



PROJEKTOVANJE SISTEMA NA ČIPU

Programiranje FPGA korištenjem Veriloga

prof. dr. Lejla
Banjanović-Mehmedović



Sadržaj izlaganja

- Jezici za programiranje FPGA
- HDL: Verilog
- Primjeri Verilog aplikacija u automatici i robotici
- Eksperimentalni dizajn FPGA baziranih rješenja
 - Primjer regulacija temperature (pametne kuće)
 - Primjer heksapoda
 - Primjer praćenja linije mobilnog robota

Kako funkcionira FPGA programiranje?

FPGA uređaji čijim se fizičkim atributima može manipulirati upotrebom jezika za opis hardvera (HDL) - prevazilaze jaz između softvera za programiranje i hardvera za programiranje.

FPGA programiranje koristi HDL za upravljanje sklopovima.

HDL čini fizičke promjene na hardveru, umjesto da striktno optimizira uređaj za pokretanje softvera.

Budući da se manipuliše hardverom od osnova, FPGA dopuštaju veliku fleksibilnost - prilagoditi osnovne funkcije kao što su memorija ili potrošnja energije ovisno o zadatku.

Jezici koji se koriste za programiranje FPGA

- HDL-ovi koji se danas prvenstveno koriste u FPGA programiranju:
 - **Lucid** (the Dataflow Programming Language)
 - **Verilog**
 - **VHDL**
- Jezici koji se mogu koristiti s objedinjenim softverskim platformama za programiranje FPGA uključuju:
 - **AI framework poput TensorFlow i Pytorch**
 - **C and C++**
 - **Python**

<https://www.icdrex.com/4-best-fpga-programming-languages-for-beginners/>

HDL

- U računarskom inžinjerstvu, HDL je specijalizirani računarski jezik koji se koristi za opisivanje strukture i ponašanja elektroničkih sklopova, a najčešće digitalnih logičkih kola.
- Također omogućuva sintezu HDL deskripcije (opisa) u net-listu (specifikacija elektroničkih komponenti i načina njihove međusobne interkonekcije), koji se zatim mogu staviti i usmjeriti za proizvodnju seta maski koje se koriste za kreiranje integrisanih sklopova.
- Koriste se za opis ponašanja (*eng. behaviour form*)
- Bitna razlika između većine programskih jezika i HDL-a je da **HDL eksplicitno uključuje pojam vremena.**
- **HDL: Verilog, VHDL, System Verilog**

Verilog vs. VHDL

- **Modelovanje hardverske strukture postiže se jednostavnije sa Verilog-om nego sa VHDL-om**, posebno za programabilni logički interfejs (PLI).
- **Vrste podataka:** U usporedbi s VHDL-om tipovi podataka Verilog-a su vrlo jednostavni, lagani za korištenje i u velikoj mjeri usmjereni prema modeliranju hardverske strukture za razliku od abstraktnog hardverskog modeliranja s VHDL-om.
- **Ponovna upotreba dizajna:** S VHDL-om procedure i funkcije mogu se smjestiti u paket tako da su dostupni bilo kojoj dizajnerskoj jedinici koja ih želi koristiti, dok s Verilog-om nema paketa jer su funkcije i postupci definisani u modulu.
- **Lakše za naučiti:** Verilog je lakše naučiti i razumjeti za razliku od VHDL koji je "strongly typed" u svom jezičnom korištenju.

Verilog

- Verilog - hardverski programski jezik za projektovanje digitalnih sistema na različitim nivoima apstrakcije:
 - Behavioral Level
 - Register-Transfer Level
 - Gate Level

Ponašajna apstrakcija

- Kod sličan C programskom jeziku; svaki algoritam je sekvencijalan, tj. sastoji se od niza instrukcija koje se izvršavaju jedna za drugom.
- Verilog opisuje sistem konkurentnim algoritmima (**Behavioral**), primjer skevencijalnih kola - u registru se dešava tranzicija pri rastućoj ili opadajućoj vrijednosti klok signala.
- Osnovni elementi:
 - Functions (funkcije)
 - Tasks (zadaci)
 - Always blokovi.
- Malo struktturnih detalja (izuzev interkonekcija modula)

Register-Transfer Level (RTL)

- Specificira karakteristike kola pomoću niza operacija i transfera podataka između registara.
- Koristi se tačno definisan sat (RTL projektovanje sadrži tačno definisano izvršavanje operacija u određenom trenutku).

Gate Level

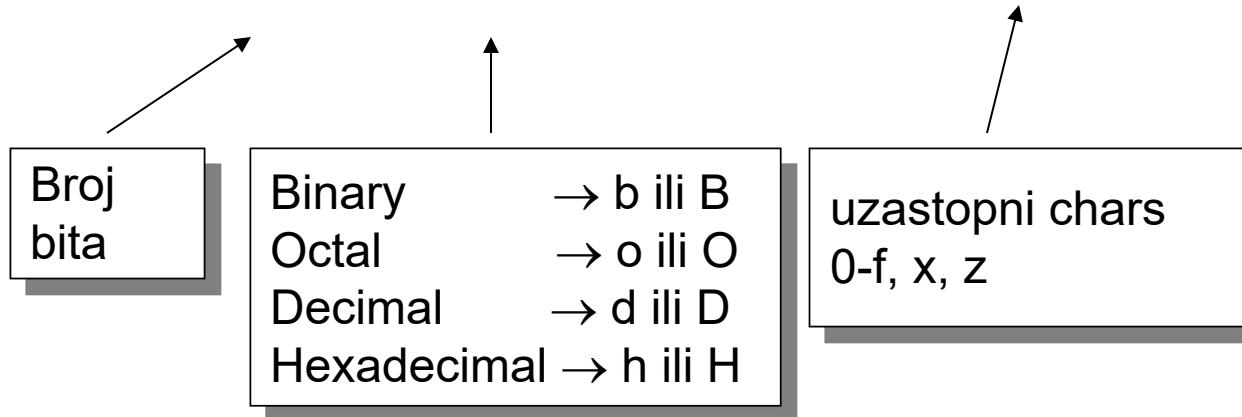
- U sklopu logičkog dizajna, karakteristike sistema su opisane logičkim linkovima i vremenskim odredbama.
 - **Interkonekcija jednostavnih komponenti**
 - **Potpuno struktuiran**
- Svi signali su diskretni (uzimaju vrijednosti iz diskretnog skupa logičkih vrijednosti):
 - '0': zero (logička nula) ili false (ground),
 - '1': one (logička jedinica) ili true
 - 'X': nepoznata logička vrijednost,
 - 'Z' : vrijednost visoke impedance (unconnected, tri-state)
- Operacije koje se mogu koristiti su:
 - AND, OR, NOT.

Leksička pravila

- Leksička pravila Verilog-a su slična pravilima C++ programskog jezika:
 - Ključne riječi se pišu malim slovima.
 - Verilog razlikuje mala i velika slova.
 - Imena varijabli mogu sadržavati bilo koju sekvencu slova, cifara, znakove ‘\$’ i _.
 - Ista pravila vrijede i za identifikatore (imenuju objekte poput modula ili registara).
 - Imena varijabli i identifikatori ne mogu početi sa ‘\$’, ne mogu u sebi imati ‘-’, zagrade ili ‘#’; mogu početi sa slovom ili _; maksimalna dužina 1024 karaktera.

Brojevi u Verilogu

<veličina>'<brojni sistem> <vrijednost>



- $8'h\ ax = 1010xxxx$
- $12'o\ 3zx7 = 011zzzxxxx111$

Elementi Veriloga – tipovi podataka

- **Wire** – predstavljaju fizičku vezu između komponenti.
 - **Wire varijabla ne čuva nikakvu vrijednost.**
 - Wire - samo jedan od podtipova net tipa podataka u koji još spadaju: „wired and (wand)”, „wired or (wor)” i “tristate bus (tri)”.
 - Svaki od net tipova ima određenu funkcionalnost koja služi da se modeliraju različiti tipovi hardware-a.
-
- **Deklaracija wire**
`wire [<range>] <net_name>;`
Opseg specificiran kao [MSb:LSb]. Default: širina 1 bit

 - **Registri (reg)** – predstavljaju varijable u kojima se čuvaju podaci i uporedivi su sa varijablama u programskim jezicima. Deklaracija se vrši pomoću ključne riječi reg. Registri čuvaju zadnju vrijednost koja im je pridružena, sve dok im neki drugi izraz ne promijeni vrijednost
 - **Deklaracija register**
`reg [<range>] <reg_name>;`

Elementi Veriloga – tipovi podataka

- Reg i Wire mogu imati sljedeće vrijednosti:
 - 0 logička nula ili ‘false’
 - 1 logička jedinica ili ‘true’
 - x nepoznata logička vrijednost
 - z vrijednost visoke impedanse
- U Verilog-u nije dozvoljena upotreba višedimenzionalnih nizova.
- Na početku simulacije reg varijabla je inicijalizirana na x, dok je varijabla wire inicijalizirana na z.

Elementi Veriloga – tipovi podataka

- Reg i wire varijable su po default-u skalarne (npr. jedan bit), ali ih možemo definisati i kao niz bita:
 - Deklaracija 4 bit wire, indeks starta od 0:
`wire [3:0] w;`
 - Deklaracija 8 bit registra:
`reg [7:0] r;`
 - Deklaracija 32 elementa memorije 8 bita širine:
`reg [7:0] mem [0:31]`
 - Ekstrakcija određenih bita:
`r[5:2]`
- **logic** se može koristiti umjesto reg ili wire
- **Parameters** nisu varijable, već su konstante: parameter foo = 42

Apstraktni tipovi podataka

- Zbog potreba korisnika u Verilog-u postoje i neki dodatni tipovi podataka koji nemaju odgovarajuću realizaciju u hardveru. Ovi tipovi uključuju **integer**, **real** i **time**.
- Integer i real imaju iste karakteristike kao i u drugim programskim jezicima (npr. C-u).
- Bitno je napomenuti da je **reg** varijabla neoznačena, dok je **integer** označeni 32-bitni cijeli broj.
- Varijabla **time** sadrži 64-bitnu vrijednost koja se koristi zajedno sa **sistemskom funkcijom \$time**

Cijeli brojevi

- Za cijele brojeve moguće je koristiti i deklaraciju reg ali je uobičajeno da se podatak tipa integer koristi za potrebe brojanja.
- Podrazumjavana širina podatka integer zavisi od širine riječi računara na kome se vrši simulacija, minimalna širina je 32 bita.
- Nosioc podatka tipa reg pamti podatke kao pozitivne cijele brojeve, dok nosioc podatka tipa integer može da pamti i negativne brojeve.

Realni brojevi

- Realni brojevi su takođe **podaci regalarskog tipa** u Verilog HDL-u. Deklaracija se vrši pomoću ključne riječi **real**.
- Vrijednosti podatka real se mogu zadati u decimalnom obliku (tj. 3.14) ili u eksponencijalnom obliku (tj. 3e2 = 300). Treba obratiti pažnju da se **za odvajanje razlomljenog dijela broja u Verilog HDL-u se koristi tačka umjesto zareza**.
- **Veličinu nosioca podatka tipa real zavisi od računara na kome se vrši simulacija.** Podrazumjevana vrijednost podatka real pri deklarisanju je 0. U slučaju dodjele vrijednosti nosiocu podatka real nosiocu integer vrši se zaokruživanje na najблиži cijeli broj.

