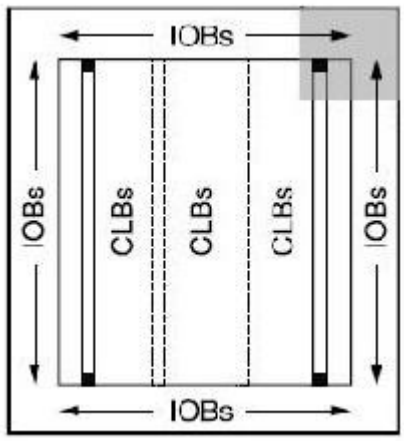
Pogledajmo osnovnu arhitekturu Spartan-3 **FPGA**-ova.

Ona se sastoji od 5 osnovnih programibilnih elemenata:

- **C**onfigurable **L**ogic **B**locks (**CLB**)
- **I**nput **O**utput **B**locks (**IOB**)
- RAM memorija
- 18-bitni sklopovi za množenje
- **D**igital **C**lock **M**anager (**DCM**)

Na slici 4, prikazana je općenita arhitektura FPGA sklopa. Može se vidjetI da se CLB-vi nalaze u centru, okruženi ulazno-izlaznim blokovima. Takav raspored je logičan jer CLB-ovi obrađuju signale koji dolaze i odlaze iz sklopa.
Logički blokovi CLB sadrže RAM bazirane LUT tablice koje igraju glavnu ulogu u implementaciji funkcija. Također služe i za pohranu podataka.

Svi su CLB-ovi međusobno povezani. Veza između pojedinih može ili ne mora biti ostvarena ovisno o položaju sklopki koje se također programiraju s ciljem ostvarenja logičke funkcije.



Slika 3. FPGA općenito [1]

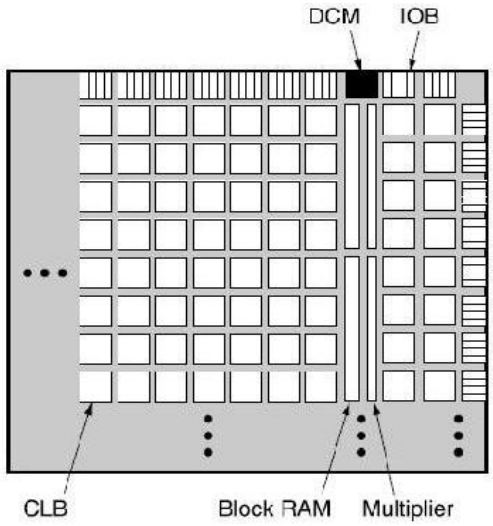
IOB ili ulazno-izlazni blokovi pružaju programirljivu, dvosmjenu vezu između izvoda (eng „pin") sklopa i FPGA interne logike. Drugim riječima, oni omogućuju komunikaciju s vanjskim svijetom. Podržano je i stanje visoke impedancije.
Ponekad je potrebno u „multi master" sustavima sklop potpuno odspojiti sa sabirnice kako bi se ona prepustila nekom drugom sklopu. To se radi tako da sklop koji prepušta sabirnicu postavi svoje izvode u stanje visoke impedancije i na taj način ne utječe na stanja na sabirnici.

Sve izvode FPGA sklopa možemo podijeliti u dvije kategorije:

- **Izvodi specijalne namjene**
  Otprilike 20-30% svih izvoda je specijalne namjene što znači da su posebno namijenjeni određenoj zadaći. To se odnosi na izvode za napajanje, ulaz radnog takta te izvode namijenjene programiranju sklopa.
- **Korisnički izvodi**
  Ostali izvodi koji se mogu programirati kao ulazni, izlazni ili biti u stanju visoke impedancije.

FPGA sklop sadrži RAM memoriju. Nije potrebno posebno objašnjavati njenu primjenu. Npr. ukoliko FPGA sklop simulira mikroprocesor, memorija služi za pohranu instrukcija i podataka.
Spartan-3 FPGA sadrži i gotove sklopove za množenje. Njihov broj ovisi o odabranom modelu, a kreće se od 4 u XC3S50 do 104 jedinice u XC3S5000. Svi primaju 2 18-bitna operanda u formatu dvojnog komplementa, dok se rezultat očekivano sprema u 36 bita. Zbog vrlo složenih zadaća koje FPGA sklopovi danas izvršavaju, sklopovi za množenje postali su standarni dio svakog FPGA sklopa.

DCM blokovi se brinu o signalu radnog takta koji se razvodi u sve dijelove sklopa. Omogućuju fino podešavanje frekvencije, njeno množenje, dijeljenje, fazni pomak itd.



Slika 4. Spartan-3 FPGA [1]

Projekt

3.1 Uvod

Glavni dio ovog seminarskog rada temelji se na projektu koji će biti demonstriran u poglavljima koja slijede. Smatram da s ciljem razumijevanja, ljudi uvijek najviše nauče proučavajući primjere.

Isto tako želim istaknuti da je izrada ovog projekta bila najviše u mom interesu, jer sam time stekao nova znanja i iskustva, što bi mi bilo uskraćeno da sam se držao samo teoretskog dijela.

3.2 O Projektu

Projekt se temelji na konkretnoj realizaciji digitalnog sustava, u mom slučaju relativno jednostavnog, uporabom FPGA sklopa. Želim se zahvaliti svom mentoru dr.sc. Hrvoju Mlinariću što mi je ustupio FPGA razvojni sustav i tim činom omogućio izradu projekta.

Prezentirat ću FPGA sklop koji uz pomoć digitalnog temperaturnog senzora mjeri temperaturu koju zatim ispisuje na display.

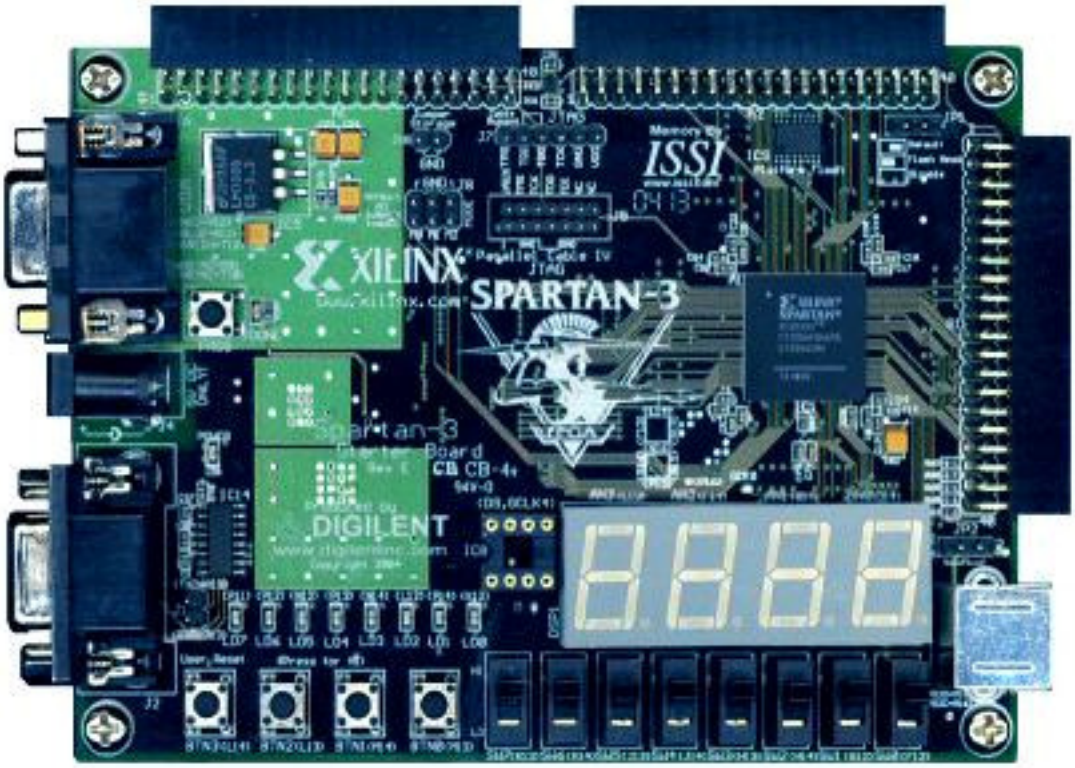
Potrebne komponente, u prvom redu Spartan-3 FPGA sklop i 7-segmenti prikaznik nalaze se na razvojnom sustavu. Jedino je temperaturni senzor dodan zasebno. Više o njima pročitajte u nastavku.

3.3 Razvojni sustav „Digilent Spartan-3 Starter Board”

Mnogi inženjeri koriste razvojne sustave pri projektiranju i programiranju digitalnih sustava. Najčešće ih izdaju na tržište matična poduzeća koja su vlasnici tehnologije na kojoj se sustavi baziraju. Primjer su brojni razvojni sustavi za mikrokontrolere (ARM, Atmel, MicroChip, Intel itd.), kao i oni temeljni na FPGA sklopovima. Razvojni sustav omogućuje dizajneru najbezbolnije testiranje i provjeru napisanog programskog koda. Inženjer ne mora gubiti vrijeme na slaganje hardvera jer mu je sve već servirano. Bitno je napomenuti da gotovi digitalni sustavi nikad ne izlaze na tržište prikovani za razvojni sustav. Razvojni sustavi služe samo i isključivo u fazi razvoja i testiranja i za tu svrhu su dizajnirani. To se posebno vidi po brojnim tipkalima, signalim LE diodama, relejima i sličnim komponentama koje su redovito spojene na sve izvode centralnog sklopa (npr. FPGA sklopa, mikrokontrolera, procesora itd.).

Digilent Spartan-3 razvojni sustav je „third party” sustav, dakle nije proizveden direktno od strane Xilinxa.

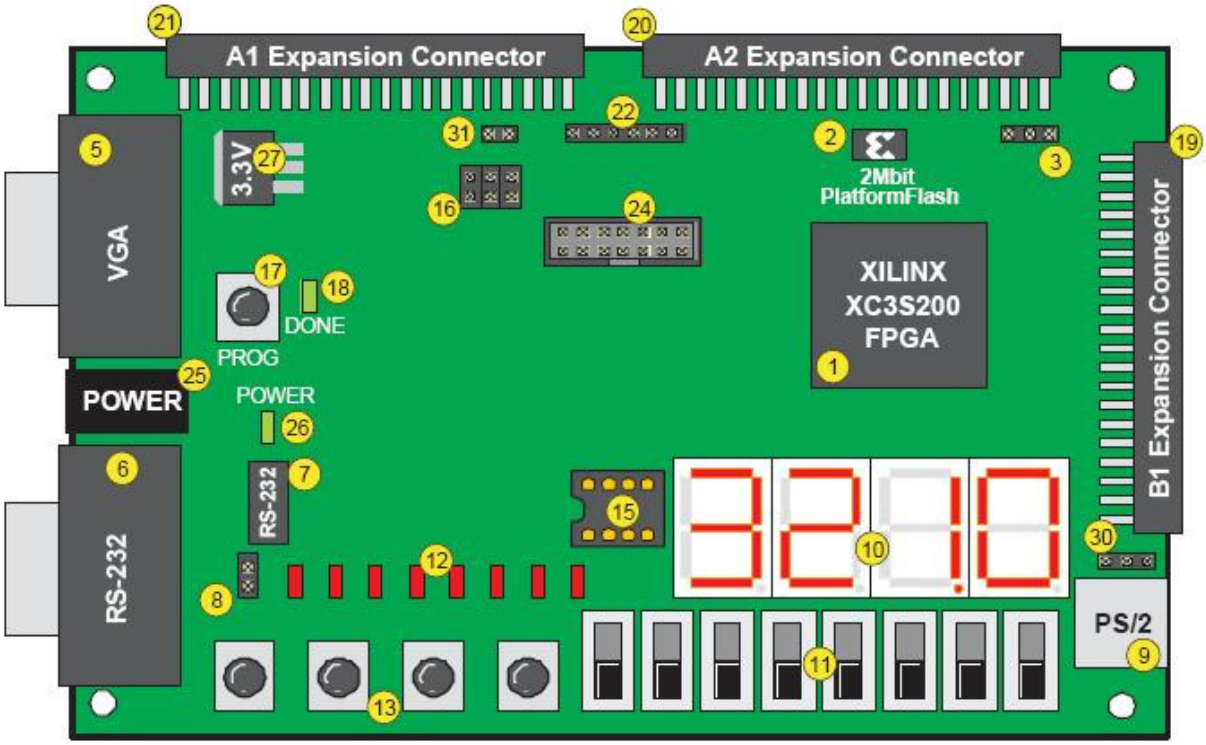
Pogledom na sljedeće slike dobit ćete najbolji uvid u dotični sustav.



Slika 5. Fizički izgled Digilent FPGA razvojnog sustava [3]

Kao što se vidi na slici *Digilent Spartan-3 Starter Board* je vrlo jednostavan sustav namijenjen većinom početnicima.

Pogledajmo neke osnovne komponente na sljedećoj slici.



Slika 6. Prikaz komponenti razvojnog sustava, gornja strana [3]

FPGA sklop prikazan je kao broj **1.** Njegovi izvodi razvedeni su po cijelom sustavu i spojeni na ostale komponente.

Xilinx Spartan-3 FPGA-ovi su **SRAM** (**S**tatic **RAM**) bazirani što znači da pri nestanku napajanja gube svoju konfiguraciju. Zbog toga postoji poseban sklop koji svaki put puni FPGA s konfiguracijskim podacima. On je prikazan kao «2Mbit PlatformFlash» i nalazi se pod brojem **2.**

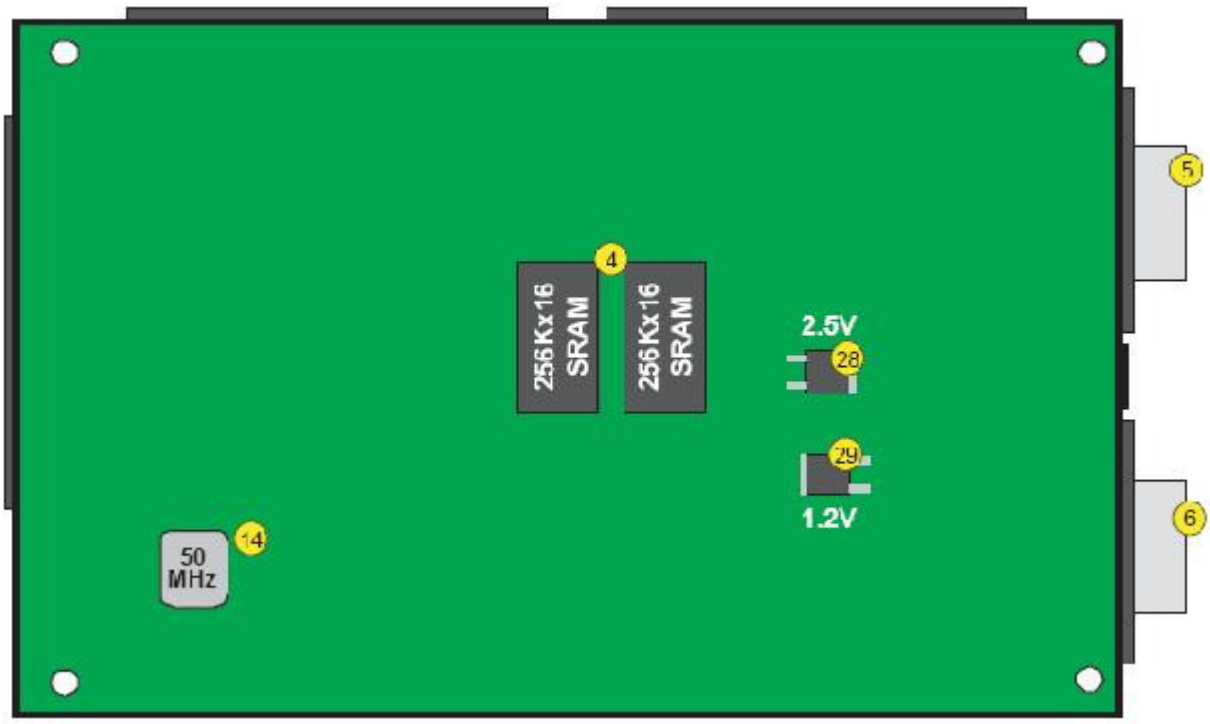
4 7-segmenta prikaznika nije teško prepoznati pod brojem **10.** Izvodi su im direktno povezani s FPGA sklopom. Više o načinu njihova rada pročitajte u poglavlju «7-segmentni prikaznici».