Nizovi

- **Verilog HDL podržava građenje jednodimenzionalnih nizova** (eng. array) od tipova podataka **reg**, **integer** i **vektora registara**.
- **Nizovi se ne mogu praviti od podataka tipa real i ne mogu se praviti višedimenzionalni nizovi.**
- Deklaracija nizova se vrši u sljedećoj formi:
- <ime_niza>[veći_broj:manji_broj] ili
<ime_niza>[manji_broj:veći_broj]

Operatori u Verilogu

Tip operatora	Simbol operatora	Operacija	Broj operanada
Aritmetički	*	množenje	2
	/	dijeljenje	2
	+	sabiranje oduzimanje	2
	-		2
	%	dijeljenje po modulu	2
Logički	!	logička negacija	1
	&&	logičko I	2
		logičko ILI	2
Relacioni	>	veće od	2
	<	manje od	2
	> =	veće ili jednako	2
	< =	manje ili jednako	2
Jednakosti	= =	jednakost	2
	!=	nejednakost	2
	= ==	case jednakost	2
	!= =	case nejednakost	2
Bit	~	bit negacija	1
	&	bit I	2
		bit ILI	2
	^	bit EX-ILI	2
	^~ ili ~^	bit EX-NILI	2
Redukcionи	&	redukcion I	1
	~&	redukcion NI	1
		redukcion ILI	1
	~	redukcion NILI	1
	^	redukcion EX-ILI	1
	^~ ili ~^	redukcion EX-NILI	1
Pomjerački	>>	pomjeranje udesno	2
	<<	pomjeranje ulijevo	2
Pridruživanje	{ }	pridruživanje	bilo koji broj
Umnožavanje	{ { } }	umnožavanje	bilo koji broj

Primitive Veriloga

- Bazni logički gejtovi
 - and
 - or
 - not
 - buf
 - xor
 - nand
 - nor
 - xnor

Verilog ključne riječi

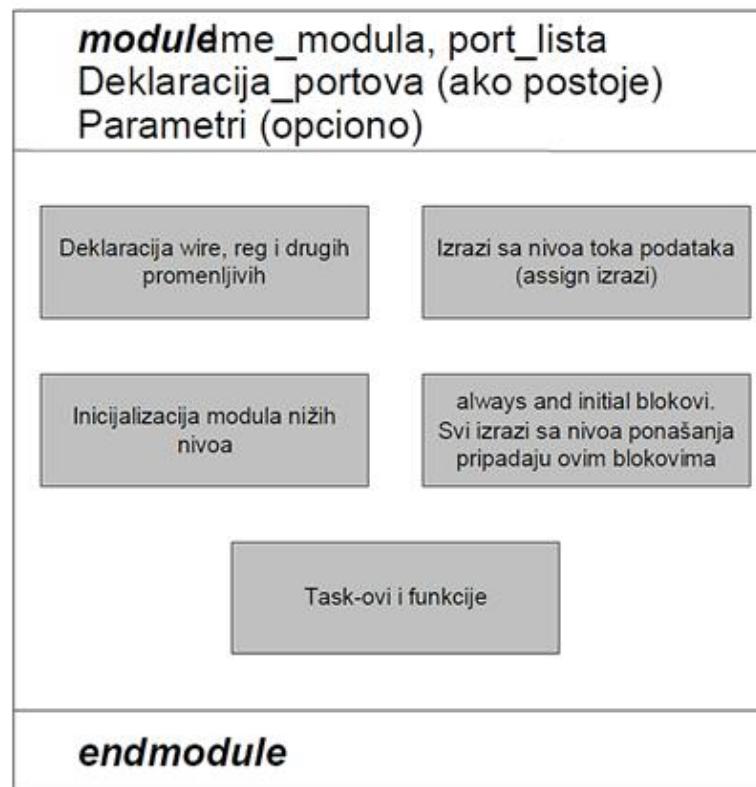
- Sve su definisane malim slovima
- Primjeri:
 - module, endmodule
 - input, output, inout
 - reg, integer, real, time
 - not, and, nand, or, nor, xor
 - parameter
 - begin, end
 - fork, join
 - specify, endspecify

Moduli

- Definisanje modula uvijek počinje pomoću ključne riječi **module**. Prvo se zadaju ime modula i lista port-ova, **zatim se deklarišu vrste port-ova i uvode parametri**. Lista portova i deklaracija portova su prisutni samo ako postoji interakcija modula i okruženja.
- Tijelo modula sačinjavaju pet komponenti koje su:
 - deklaracija tipa podataka,
 - izrazi sa nivoa toka podataka,
 - instanciranje modula nižih nivoa,
 - izrazi na nivou ponašanja i
 - Verilog funkcije.
- Navedene komponente se mogu nalaziti u bilo kom redoslijedu unutar tijela modula.

Moduli

- Modul je tekst fajl formiran od ASCII karaktera. Sastoji od nekoliko različitih dijelova



Moduli

- Definicija modula se uvek završava pomoću ključne riječi **endmodule**.
- Samo su ključna riječ **module**, ime modula i ključna riječ **endmodule** **obavezni** sve ostalo se koristi prema potrebi.
- Verilog HDL dozvoljava definisanje više modula unutar jednog ASCII fajla i to u bilo kom redoslijedu.

Moduli

- **module** – fundamentalni blok za Verilog dizajn; slični procedurama ili funkcijama u drugim programskim jezicima.
- **endmodule** – kraj modula
- Deklaracija modula
 - **module** *module_name* (*module_port*, *module_port*, ...);
<deklaracije>
<module items>
endmodule
- Primjer:
module full_adder (A, B, c_in, c_out, S);

Moduli

- Ulazna deklaracija
 - Skalar
 - **input** lista ulaznih identifikatora;
 - Primjer: input A, B, c_in;
 - Vektor
 - **input** [range] lista ulaznih identifikatora;
 - Primjer: input [15:0] A, B, data;
- Izlazna deklaracija
 - Skalar primjer : **output** c_out, OV, MINUS;
 - Vektor primjer : **output** [7:0] ACC, REG_IN, data_out;

Moduli

module & endmodule
sandwich the content of
this hardware module

Hex_to_7seg.v

```
-----  
// Module name: hex_to_7seg  
// Function: convert 4-bit hex value to drive 7 segment display  
// output is low active  
// Creator: Peter Cheung  
// Version: 1.0  
// Date: 22 Oct 2011  
-----  
  
module hex_to_7seg  (out,in);  
    output [6:0] out;      // low-active output to drive 7 segment display  
    input  [3:0] in;       // 4-bit binary input of a hexadecimnal number  
  
    assign out[6] = ~in[3]&~in[2]&~in[1] | in[3]&in[2]&~in[1]&~in[0] |  
             ~in[3]&in[2]&in[1]&in[0];  
    assign out[5] = ~in[3]&~in[2]&in[0] | ~in[3]&~in[2]&in[1] |  
             ~in[3]&in[1]&in[0] | in[3]&in[2]&~in[1]&in[0];  
    assign out[4] = ~in[3]&in[0] | ~in[3]&in[2]&~in[1] | in[3]&~in[2]&~in[1]&in[0];  
    assign out[3] = ~in[3]&in[2]&~in[1]&~in[0] | ~in[3]&~in[2]&~in[1]&in[0] |  
             in[2]&in[1]&in[0] | ~in[2]&in[1]&~in[0];  
    assign out[2] = ~in[3]&~in[2]&in[1]&~in[0] | in[3]&in[2]&~in[0] |  
             in[3]&in[2]&in[1];  
    assign out[1] = in[3]&in[2]&~in[0] | ~in[3]&in[2]&~in[1]&in[0] |  
             in[3]&in[1]&in[0] | in[2]&in[1]&~in[0];  
    assign out[0] = ~in[3]&~in[2]&~in[1]&in[0] | ~in[3]&in[2]&~in[1]&~in[0] |  
             in[3]&in[2]&~in[1]&in[0] | in[3]&~in[2]&in[1]&in[0];  
  
endmodule
```

good header helps
documenting your code

specify interface to this
module as viewed from
outside

specify a 7-bit output bus,
out[6] ... out[0]

declaration of
input and output
ports

assign used to specify
combinational circuit

Struktuirane procedure

- Opisi na nivou ponašanja se mogu pisati u okviru jedne od dvije moguće struktuirane procedure (eng. structured procedure).
- Na osnovu ključnih riječi kojima se uvode te procedure oni se zovu **initial** i **always** procedure.
- Instrukcije na nivou ponašanja se mogu pojaviti samo unutar tih procedura.
- Jeden Verilog HDL modul može da sadrži i više struktuiranih procedura.
- Definicija nove procedure se ne može početi dok nije završena prethodno započeta.

Struktuirane procedure

- Sve te procedure u toku simulacije projektovanog modula počinju u trenutku $t=0$ i izvršavaju se paralelno (konkurentno), za razliku od softverskih algoritama gdje se izvršavanje programa dešava sekvencialno, korak po korak.
- Ta osobina je povezana sa time da se u digitalnom kolu koje se realizuje na bazi HDL opisa, razni blokovi takođe rade istovremeno, paralelno obavljaju funkcije.

Procedura tipa initial

- Procedura tipa initial počinje sa **ključnom riječi initial**. Poslije ključne riječi slijedi initial blok koji može da sadrži jednu ili više dodjela vrijednosti. Izvršavanje operacija u initial bloku počinje u trenutku $t=0$ i izvršavanje se dešava tačno jednom. Ako jedan modul sadrži više initial procedura, izvršavanje svakog od njih počinje u trenutku $t=0$, izvršavaju se i završavaju se međusobno nezavisno.
- Pri projektovanju digitalnih kola initial procedure se obično **koriste za jednokratna podešavanja**. U većini slučajeva initial procedure se ne odnose na opis nekog kola, već se koriste u simulacionim modulima za **zadavanje pobudnih signala**.

Procedura tipa always

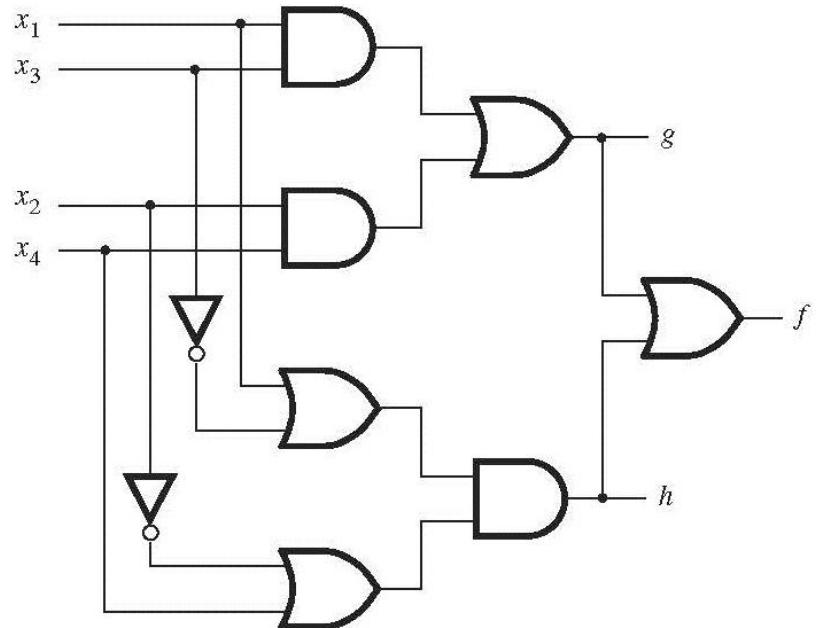
- Definisanje always procedure počinje sa **ključnom riječi always**, nakon čega slijedi always blok sa jednom ili više dodjela.
- I u ovom slučaju je potrebno koristiti ključne riječi begin i end ako se radi o bloku koji sadrži više dodjela.
- **Dodjele iz always bloka se ciklički ponavljaju u toku cijelog vremena simulacije** (ako je reč o simulacionom modulu) ili za sve vrijeme rada projektovanog kola.

Struktorna specifikacija logičkih krugova (Gate level)

```
module example2 (x1, x2, x3, x4, f, g, h);
    input x1, x2, x3, x4;
    output f, g, h;

    and (z1, x1, x3);
    and (z2, x2, x4);
    or (g, z1, z2);
    or (z3, x1, ~x3);
    or (z4, ~x2, x4);
    and (h, z3, z4);
    or (f, g, h);

endmodule
```

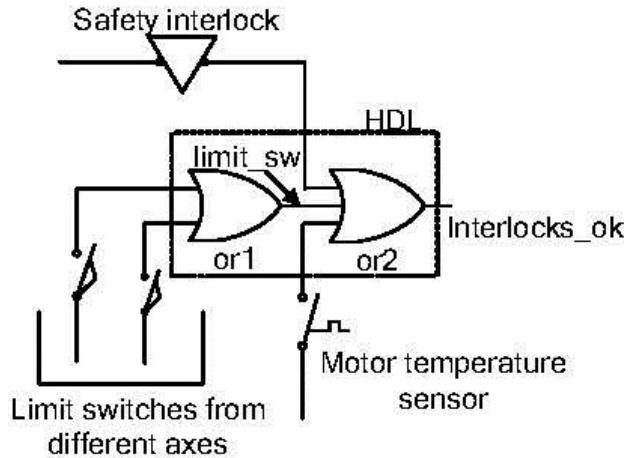


$$g = x_1 x_3 + x_2 x_4$$

$$h = (x_1 + \bar{x}_3)(\bar{x}_2 + x_4)$$

$$f = g + h$$

Primjer: Model sigurnosne blokade realizovan sa logičkim kolima



Zbog sigurnosti, mnogi senzori se koriste za zaštitu tijela robota od samodestrukcije.