Signalne LE diode, tipkala i sklopke omogućuju komunikaciju, bilo korisnika prema FPGA sklopu ili obratno. Diode mogu poslužiti kao indikacija određenih stanja sklopa dok se na tipkala i sklopke mogu dovodit pobude na koje FPGA reagira. Lako ih je prepoznati pod brojevima redom **12, 13** i **11.**

Ploča također sadrži i RS-232 (serijski) konektor (**6**) zajedno s pripadnim RS-232 sklopom koji služi kao most između FPGA sklopa i npr. računala. Svrha mu je da uskladi naponske razine između FPGA sklopa i sustava s druge strane komunikacijskog kanala.

Pod brojem **6** nalazi se VGA konektor.

A1 (**21**), A2(**20**) i B1(**30**) su konektori za proširenja. Svi izvodi su im direktno spojeni na FPGA sklop. Korisnik na njih spaja sklopove po volji.



Slika 7. Prikaz komponenti razvojnog sustava, donja strana [3]

S druge strane **PCB**-a (**P**rinted **C**ircuit **B**oard) nalazi se izvor radnog takta (**14**) frekvencije 50MHz.

Ploča je također opremljena **RAM** (**R**andom **A**ccess **M**emory) memorijom (**4**) veličine 1MB.

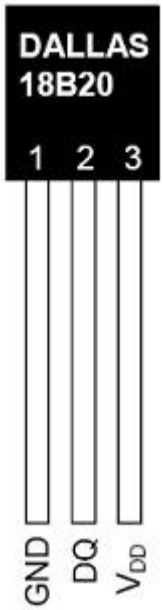
3.4 Digitalni temperaturni senzor - DS18B20

Zadaću mjerenja temperature preuzeo je *Dallas Semiconductor DS18B20 1-wire* temperaturni senzor. DS18B20 je digitalni senzor s programibilnom rezolucijom od 9-12 bita. Sva komunikacija između senzora i FPGA sklopa (u daljem tekstu ***master***) odvija se posebnim komunikacijskim protokolom koji koristi samo jednu liniju (**DQ**) za prijenos podataka.

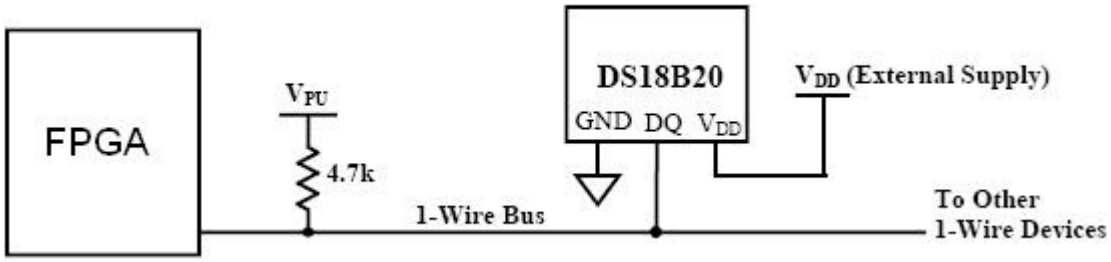
> Često se u literaturi spominju pojmovi *master* i *slave*. *Master je uređaj koji jedini može započet sabirničku transakciju. Slave uređaji samo osluškuju sabirnicu i imaju «pravo govora tek kad im master dopusti»*

Unatoč nazivu «1-wire» senzor nije moguće koristiti povezivanjem samo jedne linije DQ jer moramo dovesti adekvatno napajanje čemu služe izvodi GND i Vdd.

Krađom naboja s DQ linije u trenucima kada je ona *visoko* DS18B20 može raditi bez da se na Vdd izvod spoji napajanje. Takvo je napajanje Dallas nazvao «parazitno napajanje» U tom slučaju jedina 2 korištena izvoda su GND i DQ dok se Vdd također spaja na GND. Upitna je korisnost ove tehnologije, jer gdje postoji mogućnost povući dvije žice, postoji i tri, što je naravno mišljenje isključivo autora ovog seminara.
Shema spajanja senzora na FPGA sklop bez «parazitnog napajanja» prikazana je na slici 9. Jedina dodatna komponenta je otpornik koji služi za pritezanje linije na napajanje (visoko) kada sklopovi postave svoje izvode na koje je spojen DQ u stanje visoke impedancije.



Slika 8. DS28B20 [4]



Slika 9. Shema spajanja senzora na sustav [4]

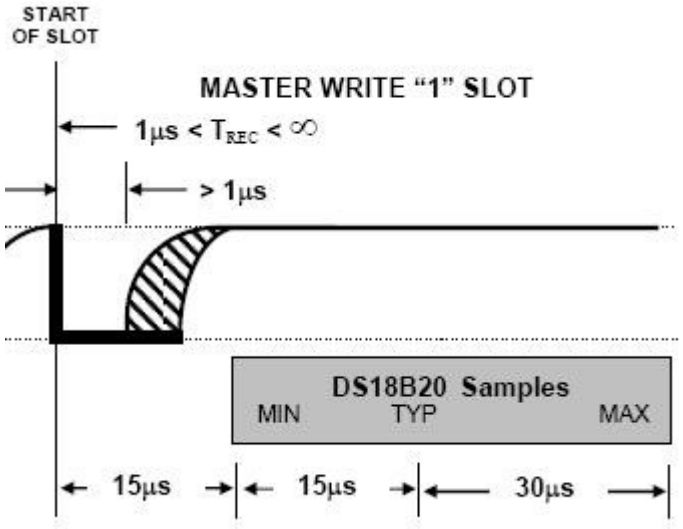
Dakle, komunikacija radi na način da oba uređaja mogu postavit liniju nisko slanjem logičke nule. Linija se postavlja visoko isključivo preko *pull-up* otpornika, kada uređaj koji šalje bit postavi izvod u stanje visoke impedancije. To se još naziva otpuštanjem linije.

Za prijenos bitova preko jedne komunikacijske linije definirani su ***time slotovi***. Oni predstavljaju skup pravila koji jednoznačno određuju koja se stanja na sabirnici moraju dogoditi da bi uređaj slao 0 ili 1, tj. čitao podatak.

Za primjer pogledajmo kako *master* može poslati senzoru logičku jedinicu, tj. jedan bit.

- *master* definira svoj DQ izvod kao izlazni i pošalje logičku nulu
- logička se nula (linija pritegnuta na masu) na sabirnici mora zadržali minimalno 1 µs.
- *master* otpušta liniju postavljanjem DQ izvoda u stanje visoke impedancije, nakon čega se ona samostalno preko pull-up otpornika priteže na napajanje
- 15 do 30 µs nakon što senzor detektira padajući brid signala na sabirnici, tj. nakon što je *master* poslao logičku nulu, senzor očitava stanje sabirnice koja je u našem slučaju visoko.
- *master* ostavlja sabirnicu visoko minimalno sljedećih 60 µs.

Sljedeća slika ilustrira gornji primjer.



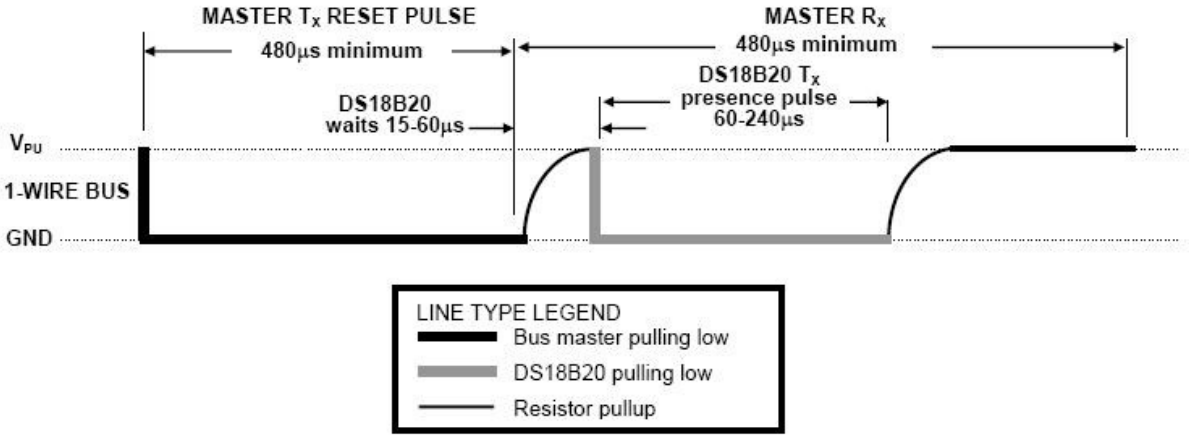
Slika 10. Pisanje logičke jedinice [4]

Za komunikaciju sa svim *1-wire* uređajima postoje dvije vrste naredbi koje *master*, u našem slučaju FPGA, šalje.

- **ROM** naredbe služe za adresiranje jednog uređaja na 1-wire sabirnici. Cijela je 1-wire tehnologija razvijena s ciljem da se omogući komunikacija teoretski beskonačno mnogo uređaja preko jedne jedine žice. U našem slučaju, to znači da FPGA sklop može koristeći samo jedan izvod mjeriti temperaturu na više mjesta. Svaki 1-wire uređaj ima jedinstveni 64-bitni ROM *serijski broj* koji ga razlikuje od svih drugih uređaja na sabirnici. U slučaju da se na sabirnici nalazi samo jedan uređaj, *master* koristi naredbu **Skip ROM** što znaci da 1-wire uređaj kao sljedeću naredbu očekuje funkcijsku naredbu.
- **Funkcijske** naredbe su specifične naredbe za svaki 1-wire uređaj. Tako naš temperaturi senzor ima naredbe tipa «***Convert T***» tj. pretvori temperaturu, pročitaj temperaturu, upiši konfiguracije podatke itd.

Za kraj bi još samo spomenuo da svaka nova komunikacija s senzorom, tj. općenito 1-wire uređajima započinje na način da *master* inicijalizira sve uređaje slanjem ***reset impusla*** nakon čega uređaji dojave svoju prisutnost šaljući ***presence pulse***. Reset impuls je impuls niske razine u trajanju od minimalno 500 µs.

Sljedeća slika ilustrira postupak inicijalizacije.



Slika 11. Postupak inicijalizacije [4]

3.5 Programski jezik za opis sklopovlja – VHDL

V*ery High Speed Integrated Circuit* ***H****ardware* ***D****escription* ***L****anguage* ili VHDL je poseban programski jezik za opis digitalnih sklopova. Razvijen je s ciljem razmjene podataka između različitih ugovornih stranki projekta radi pružanja nedvosmislenog zapisa za opis sklopovlja.

Upravo sam VHDL odabrao za opis ponašanja mog digitalnog sustava. Kod opisa FPGA sklopa jezikom VHDL bitno je napomenuti da se napisani kod često ne može prevesti u FPGA konfiguracijsku datoteku. Sam VHDL je vrlo općenit jezik i pri pisanju programa za FPGA sklop potrebno je znati mnoštvo dodatnih pravila unatoč poznavanja samog programskog jezika.

Osnovna struktura opisa sklopa sastoji se od:

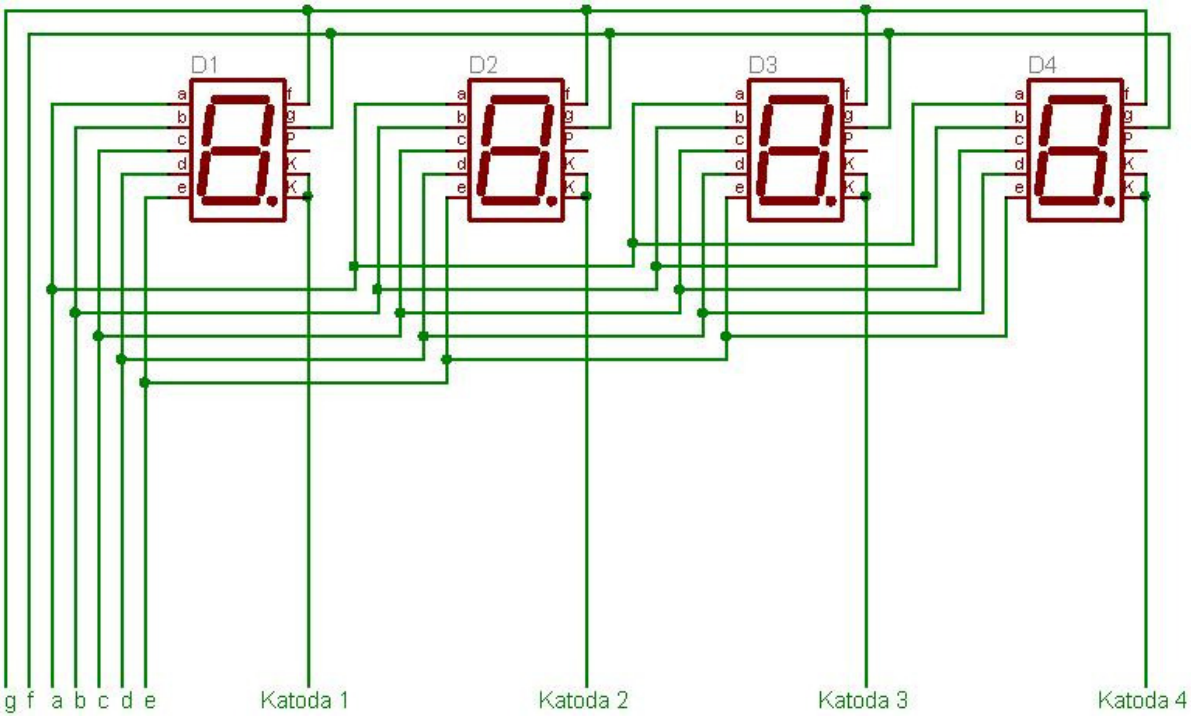
- **Entiteta**
  Entitet jednoznačno određuje prilaze sklopu. Drugim riječima, u njemu se definiraju svi izvodi. Svaki izvod mora imat definiran smjer (ulaz, izlaz ili ulaz-izlaz) kao i širinu tj, broj linija ako se radi o sabirnici.
- **Arhitekture**
  U arhitekturi se opisuje funkcija sklopa. Općenito, signali koji ulaze u sklop se obrađuju i prosljeđuju na izlaz.