Senzori: granična sklopka, senzor blizine, elektro sigurnosna sklopka.

Zadatak: Konvertovati kontrolnu shemu u digitalnu logiku korištenjem Verilog HDL.

Elektro sigurnosna sklopka realizovana korištenjem osnovnih logičkih kola

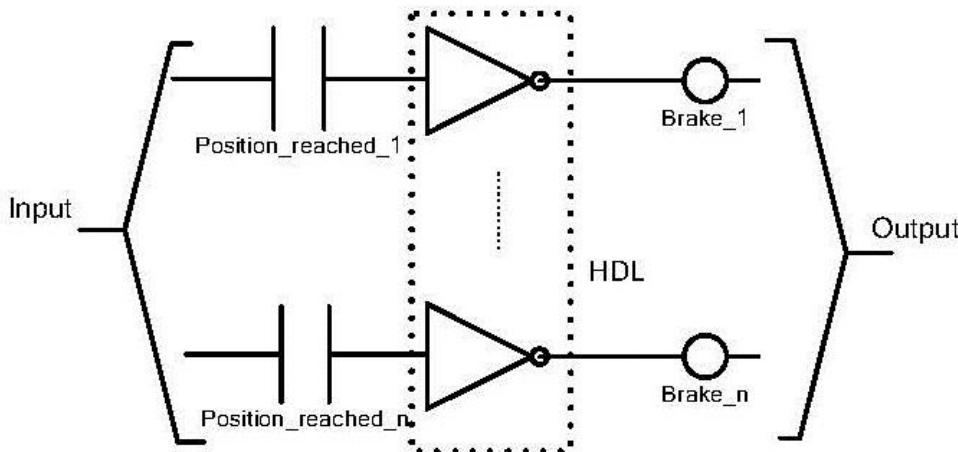
```
module gate_1 (input lim_sw1, lim_sw2, motor_temp, safety, output interlocks_ok);

    wire limit_sw;

    or or1 ( limit_sw , lim_sw1 , lim_sw2);
    or or2 (interlocks_ok , limit_sw , motor_temp , safety);

endmodule
```

Primjer: Prekidna sklopka realizovana sa logičkim kolima



```
module brake( input axis_position, output brake);
```

```
// Gate Instantiation
```

```
not (brake, axis_position);
```

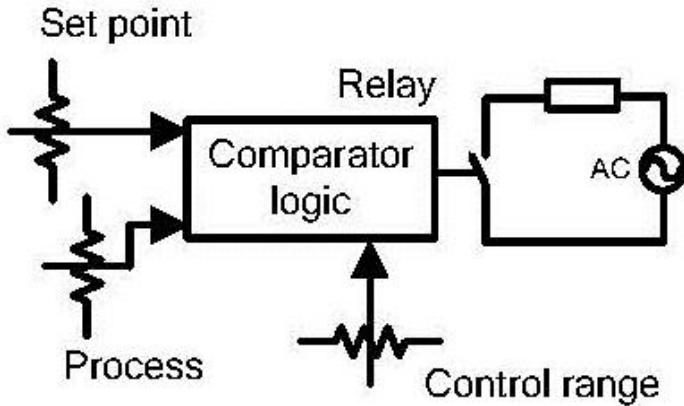
```
endmodule
```

Zbog zadržavanja robotskog zgloba na željenoj lokaciji nakon pozicioniranja, često se koristi prekid. Realizuje se kao dio kontrole motora zgloba. Kod pokretanog robotskog zgloba (obično napajanjem, logic 1), prekid se realizuje kada zglob dosegne njegovu predefinisanu poziciju, tj. zakoči se (logic 0).

Pridruživanja

- **Kontinualna pridruživanja** pridružuju vrijednosti (vektor i scalar) kad god ulazni operator promijeni vrijednost (prati **wire** varijabe)
 - Ključna riječ **assign** se koristi **za razlikovanje proceduralnog od kontinualnog pridruživanja.**
Primjer: **assign** out = in1 & in2;
- **Proceduralno pridruživanje** pobuđenih vrijednosti u registre (vektor i scalar)
 - Dešava se unutar procedura kao **always** i **initial**
 - Blok i Ne-Blok pridruživanja unutar procedura
- Pravilo u Verilogu – ukoliko se izvršava pridruživanje variable u okviru **always** bloka, **varijabla se mora deklarisati kao reg**, ne kao **net (wire)**.

Primjer: Kombinaciono kolo realizovano korištenjem modeliranja toka podataka



ON/OFF upravljačka shema

Ulazi u kontroler, željena vrijednost i procesne vrijednosti su raspoložive iz 8-bit uni-polar ADC. Verilog uključuje **assign** iskaz za realizaciju kombinacione logike za proračun greške.

Kreiranje logike za ON/OFF kontroler, sličan korištenim u frižiderima i prečistačima zraka. Korisnik postavlja iznos potrebne upravljačke akcije. **Digitalno kolo proračunava razliku između željene i stvarne temperature i uključuje relej.**

```
module on_off (input [7:0] set_point, process, input [3:0] control_range, output relay);

wire [7:0] error; // Unsigned value

assign error = set_point - process; // Set point is always more than the process value

assign relay = ( error > control_range ) ? 1'b1 : 1'b0;

endmodule
```

Projektovanje sistema
na čipu 3

Proceduralna pridruživanja

- **Iskaz blok pridruživanja (= operator)** djeluje kao u tradicionalnim programskim jezicima.
 - Izvršava se u redoslijedu koji je specificiran u sekvencijalnom iskazu.
 - Korištenje blocking pridruživanja u *always* blokovima koja su isključivo kombinaciona
- **Iskaz ne-blok pridruživanja(<= operator)**
 - Dozvoljava izvršenje svakog iskaza bez obzira na rezultate iz predhodnih sekvencijalnih iskaza.
 - Nonblocking pridruživanje u *always* blokovima za sintezu/simulaciju sekvencijalne logike ili miksa kombinacionih i sekvencijalnih kola.

Blok i ne-blok pridruživanje

- Verilog ima dvije forme pridruživanja: **blok** i **ne-blok**.
- Blok pridruživanje** = se izvršava u redoslijedu, kako se pojavljuje, tj. jedno iza drugog. Stoga prvi iskaz „blokira“ drugi, sve dok se on prvo izvrši, razlog za ovaj naziv!

```
a = b;      blocking  
b = a;  
// both a & b = b
```

```
always @ (a or b or c)  
begin  
    x = a | b;           1. Evaluate a | b, assign result to x  
    y = a ^ b ^ c;       2. Evaluate a^b^c, assign result to y  
    z = b & ~c;          3. Evaluate b&(~c), assign result to z  
end
```

- Ne-blok pridruživanje** **<=** se izvršava u paraleli. Nema blokiranja slijedećeg izraza od strane ranije navedenog izraza. Obratiti pažnju na efekte izvršavanja u okviru **always**:

```
a <= b;  Non-blocking  
b <= a;  
// swap a and b
```

```
always @ (a or b or c)  
begin  
    x <= a | b;           1. Evaluate a | b but defer assignment of x  
    y <= a ^ b ^ c;       2. Evaluate a^b^c but defer assignment of y  
    z <= b & ~c;          3. Evaluate b&(~c) but defer assignment of z  
end  
                                         4. Assign x, y, and z with their new values
```

Kontrolne konstrukcije u Verilogu

- Prilagođene izvršavanju na hardveru:
 - **If-else iskaz** se koristi kada se na osnovu kriterija vrši izbor dijela koda koji će se izvršavati.
 - **Case iskaz** se koristi kada treba provjeriti vrijednost jedne varijable.
 - **‘For’ petlja** - izvršava dio koda tačno određen broj puta.
 - **‘While’ petlja**- izvršava određeni dio koda sve dok je zadani uslov tačan.
 - **,Repeat’ petlja** ponavlja neke izraze određeni broj puta.
 - **,Forever’ petlja**

If-else izkaz

```
if(expr1)
    true_stmt1;
else if(expr2)
    true_stmt2;
..
else
    def_stmt;
```

Mora se koristiti 'reg'
deklaracija ako je
signal pridružen iz always
bloka.

Proces trigerovan na
bilo koju promjenu
signala in ili sel

Primjer: 4-to-1 mux:

```
module mux4_1(out, in, sel);
    output out;
    input [3:0] in;
    input [1:0] sel;
    reg out;
    wire [3:0] in;
    wire [1:0] sel;
    always @ (in or sel)
        if (sel == 0)
            out = in[0];
        else if (sel == 1)
            out = in[1];
        else if (sel == 2)
            out = in[2];
        else
            out = in[3];
endmodule
```

Blok
pridruživanje.

Case iskaz

case (expr)

item_1, .., item_n: stmt1;

item_n+1, .., item_m: stmt2;

..

default: def_stmt;

endcase

Primjer: 4-to-1 mux:

```
module mux4_1(out, in, sel);
    output out;
    input [3:0] in;
    input [1:0] sel;

    reg out;
    wire [3:0] in;
    wire [1:0] sel;

    always @ (in or sel)
        case (sel)
            0: out = in[0];
            1: out = in[1];
            2: out = in[2];
            3: out = in[3];
        endcase
    endmodule
```

For petlja

- Ova petlja se definiše pomoću ključne riječi **for** i izraza u zagradi pored te ključne riječi.
- Izraz sadrži tri elementa:
 - jedan početni uslov,
 - provjeru uslova za izlazak iz petlje,
 - jednu dodjelu koja modifikuje vrijednost podatka kojom je određen momenat izlaska iz petlje.
- **Sintaksa:**
- **for (<initialassignment>;<expression>,<step assignment>)<statement>**

```
module dec2to4 (W, Y, En);
    input [1:0] W;
    input En;
    output [0:3] Y;
    reg [0:3] Y;
    integer k;

    always @ (W or En)
        for (k = 0; k <= 3; k = k+1)
            if ((W == k) && (En == 1))
                Y[k] = 1;
            else
                Y[k] = 0;
endmodule
```

While petlja

- Ova petlja se definiše pomoću ključne riječi **while** i izraza u zagradi pored te ključne riječi. Dodjele koje pripadaju petlji se ponavljaju dok izraz u zagradi daje tačnu vrijednost.
- Dodjele se neće izvršiti niti jednom ako je izraz netačan u startu.
- Ako ima više od jedne dodjele u petlji, oni treba da se smjeste između ključnih riječi begin i end.
- **Sintaksa:**
- **while (< expression >) < statement >**

Repeat petlja

- Ova petlja se definiše pomoću ključne riječi **repeat** i brojne vrijednosti u zagradi pored te ključne riječi. Umjesto brojne vrijednosti može da se pojavi i neki identifikator ili neki izraz, ali se njegova vrijednost računa prije ulaska u petlju i unutar petlje se smatra konstantom. Prema tome, **petlja tipa repeat se koristi u situacijama kada dodjele navedene u petlji treba izvršiti određeni broj puta.**
- **Sintaksa:**
- **repeat (< number >) < statement >**
- **repeat (16)**
begin

```
$display ("Current value of i is %d", i);
```

```
i = i + 1;
```

```
end
```

Petlja forever

- Ova petlja se definiše pomoću ključne riječi **forever**.
Definicija ne sadrži uslov za izlazak iz petlje, zato se petlja u principu neograničeno izvršava (pri simulacijama tražeće do kraja simulacije, pri hardverskoj realizaciji tražeće sve dok kolo dobije napajanje).
- **Sintaksa:**
- **forever < statement >**

Funkcije

- Svrha funkcije je da omogući pisanje koda na modularan način bez definisanja odvojenih modula.
- Funkcija je definisana unutar modula i poziva se ili u izrazu kontinuiranog dodjeljivanja, ili u izrazu proceduralnog dodjeljivanja unutar tog modula.
- Funkcija može imati više od jednog inputa, **ali nema outputa, jer samo ime funkcije služi kao output varijabla.**

```
function [range | integer] function_name;  
[input declarations]  
[parameter, reg, integer declarations]  
begin  
    statement;  
end  
endfunction
```