Opisujući projekt neću ići u detalje jezika VHDL jer to bi prešlo granicu konteksta ovog seminara. Radi lakšeg razumijevanja čitatelju, fokus je na ideji, a ne samom programskom kodu.

3.6 7-segmentni prikaznici

Za ispis temperature iskoristio sam 7-segmente prikaznike koji se već nalaze na razvojnom sustavu. Slika se dobiva paljenjem, tj. gašenjem svakog nezavisnog segmenta zasebno. Prikaznik se u suštini sastoji od 7 LE dioda koje imaju zajedničku katodu, ili anodu ovisno o izboru. U slučaju zajedničke katode, segment se pali na način da se na anodu LE diode dovede logička jedinica, tj. napajanje dok se katoda uzemi. Na taj način svijetle samo oni segmenti čije diode imaju spojenu anodu na napajanje.

Na slici 12 prikazan je način spajanja 4 prikaznika u kaskadu. Uvijek se kod ispisa na više ovakvih prikaznika koristi metoda vremenskog multpleksiranja. Kao što se vidi sa slike anode istih segmenata na sva 4 prikaznika spojene su zajedno, dok su katode odvojene. Dakle, ukoliko želimo ispisati npr. broj „4” on će se ispisati na sva 4 prikaznika istovremeno ukoliko su sve katode spojene na masu. Kako onda ispisati podatak koji neće imati sva četiri znaka ista? Koristeći vremenski multipleks, program koji ispisuje podatak izvodi sljedeći algoritam.



Slika 12. 7-segmentni prikaznici u kaskadi

Za primjer uzmimo da želimo ispisati „1234”.

- Pošalji podatak koji odgovara ispisu broja „1”.
- Omogući ispis isključivo prvom prikazniku uzemljujući njegovu katodu. Ostala 3 prikaznika su ugašena jer im katode nisu uzemljene.
- Pošalji podatak koji odgovara ispisu broja „2”, no ovaj put omogući jedino drugi prikaznik dok su ostali ugašeni
- Znamenke 3 i 4 se ispisuju na isti način svaka na svoj prikaznik
- Izvršavaj to kružno minimalno 85 puta u sekundi.

Unatoč tome što su dobar dio vremena 3 od 4 prikaznika ugašena, čovjek koji gleda ispis ima dojam da je sve u redu. Algoritam koristi tromost ljudskog oka koje je presporo da primjeti brzo gašenje i paljenje prikaza.

Prednosti ovakog ispisa su očite. Za spajanje 4 7-segmenta prikaznika trebalo bi nam 32 izvoda (7*4 za anode + 4 zajedničke katode) FPGA sklopa naspram 11 koristeći vremenski multipleks. Osim smanjenja broja potrebnih izvoda, smanjena je i potrošnja energije jer je samo 1 prikaznik upaljen u određenom trenutku vremena.

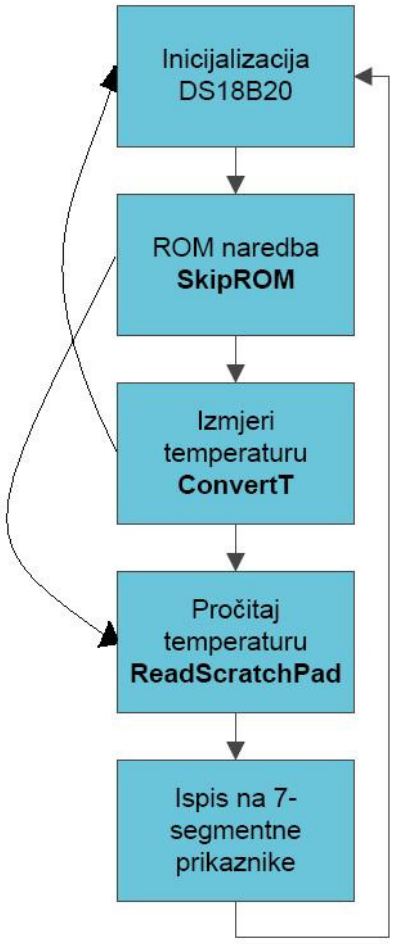
3.7 Realizacija sklopa

3.7.1 Algoritam

Kao što je već spomenuto u prošlim poglavljima, digitalni sustav u sklopu ovog seminara izvršavat će jednostavnu zadaću mjerenja temperature s 1-wire temperaturnog senzora. Temperatura će se ispisivati na 7-segmentne prikaznike nakon čega se cijeli postupak ponavlja.

Uvijek prije početka programiranja, inženjer mora znati osnovni algoritam preko kojeg će ostvarit program, tj. sklop u našem slučaju.

Algoritam mjerenja i ispisa temperature prikazan je na slici 13.



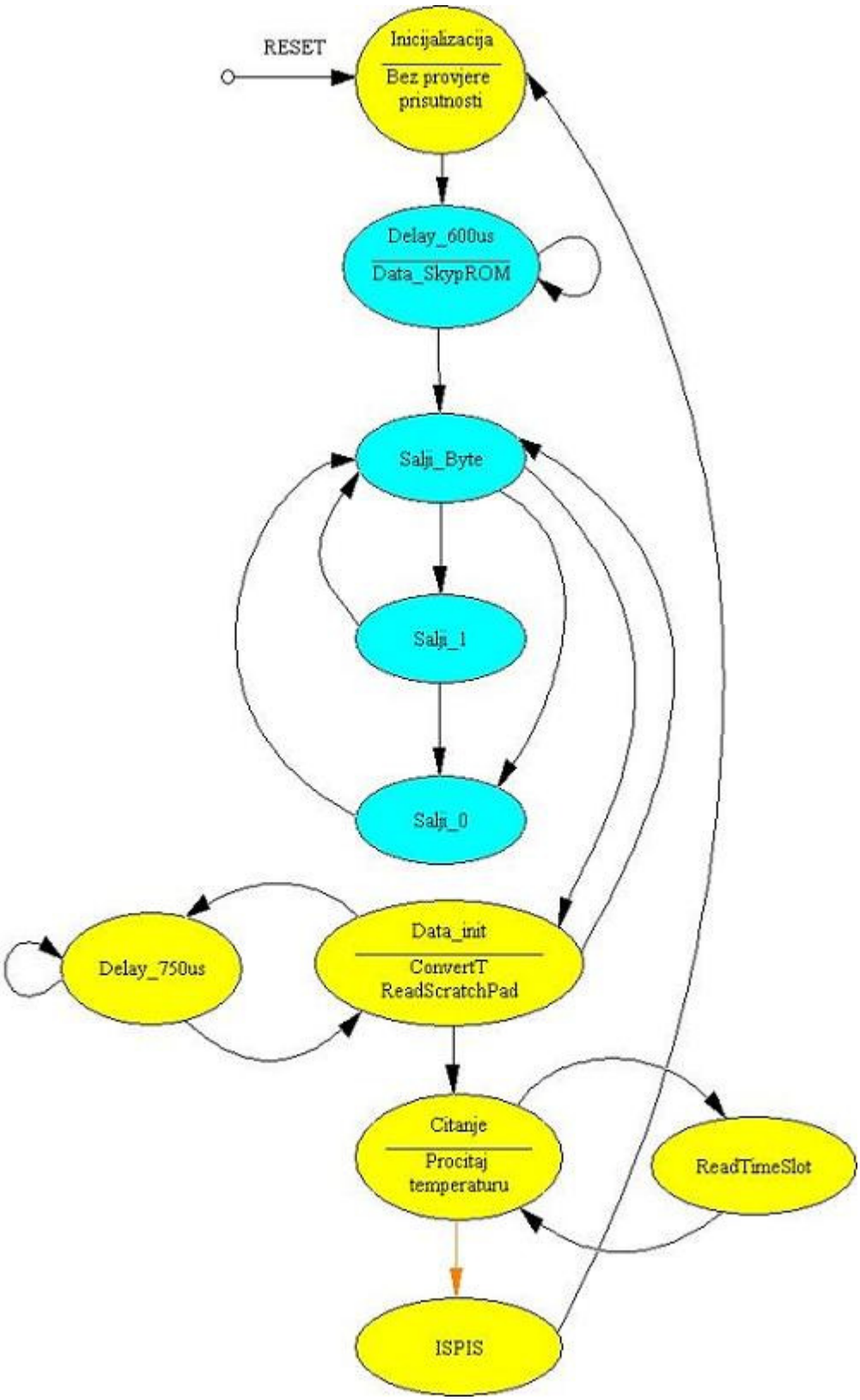
Ukratko, sastoji se od 5 koraka:

1. U početku, FPGA sklop mora inicijalizirati temperaturni senzor držeći liniju nisko minimalno 500 µs. Nakon toga senzor javlja svoju prisutnost šaljući *presence pulse*. Radi jednostavnosti, FPGA sklop neće provjeravati taj impuls. Pretpostavka je da je senzor spojen na sabirnicu i da radi ispravno.
2. Pošto je samo jedan 1-wire sklop spojen na sabirnicu nije potrebno adresirati poseban uređaj. Radi toga FPGA šalje **SkipROM** naredbu.
3. DS18B20 je adresiran i čeka naredbu od *mastera*. FPGA salje **ConvertT** nakon koje senzor započinje proces mjerenja temperature. Proces mjerenja temperature ne izvršava se trenutno već traje neko vrijeme. Za 12-bitnu rezoluciju to vrijeme iznosi maksimalno 750 ms. FPGA uređaj mora pričekati to vrijeme prije nego započne čitati temperaturu.
4. Temperatura se čita slanjem ReadScratchPad naredbe. Nakon što DS18B20 primi naredbu, *master* pokreće niz read time slotova tijekom kojih čita temperaturu bit po bit.
5. Izmjerena vrijednost temperature ispisuje se na 7-segmentne prikaznike.

Bitno je napomenut da je nakon svake funkcijske komande senzor potrebno inicijalizirati te poslat SkipROM. Zbog toga su na slici strelice povratka.

Da bi proveo gornji algoritam na FPGA sklopu uporabom VHDL jezika, odlučio sam napraviti automat s konačnim broj stanja, tj. **FSM** (**F**inite **S**tate **M**achine)

3.7.2 Automat s konačnim brojem stanja – FSM



Slika 14. Automat

Nakon razmišljanja o tome kako realizirati dotični sklop, automat s konačnim brojem stanja postavio se kao najjednostavnije rješenje. Načelna shema automata prikazana je na slici 14.

Zbog prevelikog broja stanja automat na slici 14 ne sadrži sva stanja korištena pri realizaciji. Npr. stanja «Salji_1, Salji_0, ReadTimeSlot itd.» sastoje se od više zasebnih stanja što nije prikazano. Razlog je tome što bi slika bila prevelika s velikim brojem prijelaza što bi je učinilo nečitljivom.

Osnova ideja iza realizacije automata je rastaviti algoritam u određeni broj malih i nezavisnih koraka. Sklop izvršavajući te korake izvršava svoju funkciju «dio po dio».

Pogledajmo kako to ostvarit programski u VHDL-u.

```
Process (CLKIN)
begin
     if CLKIN'event and CLKIN ='1' then
               case current_state is
                         when S0 => {skup naredbi}
                         when S0 => {skup naredbi}
                         when S0 => {skup naredbi}
                         …….
                         …….
                         when others =>
                                        next_state <= S0;
                 end case;
         end if;
end process;

--Reset i revaluacija trenutnog stanja
Process (CLKIN, RESET)
begin
      if (reset = '1') then
            current_state <= S0;
      elsif (CLKIN'event and CLKIN = '0') then
            current_state <= next_state;
      end if;
end Process;
```

Stanja se nalaze unutar procesa koji ima okidanje na svaku promjenu CLKIN signala, tj. radnog takta. Ako se pritom dogodio rastući brid signala, *case* naredba izvršava isključivo jedan **skup naredbi.** Koji će se skup naredbi, tj. koje stanje se izvršava ovisi o vrijednosti *current_state*.

Drugi proces služi za definiranje prijelaza stanja i reset. Također reagira na svaku promjenu radnog takta. Ako se pritom dogodio padajući brid CLKIN signala, buduće stanje postaje trenutno stanje koje prvi proces izvršava na sljedeći rastući brid.

Radi ilustracije pogledajmo detaljnije stanje **Salji_1** čija je osnova ideja opisana u poglavlju «*3.4 Digitalni temperaturni senzor - DS18B20*» na strani 11.

```
--Write_1
when S5 =>
          DQ <= '0';
          next_state <= S6;
--Delayus(1)
when S6 =>
          CTR <= CTR + "00000000000000000000000001";
          if (CTR = "00000000000000000000110010") then
             next_state <= S7;
             CTR <= "00000000000000000000000000";
          end if;
--Otpusti liniju
when S7 =>
          DQ <= 'Z';
          next_state <= S8;
--Delayus(60)-> END Write_1
when S8 =>
          CTR <= CTR + "00000000000000000000000001";
          if (CTR = "00000000000000101110111000") then
             next_state <= S4;
             SendDATA <= SendDATA ror 1;
             done <= done + "0001";
             CTR <= "00000000000000000000000000";
          end if;
```

Slanje logičke jedinice rastavljeno je u 4 koraka, tj. 4 stanja. Ona su označena kao S5, S6, S7 i S8.

- U stanju S5 FPGA postavlja liniju nisko nakon čega odmah prelazi u sljedeće stanje. Pod pojmom odmah podrazumijeva se da se to događa tek na sljedeći rastući brid takta. Dakle, stanja se izvršavaju na rastući brid, nova vrijednost trenutnog stanja upisuje se na padajući brid, dok se to novo stanje može najranije izvršiti tek na rastući brid koji slijedi.
- S6 je stanje čekanja od 1 µs. Izvedeno je na način da stanje samo sebe poziva velik broj puta uvećavajuci signal CTR svaki put za jedan. Kada CTR dostigne vrijednost navedu u *if* naredbi, automat prelazi u sljedeće stanje i vraća vrijednost CTR signala na nula. Bitno je napomenuti da se sve naredbe unutar nekog bloka naredbi, u našem slučaju unutar if-a izvršavaju paralelno!
- U stanju S7 FPGA otpušta liniju postavljajući ju u stanje visoke impedancije
- Svaki **time slot** traje minimalno 60 µs. 15 do 30 µs nakon što je FPGA otpustio liniju, DS18B20 pročitati će logičku jedinicu sa sabirnice pošto je ona pritegnuta na napajanje. Kada *tajmer* odbroji dotično vrijeme automat se vraća u stanje S4 koje je centralno stanje za slanje podataka. Dodatne dvije naredbe rotacije (ror) i uvećanja za jedan služe kao priprema za slanje sljedećeg bita.