Primjer funkcije

```
• module mux_f (W, S16, f);
  •   input [0:15] W;
  •   input [3:0] S16;
  •   output reg f;
  •   reg [0:3] M;

  •   function mux4to1;
  •     input [0:3] W;
  •     input [1:0] S;
  •     if (S == 0) mux4to1 = W[0];
  •     else if (S == 1) mux4to1 = W[1];
  •     else if (S == 2) mux4to1 = W[2];
  •     else if (S == 3) mux4to1 = W[3];
  •   endfunction

  •   always @(W, S16)
  •   begin
  •     M[0] = mux4to1(W[0:3], S16[1:0]);
  •     M[1] = mux4to1(W[4:7], S16[1:0]);
  •     M[2] = mux4to1(W[8:11], S16[1:0]);
  •     M[3] = mux4to1(W[12:15], S16[1:0]);
  •     f = mux4to1(M[0:3], S16[3:2]);
  •   end
  • endmodule
```

Projektovanje sistema
na čipu 3

f = mux4to1 (M[0:3], S16[3:2]);

In-ekvivalent ovog djela koda



```
if (S16[3:2] == 0) f = M[0];
else if (S16[3:2] == 1) f = M[1];
else if (S16[3:2] == 2) f = M[2];
else if (S16[3:2] == 3) f = M[3];
```

Taskovi u Verilogu

- Drugi način pisanja 16-to-1 moltiplesera je sa Verilog taskovima, koji je sličan funkciji.
- **Dok funkcija vraća vrijednost, task je ne vraća.**
- **Ima input i output varijable** kao modul i može se pozvati samo unutar always (ili initial) bloka.

```
• module mux_t (W, S16, f);
•   input [0:15] W;
•   input [3:0] S16;
•   output reg f;
•   reg [0:3] M;

• task mux4to1;
•   input [0:3] W;
•   input [1:0] S;
•   output Result
•   if (S == 0) Result = W[0];
•   else if (S == 1) Result = W[1];
•   else if (S == 2) Result = W[2];
•   else if (S == 3) Result = W[3];
• endtask

•
• always @(W, S16)
• begin
•   mux4to1(W[0:3], S16[1:0], M[0]);
•   mux4to1(W[4:7], S16[1:0], M[1]);
•   mux4to1(W[8:11], S16[1:0], M[2]);
•   mux4to1(W[12:15], S16[1:0], M[3]);
•   mux4to1(M[0:3], S16[3:2], f);
• end
• endmodule
```

Usporedba task i function formi

- **Function i task-** za postavljanje istih djelova koda, koji se ponavljaju.
- Funkcije mogu vraćati vrijednost, dok taskovi ne mogu.
- Funkcije i taskovi imaju istu semantiku; razlika je da **taskovi imaju kašnjenja, dok funkcije nemaju bilo kakva kašnjenja.** To znači da se **funkcije koriste za modeliranje kombinacionih kola.**
- Funkcije ne mogu omogućiti task, dok **task može omogućiti druge taskove i funkcije.**

Primjeri projektovanja u automatici i robotici u Verilogu

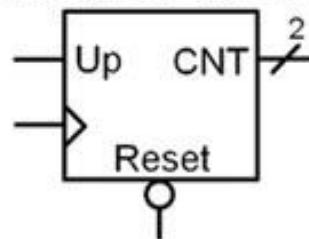
Primjer: FSM za Up/Down brojač

Example: 2-Bit Up/Down Counter in Verilog – Design Description and Port Definition

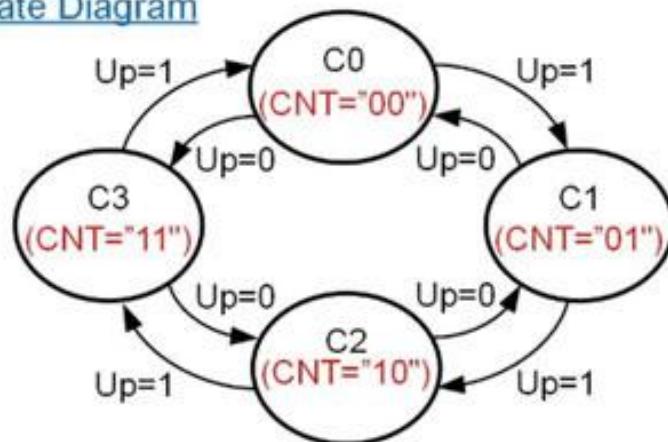
This system will output a synchronous, 2-bit, binary counter. When the system input Up=1, the system will count up. When Up=0, the system will count down. The output of the counter is called CNT.

Port Definition

Counter_2bitUpDown.v



State Diagram



```
module Counter_2bitUpDown
  (output reg [1:0] CNT,
   input wire      Clock, Reset, Up);
  :
```

Up/Down brojač u Verilogu

Example: 2-Bit Up/Down Counter in Verilog – Full Model (Three Block Approach)

```
module Counter_2bitUpDown (output reg [1:0] CNT,
                           input wire      Clock, Reset, Up);

    reg [1:0] current_state, next_state;
    parameter C0 = 2'b00,
              C1 = 2'b01,
              C2 = 2'b10,
              C3 = 2'b11;

    always @ (posedge Clock or negedge Reset)
    begin: STATE_MEMORY
        if (!Reset)
            current_state <= C0;
        else
            current_state <= next_state;
    end

    always @ (current_state or Up)
    begin: NEXT_STATE_LOGIC
        case (current_state)
            C0 : if (Up == 1'b1) next_state = C1; else next_state = C3;
            C1 : if (Up == 1'b1) next_state = C2; else next_state = C0;
            C2 : if (Up == 1'b1) next_state = C3; else next_state = C1;
            C3 : if (Up == 1'b1) next_state = C0; else next_state = C2;
        endcase
    end

    always @ (current_state)
    begin: OUTPUT_LOGIC
        case (current_state)
            C0 : CNT = 2'b00;
            C1 : CNT = 2'b01;
            C2 : CNT = 2'b10;
            C3 : CNT = 2'b11;
        endcase
    end
endmodule
```

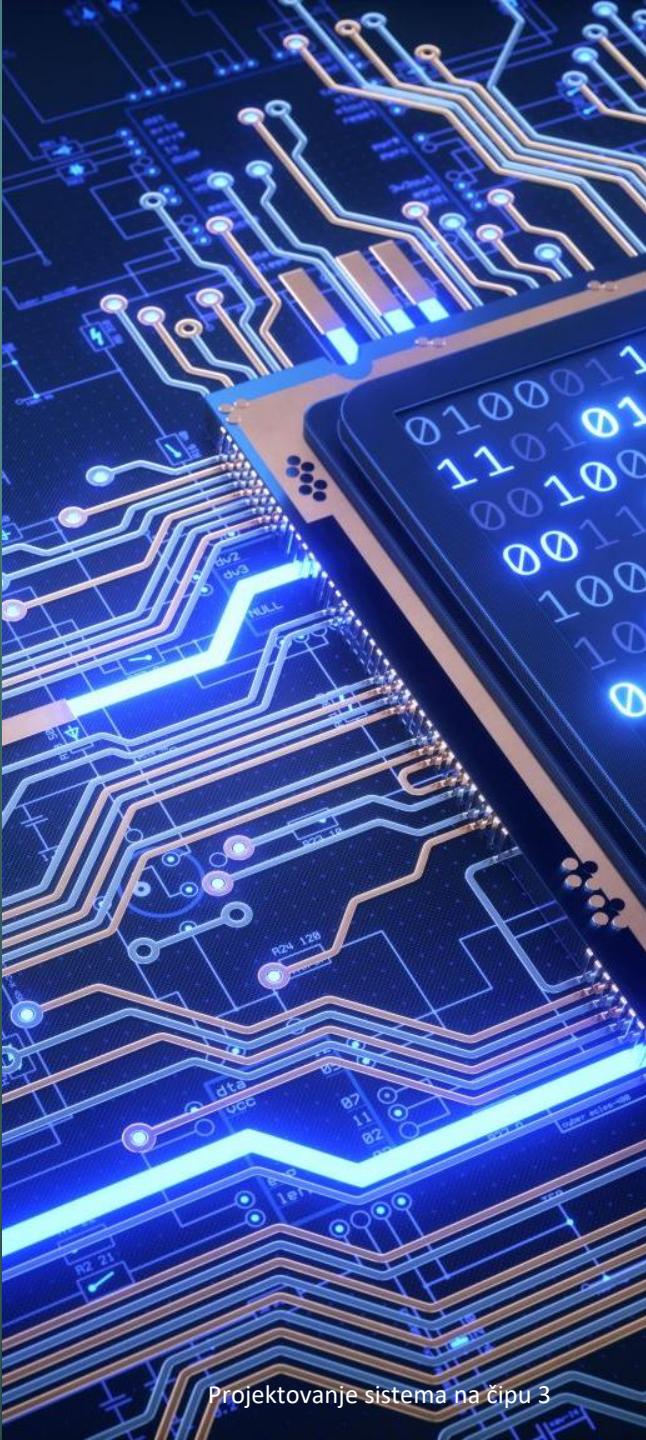
State variables and state encoding.

State memory block.

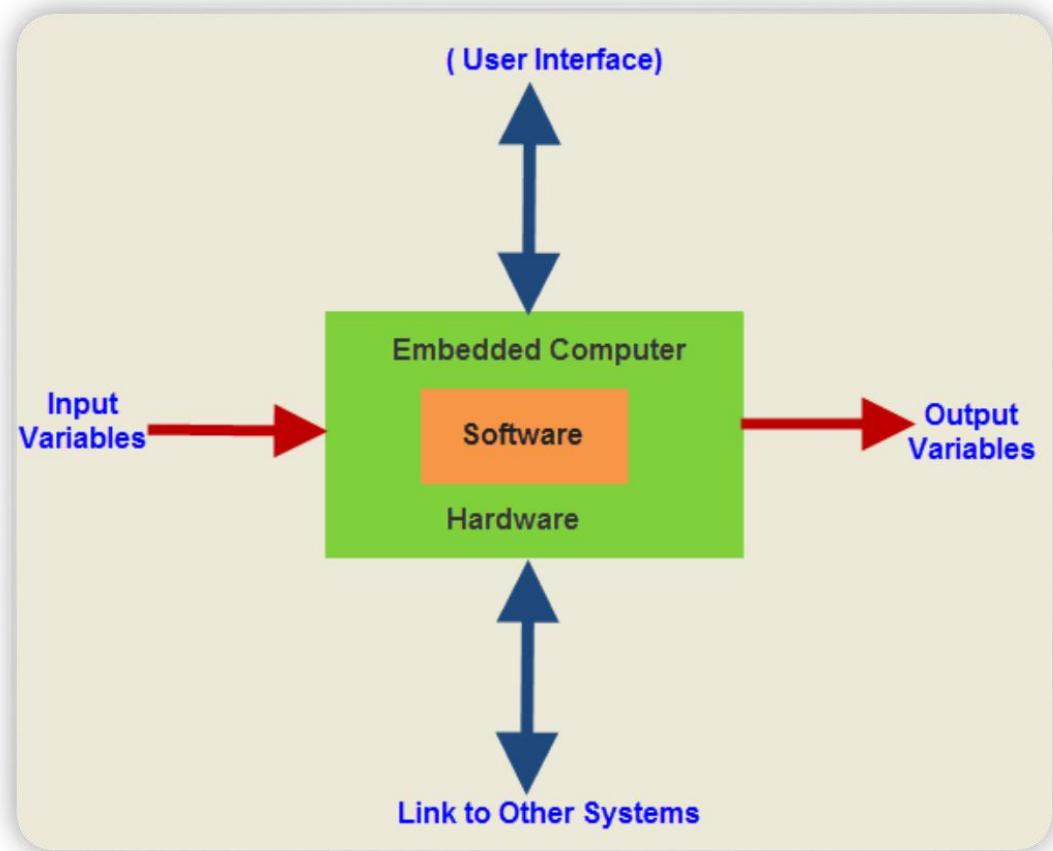
Next state logic block.

Output logic block. Note that since this is a Moore machine only the current state is listed in the sensitivity list.