Da bi razumjeli kako se šalje byte podataka pogledajmo detaljnije stanje S4.

```
when S4 =>
          --Ako je poslano svih 8 bita
          if done = "1000" then
             done <= "0000";
             case SendDATA is
                   when "11001100" =>
                         next_state <= S13; --Idi na ConvertT
                   when "01000100" =>
                         next_state <= S14; --Idi na WaitToConvertT
                   when "10111110" =>
                         next_state <= S16; --Citaj temp !
                   when others =>
                         next_state <= ERROR;
             end case;
          elsif ((SendDATA and "00000001") = "00000001") then
             next_state <= S5;  --šalji 1
          elsif ((SendDATA and "00000001") = "00000000") then
             next_state <= S9;  --šalji 0
          end if;
```

SendDATA sadrži byte podataka koji se šalje, dok *done* signalizira da je poslano svih 8-bita. Stanje S4 ovisno o bitu najmanje težine prelazi u stanje S5, tj. S9 u slučaju slanja logičke nule. Ta dva stanja brinu se da na kraju povećaju done za jedan i rotiraju SendDATA desno za jedan bit pripremivši tako sve za slanje sljedećeg bita. Kada vrijednost done postane 8 slanje byte-a je završeno. Pošto se SendDATA nakon 8 rotacija vrati na početnu vrijednost, automat nakon prijenosa byte-a točno zna u koje se stanje mora vratiti ispitujući podatak u SendDATA.

Čitanje temperature izvedeno je na sličan način. Jedno centralno stanje poziva više puta stanja koja oponašaju **Read Time Slot.**

Radi veće preglednosti seminarskog rada cijeli kod nalazi se na kraju u poglavlju «Cijeli programski kod»

Sažetak

Seminarski rad se dotiče teme programibilnih logičkih sklopova. Programibilni logički sklopovi su elektronički sklopovi koji se koriste za izradu digitalnih sklopova i sustava.

U uvodu je ukratko objašnjena osnova ideja koja stoji iza tih sklopova. U sljedećem poglavlju opisani su prvi programibilni sklopovi napravljeni od ROM memorija. Takvi sklopovi imaju brojne nedostatke čije su mane ispravljene dolaskom PAL i PLA sklopova na tržište.

Sljedeća generacija programibilnih sklopova dolazi pod nazivom Complex PLD čija struktura obuhvaća više jednostavnih PAL i PLA sklopova u jednu cjelinu.
Najmodernijim programibilnim sklopovima današnjice FPGA, posvećena je posebna pažnja. Objašnjena je njihova osnovna struktura, način rada i primjena.

Kao najbolji način prezentacije programibilnih sklopova u ostatku seminara predstavljena je realizacija sklopa koji mjeri i ispisuje temperaturu uporabom FPGA tehnologije.

Opisan je razvojni sustav s FPGA sklopom kao i digitalni temperaturi senzor DS18B220.
Posebno je poglavlje posvećeno ispisu teperature na 7-segmente prikaznike gdje se detaljno objašnjava njihov način rada.

Umjesto detaljnog opisa VHDL programskog koda autor je pažnju prenio na osnovu ideju realizacije sklopa. VHDL se ukratko objašnjava u zasebnom poglavlju.

Sklop je realiziran uporabom automata s konačnim brojem stanja. Čitatelj ima priliku vidjeti kako efikasno realizirati FSM u VHDLu. Detaljno je objašnjen način rada konačnog sklopa.

Zainteresirani čitatelji koji žele znati više mogu proučiti cijeli VHDL kod priložen na kraju seminara kao i korištenu literaturu.

Zaključak

Od pojave programibilnih sklopova 70'-ih godina prošlog stoljeća bitno se promijenio način na koji poduzeća proizvode digitalne sustave. Uveliko se skratilo vrijeme razvoja, tj. vrijeme proteklo od sklopa na papiru do sklopa na *polici dućana.*

Osim toga, ASIC sklopovi danas nisu više jedina mogućnost tehnologije izrade. Alternativa su im programibilni sklopovi koji se koriste u svim područjima industrije, od proizvodnje do zabavne elektronike. Sve je to zadnjih 30-ak godina utjecalo na konstantno snižavanje cijena potrošačke elektronike koje danas imamo u izobolju.

Uostalom, programibilni sklopovi su omogućili i pojedincima poput mene, da u miru svog doma razviju svoj vlastiti digitalni sklop.