Eksperimentalni dizajn FPGA baziranih rješenja

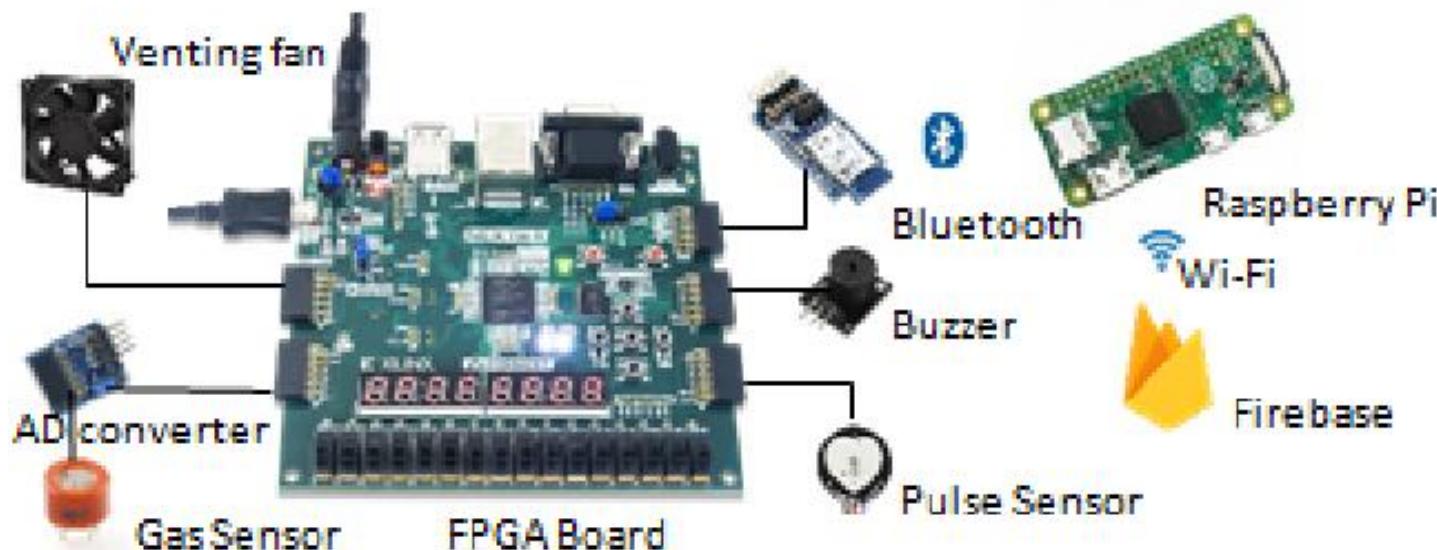


FPGA bazirani sistem monitoringa i upravljanja

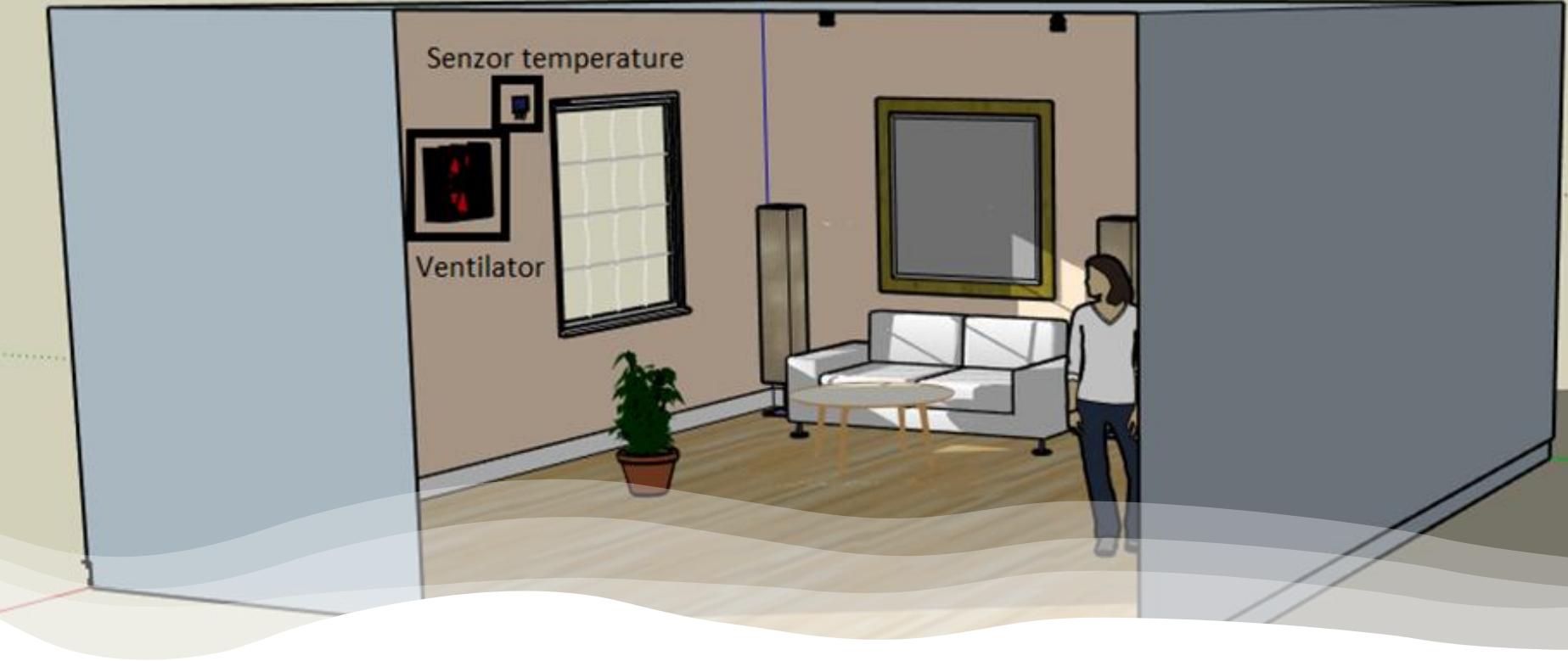


Specifikacija senzor-motor dizajna

- Specifikacija:
 - specifikacija dizajna
 - hardverske aspekte (senzorika, aktuatori, selektirana rekonfigurable ili SoC platforma)
 - softverska rješenja (način programiranja softverski i hardverski programske jezice)



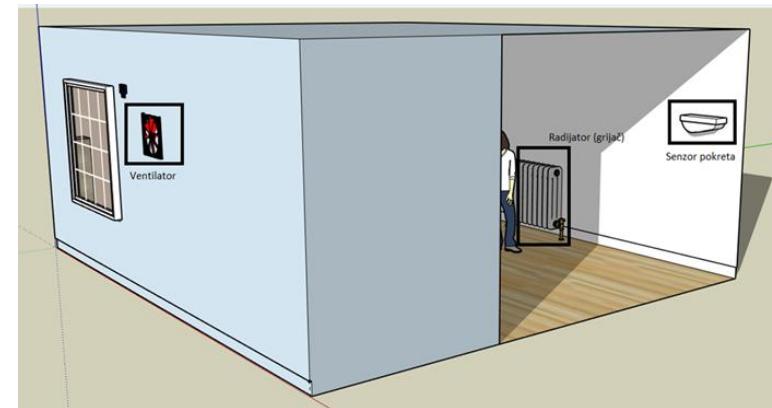
<https://www.semanticscholar.org/paper/A-pulse-sensor-interface-design-for-FPGA-based-Wang-Jang/7b20f20f31c6dd5d259891009104796acaf0c384>



Primjer: Regulacija temperature zraka u prostoriji pri detekciji ljudskog prisustva

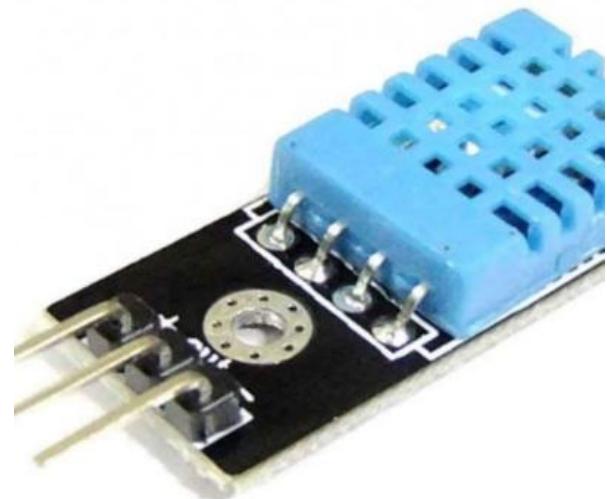
Regulacija temperature, pametne kuće

- Zadate referentne vrijednosti: HT (Visoka temperatura), LT (Niska temperatura) i HP (Prisustvo čovjeka).
- U prostoriji, postavljen motion senzor - detektuje kada čovjek uđe u prostoriju. Ukoliko je prostorija prazna regulacija temperature se neće izvršavati. Pri detekciji čovjeka, započinje regulacija temperature.
- Vrši se mjerjenje trenutne temperature u prostoru pomoću DHT11 senzora. Interval optimalne temperature je između LT i HT.
- **Ukoliko senzor DHT11 izmjeri veću temperaturu od HT, podaci sa arduina se šalju na FPGA i uključuje se DC ventilator u cilju smanjenja temperature u prostoriji na optimalnu temperaturu. Kada se dostigne optimalna temperatura ventilator se gasi.**
- **Ukoliko senzor DHT11 izmjeri manju temperaturu od LT, tada se uključi sijalica (imitacija grijajuća) Grijajući će raditi sve dok se ne postigne željena temperatura.**



DHT11 senzor

- Mjerenje temperature i vlagu zraka koristenjem DHT11 senzora.
- Senzor dolazi s namjenskim NTC-om za mjerjenje temperature i 8-bitnim mikrokontrolerom za izlaz vrijednosti temperature i vlažnosti kao serijskih podataka. Senzor je takođe fabrički kalibriran i stoga je jednostavan za povezivanje sa drugim mikrokontrolerima.
- Senzor može izmjeriti temperaturu od 0°C do 50°C i vlažnost od 20% do 90% s preciznošću od $\pm 2^{\circ}\text{C}$ i $\pm 5\%$, a radni napon mu je od 3V do 5.5V.
- DHT11 koristi samo jednu žicu za prenos podataka na korištenu pločicu. Snaga dolazi iz odvojenih 5V i uzemljenih žica. Između signalne linije i 5V linije potreban je povlačni otpor od 10K Ohma kako bi se osiguralo da nivo signala po defaultu ostaje visok.



Senzor pokreta

Senzor pokreta je električni uređaj koji koristi senzor za otkrivanje pokreta u blizini. Takav je uređaj često integriran kao komponenta sistema koji automatski izvršava zadatak ili upozorava korisnika na kretanje u nekom području.



Najpoznatiji su IR(infrared) i ultrasonični senzori.



IR senzor



Ultrasonični senzor

Ventilator

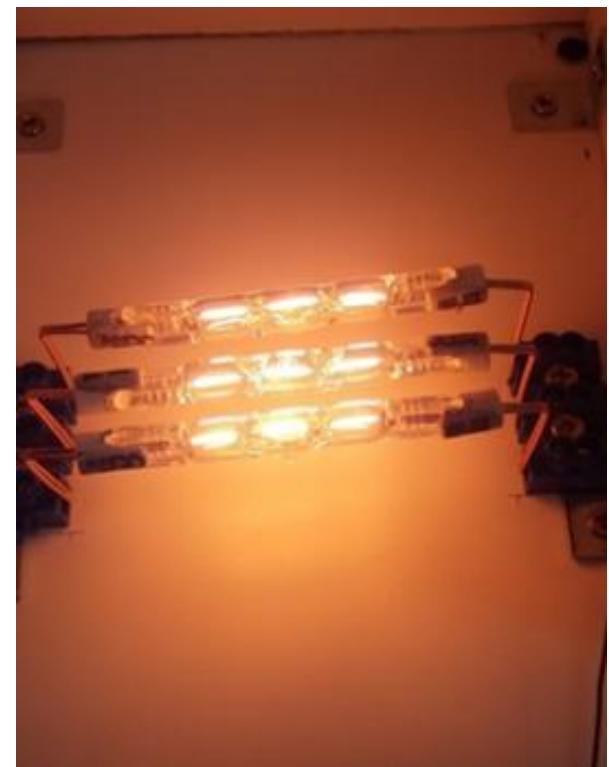
- Za hlađenje prosorije, korišten ventilator od laptopa (cooler).
- Ventilatori se koriste za uvlačenje hladnijeg zraka u kućište izvana, izbacivanje toplog zraka iznutra i pomicanje zraka preko hladnjaka radi hlađenja određene komponente.
- Jednostavni ventilator sastoji se od lopatice ventilatora, DC motora, potenciometra i napajanja.



DC ventilator

Halogene sijalice kao grijači

- Halogena sijalica je sijalica sa žarnom niti, ali poboljšana, tako da je skoro dvostruko učinkovitija od obične sijalica sa žarnom niti.
- Kvalitet svjetlosti koju daje ne opada sa radnim satima te ima kompaktniju izvedbu.
- Umjesto vakuma ili inertnog plina koji se nalazi u običnim sijalicama, u halogenoj sijalici se nalazi halogeni plin koji reaguje s atomima plinovitog volframa (koji je ispario sa žarne niti). Kada taj spoj ponovo dođe na žarnu nit, visoka temperatura ga razbije i atom volframa ponovo postane dio žarne niti, čime se produži radni vijek koji iznosi oko 2000 sati.

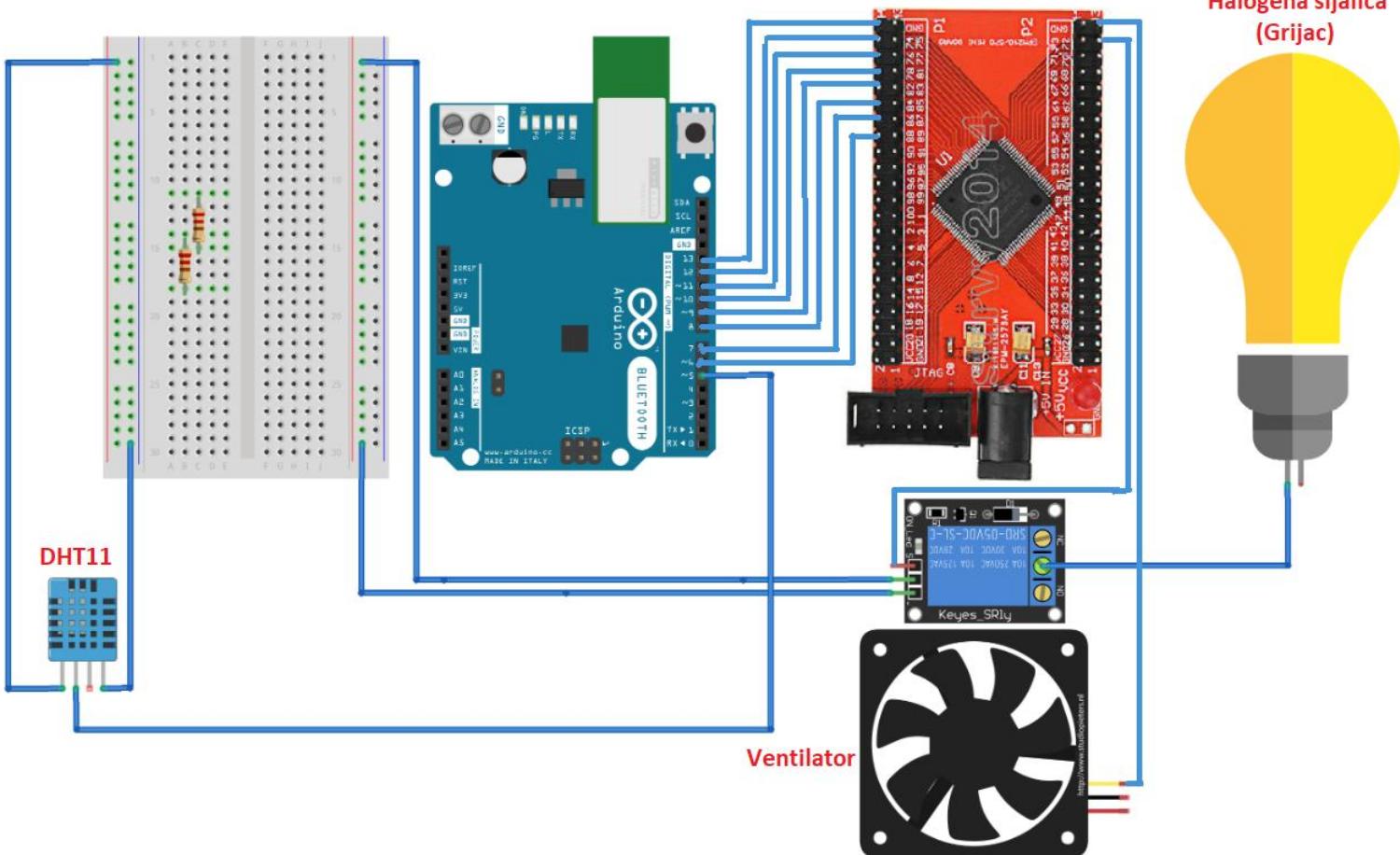




Releji

- Relej je jedna od najčešće korišćenih električkih komponenti u savremenoj automatizaciji.
- Posjeduju niz pogodnosti koje utiču na čestu primjenu, a najznačajnije od njih su rad na opsegu temperatura od -40°C do +80°C kao i lako održavanje.
- Postoje nekoliko vrste releja: releji snage, step releji, industrijski releji, vremenski releji itd.

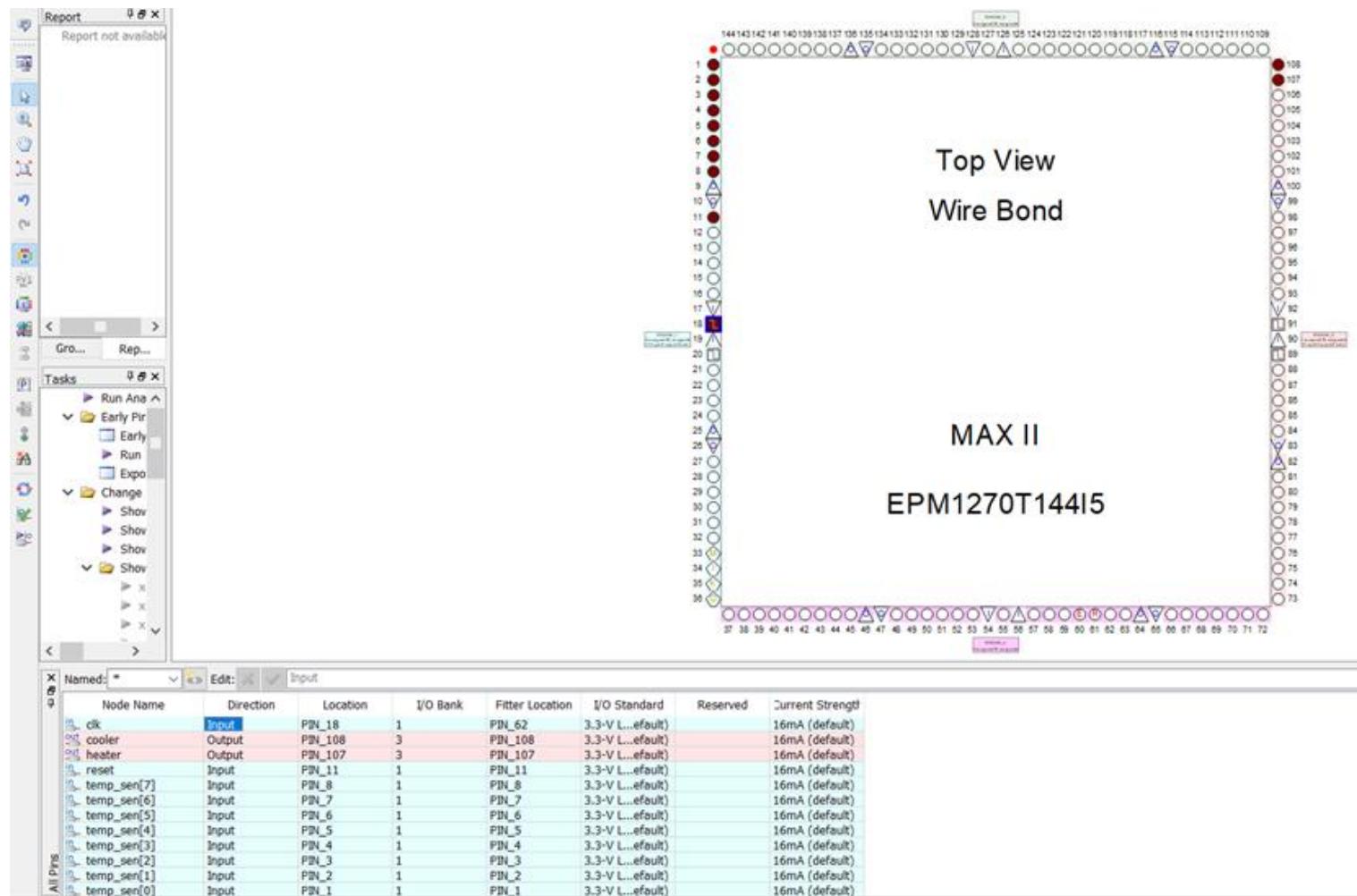
Šema spajanja

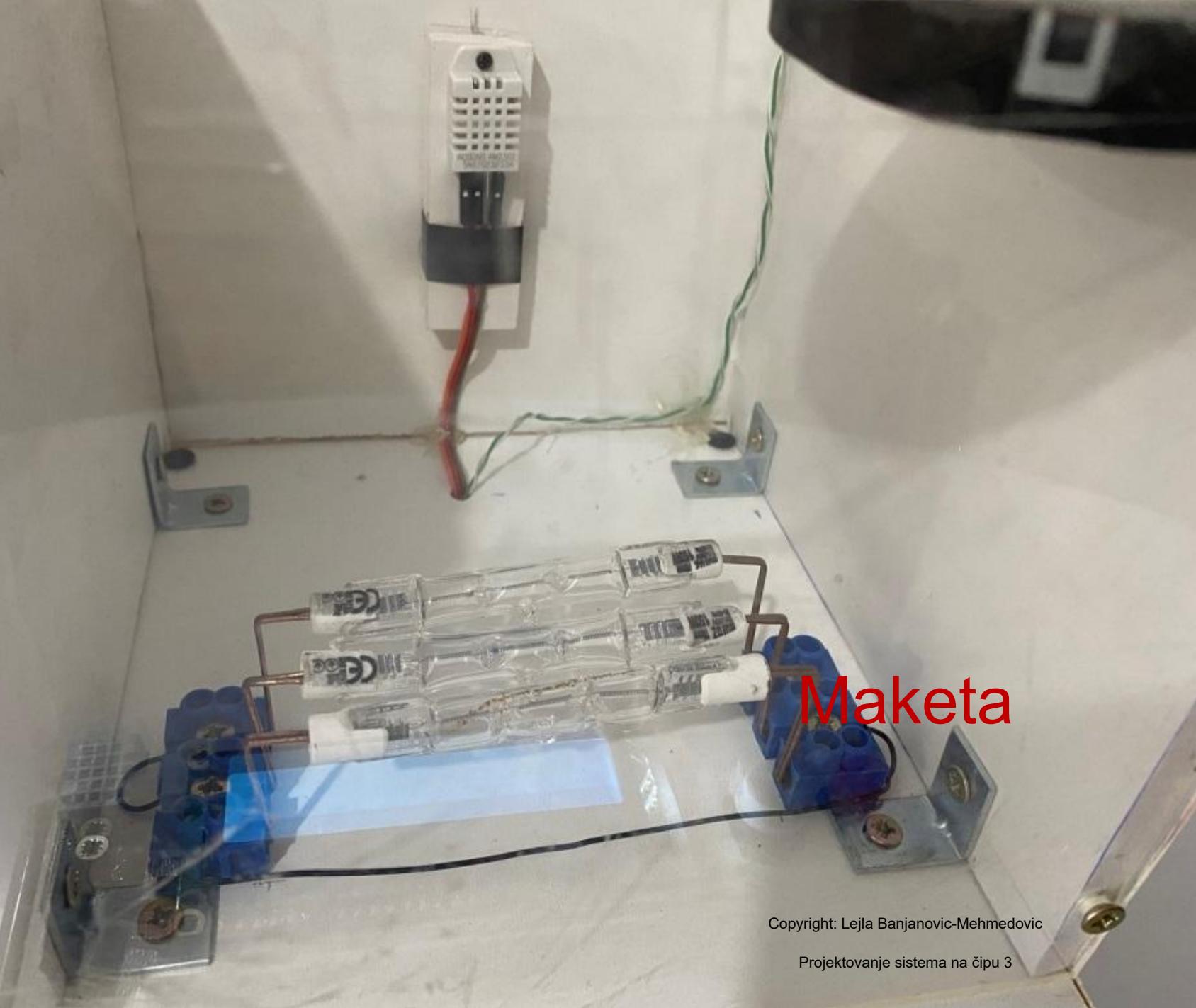


Projektovanje sistema na čipu 3

Copyright: Lejla Banjanovic-
Mehmedovic

Pin Planner





Maketa

Copyright: Lejla Banjanovic-Mehmedovic

Projektovanje sistema na čipu 3

```

`define start 4'd0
`define temp_heat 4'd1
`define temp_cool 4'd2

module TempRegulation(clk,reset,temp_sen,heater,cooler);

    input clk,reset;
    input [7:0] temp_sen; //mjerena temp
    output reg heater,cooler; //grijac i ventilator
    reg [3:0] current_state; //trenutno stanje
    reg [3:0] next_state; //sljedece stanje
    reg [7:0] desired_temp; //zeljena temp
    reg motion_sen; //sensor pokreta
    wire clk;

    initial begin
        current_state=`start;
        next_state= `start;
        heater='b0;
        cooler='b0;
        desired_temp=8'b00011001; //25
        motion_sen='b1;
    end

    always @(clk)
        current_state=next_state;
        always @ (current_state)
            begin
                case(current_state)
                    `start:
                        begin
                            heater='b0;
                            cooler='b0;
                        end