Cijeli programski kod

```
----------------------------------------
-- Napravio: Ivan Dodig
-- Naziv projekta: Temperaturna stanica
-- Datum pocetka: 01.05.2006
-- Datum završetka: NA
-- Status: Radi!
----------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity DS18B20 is
    Port (
           CLKIN : in  STD_LOGIC;
                     RESET : in  STD_LOGIC;
                     DQ    : inout  STD_LOGIC;
                     LED   : inout  STD_LOGIC_VECTOR (7 downto 0);
           SEG   : out  STD_LOGIC_VECTOR (6 downto 0);
                     AN0   : inout  STD_LOGIC;
           AN1   : inout  STD_LOGIC;
           AN2   : inout  STD_LOGIC;
                     AN3   : inout  STD_LOGIC);
end DS18B20;

architecture Behavioral of DS18B20 is
--Definiranje signala
signal CTR : std_logic_vector (25 downto 0) := "00000000000000000000000000";
signal CTR2: std_logic_vector (12 downto 0) := "0000000000000";
signal done: std_logic_vector ( 3 downto 0) := "0000";
signal SendDATA: bit_vector (7 downto 0) := "00000000";
signal TEMP: bit_vector (11 downto 0) := "000000000000";
signal DIGIT: bit_vector (3 downto 0) := "0000";
signal UNIT: bit_vector (3 downto 0) := "0000";
signal DIGIT_I: bit_vector (3 downto 0) := "0000";
signal UNIT_I: bit_vector (3 downto 0) := "0000";
signal DSRead : bit;
signal status : bit := '0';
--FSM deklaracija
type state_type is
(S0,S1,S2,S3,S4,S5,S6,S7,S8,S9,S10,S11,S12,S13,S14,S15,S16,S17,S18,S19,S20,S21,S22,S
23,S24,S25,S26,S27,S28,S29,ERROR);
signal next_state,current_state : state_type;
begin
            --FSM DS18B20 komunikacija
STANJA:     Process (CLKIN)
      begin
      if CLKIN'event and CLKIN ='1' then
                  case current_state is
            --1_wire inicijalizacija
            when S0 =>
                  UNIT <= "0000";
                  DIGIT <= "0000";
                  DQ <= '0';
                  next_state <= S1;
            --Delayus(500)
            when S1 =>
                  CTR <= CTR + "00000000000000000000000001";
                  if (CTR = "00000000000110000110101000") then
                        next_state <= S2;
                        CTR <= "00000000000000000000000000";
                  end if;
            --Otpusti liniju
```

```
when S2 =>
       DQ <= 'Z';
       next_state <= S3;
--Delayus(600) -> END inicijalizacija
when S3 =>
 CTR <= CTR + "00000000000000000000000001";
       if (CTR = "00000000000111010100110000") then
              next_state <= S4;
              CTR <= "00000000000000000000000000";
              SendDATA <= "11001100"; --OxCC SkipROM
       end if;
--SkipROM !!
when S4 =>
--Ako je poslano svih 8 bita
if done = "1000" then
       done <= "0000";
       case SendDATA is
       when "11001100" =>
              if status = '0' then
                     next_state <= S13; --Idi na ConvertT
              elsif status = '1' then
                     next_state <= S15;
                     status <= '0';
              end if;
       when "01000100" =>
              next_state <= S14; --Idi na WaitToConvertT
       when "10111110" =>
              next_state <= S16; --Citaj temp !
       when others =>
              next_state <= ERROR;
       end case;
--SkipROM(0) Jel mogu tak umjesto ovoga ? Pitat H.M.
elsif ((SendDATA and "00000001") = "00000001") then
       next_state <= S5;  --salji 1
elsif ((SendDATA and "00000001") = "00000000") then
       next_state <= S9;  --salji 0
end if;
--Write_1
when S5 =>
       DQ <= '0';
       next_state <= S6;
--Delayus(1)
when S6 =>
       CTR <= CTR + "00000000000000000000000001";
       if (CTR = "00000000000000000000110010") then
              next_state <= S7;
              CTR <= "00000000000000000000000000";
       end if;
--Otpusti liniju
when S7 =>
       DQ <= 'Z';
       next_state <= S8;
--Delayus(60)-> END Write_1
when S8 =>
       CTR <= CTR + "00000000000000000000000001";
       if (CTR = "00000000000000101110111000") then
              next_state <= S4;
              SendDATA <= SendDATA ror 1;
              done <= done + "0001";
              CTR <= "00000000000000000000000000";
       end if;
--Write_0
when S9 =>
       DQ <= '0';
       next_state <= S10;
--Delayus(60)
```

```
when S10 =>
       CTR <= CTR + "00000000000000000000000001";
       if (CTR = "00000000000000101110111000") then
              next_state <= S11;
              CTR <= "00000000000000000000000000";
       end if;
--Otpusti liniju
when S11 =>
       DQ <= 'Z';
       next_state <= S12;
--Delayus(1) END Write_0
when S12 =>
       CTR <= CTR + "00000000000000000000000001";
       if (CTR = "00000000000000000000110010") then
              next_state <= S4;
              SendDATA <= SendDATA ror 1;
              done <= done + "0001";
              CTR <= "00000000000000000000000000";
       end if;
--Convert TEMP
when S13 =>

       SendDATA <= "01000100";  --0x44
       next_state <= S4;
--WaitToConvertT (samo veliki delay 1sec bez prozivanja)
when S14 =>
       CTR <= CTR + "00000000000000000000000001";
       if (CTR = "10111110101111000010000000") then
              next_state <= S0;
              CTR <= "00000000000000000000000000";
              status <= '1';
       end if;
--Read ScratchPAD / TEMP
when S15 =>
       SendDATA <= "10111110"; --0xBE
       next_state <= S4;
--Read Centralno stanje
when S16 =>
       if done = "1100" then
              done <= "0000";
              next_state <= S23;  --Pripremi ispis
              --Tu TEMP pripremit za ispit
              TEMP <= TEMP srl 4;
       else
              TEMP <= TEMP ror 1;
              next_state <= S17;
       end if;
--ReadTimeSlot START
--BUS_LOW
when S17 =>
       DSRead <= '0';
       DQ <= '0';
       next_state <= S18;
--Delayus(1)
when S18 =>
       CTR <= CTR + "00000000000000000000000001";
       if (CTR = "00000000000000000000110010") then
              next_state <= S19;
              CTR <= "00000000000000000000000000";
       end if;
--Otpusti liniju
when S19 =>

                     DQ <= 'Z';
                     next_state <= S20;
--delayus(5) da se pojavi valjani podatak
```

```
when S20 =>
       CTR <= CTR + "00000000000000000000000001";
       if (CTR = "00000000000000000011111010") then
              next_state <= S21;
              CTR <= "00000000000000000000000000";
       end if;
--Procitaj bit
when S21 =>
       if DQ = '1' then
              DSRead <= '1';
       end if;
       next_state <= S22;

--delayus(60) Pricekaj kraj TimeSlota, Upisi bit u temp i vrati se
when S22 =>
       CTR <= CTR + "00000000000000000000000001";
       if (CTR = "00000000000000101110111000") then
              next_state <= S16;
              CTR <= "00000000000000000000000000";
              TEMP(11) <= DSRead;
              done <= done + "0001";
       end if;
when S23 =>
       if done = "1000" then
              done <= "0000";
              next_state <= S0;
              DIGIT_I <= DIGIT;
              UNIT_I <=  UNIT;
       else
              next_state <= S24;
       end if;

when S24 =>
       DIGIT <= DIGIT sll 1;
       next_state <= S25;

when S25 =>
       DIGIT(0) <= UNIT(3);
       next_state <= S26;

when S26 =>
       UNIT <= UNIT sll 1;
       next_state <= S27;

when S27 =>
       UNIT(0) <= TEMP(7);
       next_state <= S28;

when S28 =>
       if ((done xor "0111") /= "0000") then
              case UNIT is
                     when "0101" => UNIT <= "1000";
                     when "0110" => UNIT <= "1001";
                     when "0111" => UNIT <= "1010";
                     when "1000" => UNIT <= "1011";
                     when "1001" => UNIT <= "1100";
                     when others => UNIT <= UNIT;
              end case;
       end if;
       TEMP <= TEMP sll 1;
       next_state <= S23;
       done <= done + "0001";

when ERROR =>
                     --LED <= "11100000";
                     next_state <= ERROR;
```

```
        --Unknown state
        when others =>
                                    next_state <= ERROR;
        end case;
    end if;
    end process;

    --Reset i revaluacija trenutnog stanja
    Process(CLKIN, RESET)
    begin
        if (reset = '1') then
            current_state <= S0;
        elsif (CLKIN'event and CLKIN = '0') then
            current_state <= next_state;
        end if;
    end Process;

    --Ispis na 7seg display
Process (CLKIN)
begin
     if CLKIN'event and CLKIN = '1' then
    if (CTR2 = "0000000000000") then
      if (AN0='0') then
        AN0 <= '1';
            if UNIT_I = "0000" then
                SEG <= "1000000"; --0
            elsif UNIT_I = "0001" then
                SEG <= "1111001"; --1
            elsif UNIT_I = "0010" then
                SEG <= "0100100"; --2
            elsif UNIT_I = "0011" then
                SEG <= "0110000"; --3
            elsif UNIT_I = "0100" then
                SEG <= "0011001"; --4
            elsif UNIT_I = "0101" then
                SEG <= "0010010"; --5
            elsif UNIT_I = "0110" then
                SEG <= "0000010"; --6
            elsif UNIT_I = "0111" then
                SEG <= "1111000"; --7
            elsif UNIT_I = "1000" then
                SEG <= "0000000"; --8
            elsif UNIT_I = "1001" then
                SEG <= "0010000"; --9
            else
                SEG <= "0001110"; --F
            end if;
             AN2 <= '0';

         elsif (AN2='0') then
       AN2 <= '1';
             if DIGIT_I = "0000" then
                SEG <= "1000000"; --0
            elsif DIGIT_I = "0001" then
                SEG <= "1111001"; --1
            elsif DIGIT_I = "0010" then
                SEG <= "0100100"; --2
            elsif DIGIT_I = "0011" then
                SEG <= "0110000"; --3
            elsif DIGIT_I = "0100" then
                SEG <= "0011001"; --4
            elsif DIGIT_I = "0101" then
                SEG <= "0010010"; --5
            elsif DIGIT_I = "0110" then
                SEG <= "0000010"; --6
```

```
                    elsif DIGIT_I = "0111" then
                          SEG <= "1111000"; --7
                    elsif DIGIT_I = "1000" then
                          SEG <= "0000000"; --8
                    elsif DIGIT_I = "1001" then
                          SEG <= "0010000"; --9
                    else
                          SEG <= "0001110"; --F
                    end if;
                     AN3 <= '0';

               elsif (AN3='0') then
                     AN3 <= '1';
                     SEG <= "1000110";   --C
                     AN0 <= '0';
               end if;
      end if;
      CTR2 <= CTR2 + "0000000000001";
      if (CTR2 > "1000000000000") then
        CTR2 <= "0000000000000";
      end if;
    end if;
  End Process;

end Behavioral;
```

Literatura

[1] Uroš Peruško, Vlado Glavinić : Digitalni sustavi
Školska knjiga, Zagreb, 2005

[2] Xilinx : Spartan-3 Complete Data Sheet
http://www.xilinx.com

[3] Digilent Inc: Spartan-3 Starter Board Reference Manual
http://www.digilentinc.com/Products

[4] Dallas Semiconductor : DS18B20 Data sheet
http://www.maxim-ic.com/