```

Verilog kod

--hladjenje ili zagrijavanje zapocinje tek onda kada se detektuje covjek
--u prostoriji, tj kada je motion_sen==1

Verilog kod

```
'temp_heat: //zagrijavanje
    begin if(motion_sen==1)
        begin
            heater ='b1;
            cooler='b0;
        end
    else
        heater ='b0;
        cooler ='b0;
    end

'temp_cool: //hladjenje
    begin if(motion_sen==1)
        begin
            cooler ='b1;
            heater ='b0;
        end
    else
        cooler ='b0;
        heater ='b0;
    end

endcase
end
```

--u zavisnosti od mjerene temp,trenutnog stanja i reseta zapocinje kod u always blocku, u kojem se određuje koje je --sljedeće stanje, tj da li se pali grijac ili ventilator, ili nijedno

```
always @(current_state,temp_sen,reset)
begin
if(reset=='b1)
next_state='start;
else
case(current_state)

`start: begin
    if(temp_sen > desired_temp)
        next_state=`temp_cool;
    else if(temp_sen < desired_temp)
        next_state=`temp_heat;
end

`temp_cool: begin
    if(temp_sen < desired_temp)
        next_state=`temp_heat;
end

`temp_heat: begin
    if(temp_sen > desired_temp)
        next_state=`temp_cool;
end

endcase
end
endmodule
```

Verilog kod

Arduino kod za akviziciju signala

```
void setup(){
    Serial.begin(9600);
    Serial.println("Trenutna_temp,Zeljena_temp,
heater,DCFan");
}

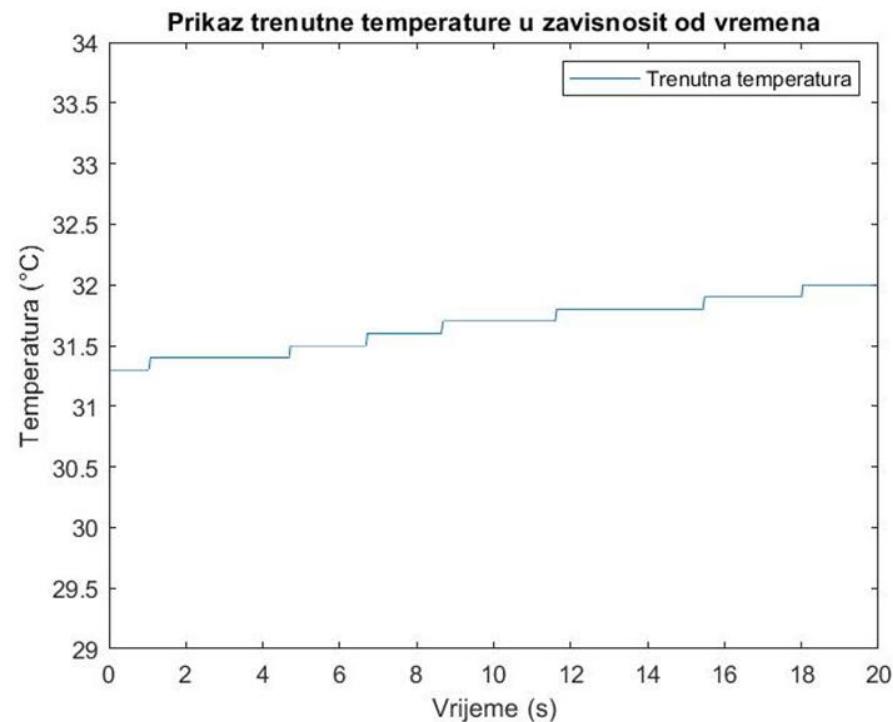
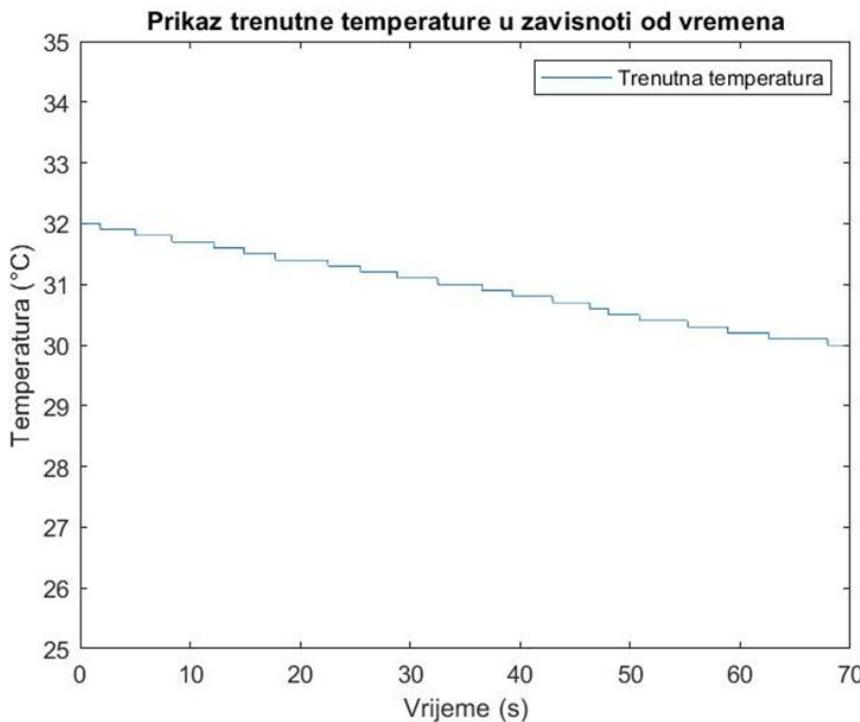
void loop(){
    double Trenutna_temp= digitalRead(5);
    double heater = digitalRead(6);
    double DCFan = digitalRead(9);
    Serial.print(Trenutna_temp);
    Serial.print(",");
    Serial.print(Zeljena_temp);
    Serial.print(",");
    Serial.print(heater);
    Serial.print(",");
    Serial.print(DCFan);
    Serial.print(",");
}
}
```

MATLAB kod za plotanje vremenskih dijagrama

- `data = xlsread('RealTime.xlsx');`
- `timeData = xlsread('Time.xlsx');`
- `Time = timeData(:,1);`
- `Ventilator = data(:,1);`
- `Grijac = data(:,2);`
- `Zeljena_temp = data(:,3);`
- `Trenutna_temp = data(:,4);`
- `figure(1)`
- `plot(Time,Ventilator)`
- `hold on`
- `legend('Ventilator')`
- `ylabel('Trenutno stanje ventilatora');`
- `xlabel('Vrijeme (s)');`
- `Title('Prikaz rada ventilatora u zavisnosti od vremena');`

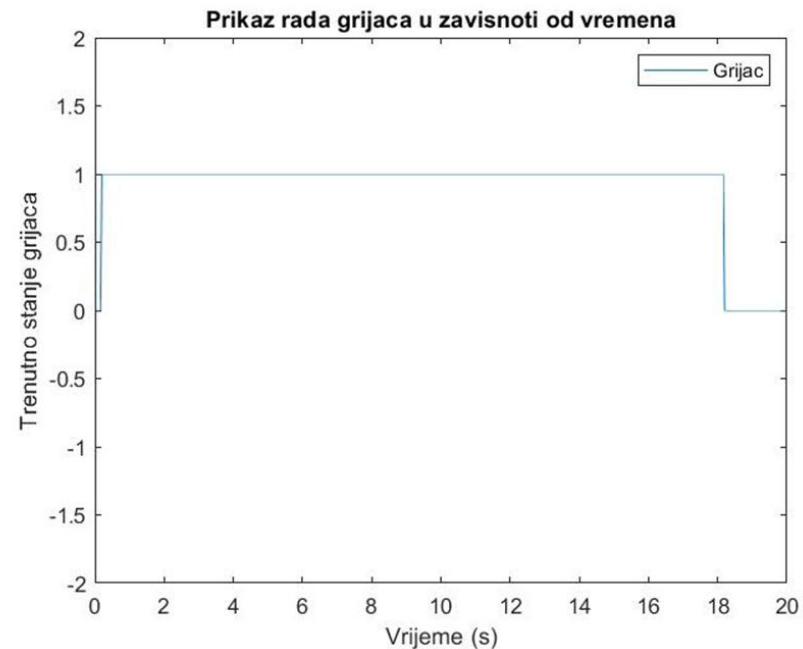
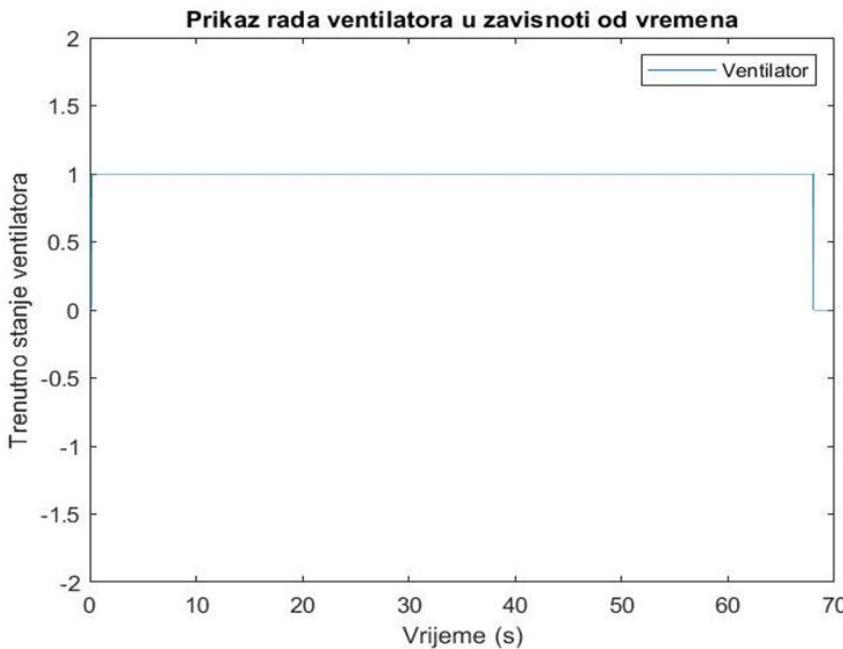
Upravljanje hlađenjem i zagrijavanjem prostorije

- Lijeva slika: trenutna temperatura u prostoriji Ta temperatura iznosi 32°C . S obzirom da je željena temperatura manja od trenutne(mjerene) temperature, mora se izvršiti hlađenje prostorije.



Upravljanje hlađenjem i zagrijavanjem prostorije

- 2 komplementarna slučaja



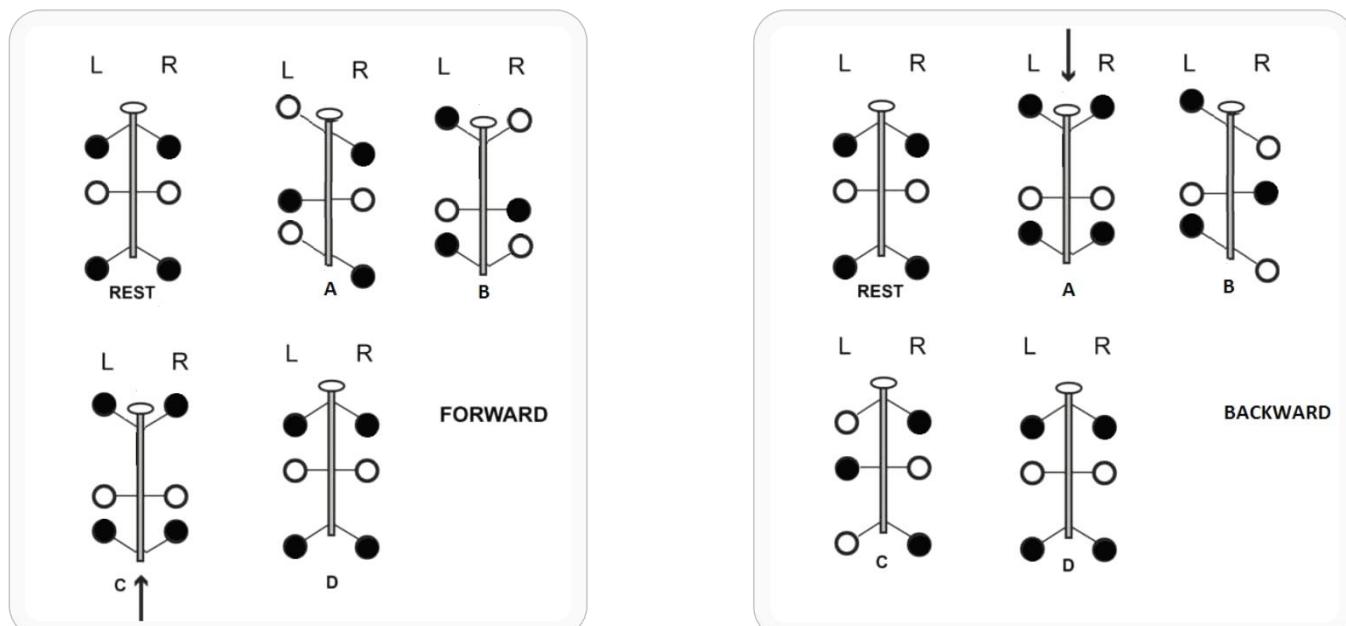
Primjer 2: FPGA bazirana navigacija hexapoda

- Biološka inspiracija za ugradbeni dizajn-kretanje insekta



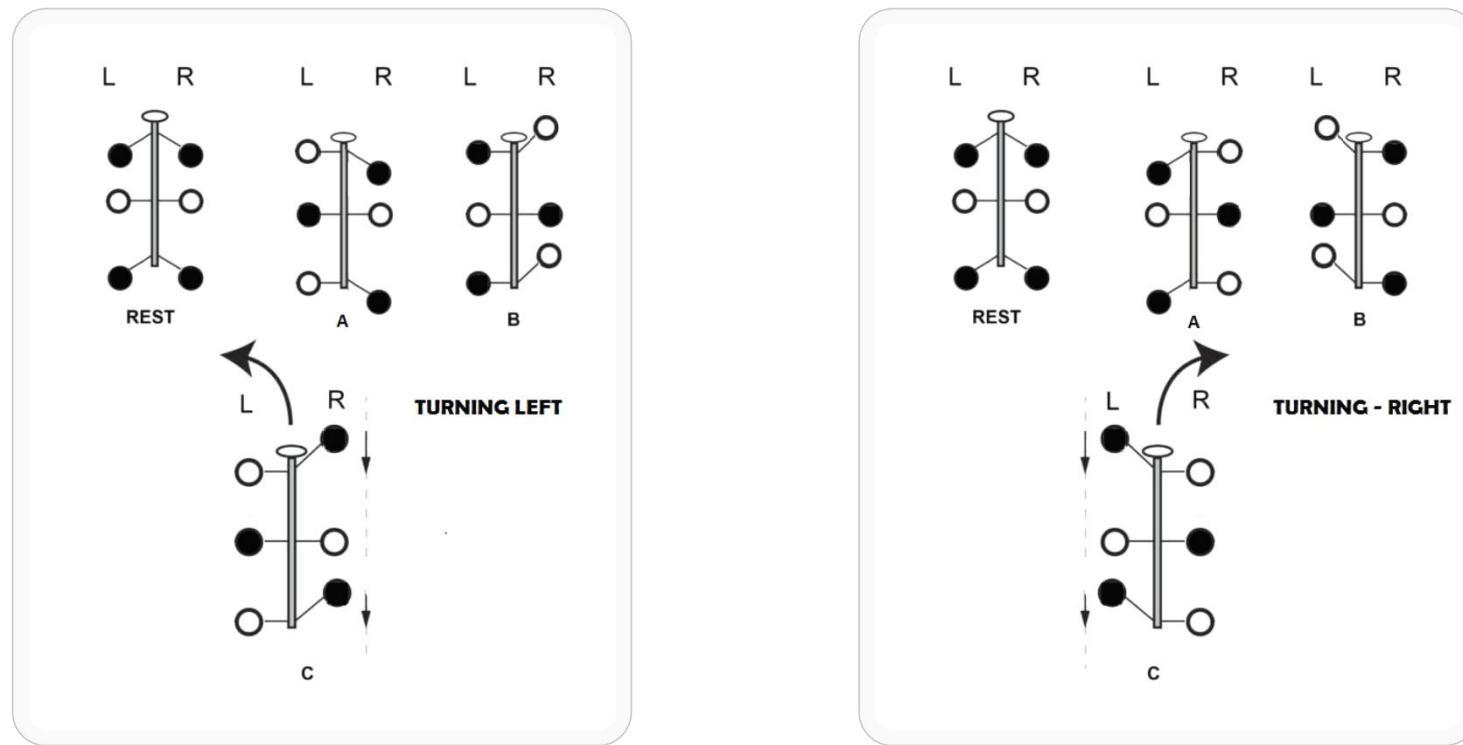
Algoritam ponašanja

Srednje nogu se pokreću pomoću jednog servo motora, dok su prednja i zadnja noga na lijevoj, odnosno desnoj strani spojene, i pokreću se pomoću svojih servo motora. Svaka noga je predstavljena pomoću kruga. Krug koji je obojen crnom bojom označava nogu koja je na podu, dok je bijelom bojom označena noga koja se nalazi u vazduhu.

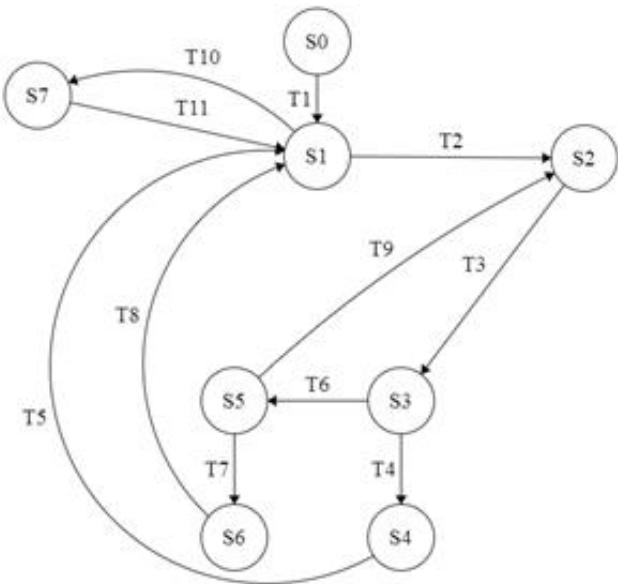


Algoritam ponašanja

Pozicija REST označava početno stanje, dok se kroz ostala stanja vrši pomjeranje odgovarajućih nogu. **Osim ova tri servo motora koji služe za pokretanje hexabota, četvrti servo motor se koristi za pokretanje glave, odnosno ultrasoničnog senzora.**

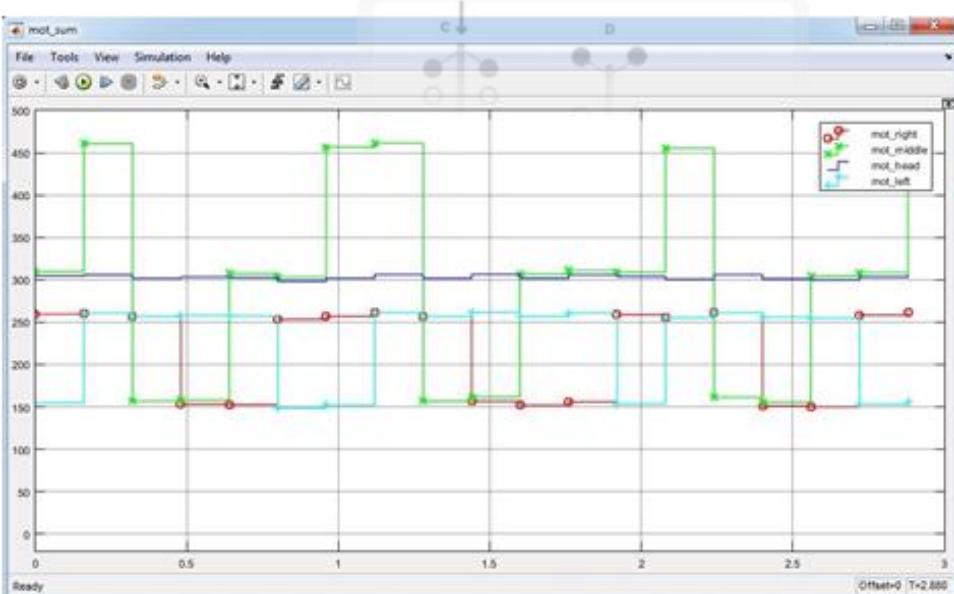
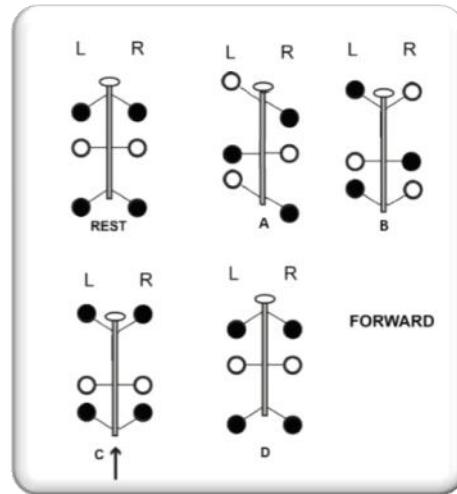


Mašina konačnog stanja



States	Description
S0	Standby state
S1	Normal walking
S2	Backward walking
S3	Turn head to the right
S4	Turn right
S5	Turn head to the left
S6	Turn left
S7	Over a rough terrain
Transitions	Description
T1	Turn the main switch on; Switch to Normal walking mode
T2	Obstacle detected; Switch to Backward mode
T3	Go three step in reverse; Switch to Turn head right mode
T4	Obstacle is not detected; Switch to Turn right mode
T5	Turning right is finished; Switch to Normal walking mode
T6	Obstacle is detected; Switch to Turn head left mode
T7	Obstacle is not detected; Switch to Turn left mode
T8	Turning left is finished; Switch to Normal walking mode
T9	Obstacle is detected; Switch to Backward mode
T10	Rough terrain is detected; Switch to Over a rough terrain mode
T11	Rough terrain is not detected; Switch to Normal walking mode
*	From each state we can switch to state S0 if the main switch is off.

Eksperimentalni rezultati navigacije heksapoda

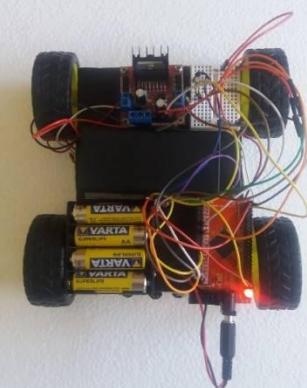
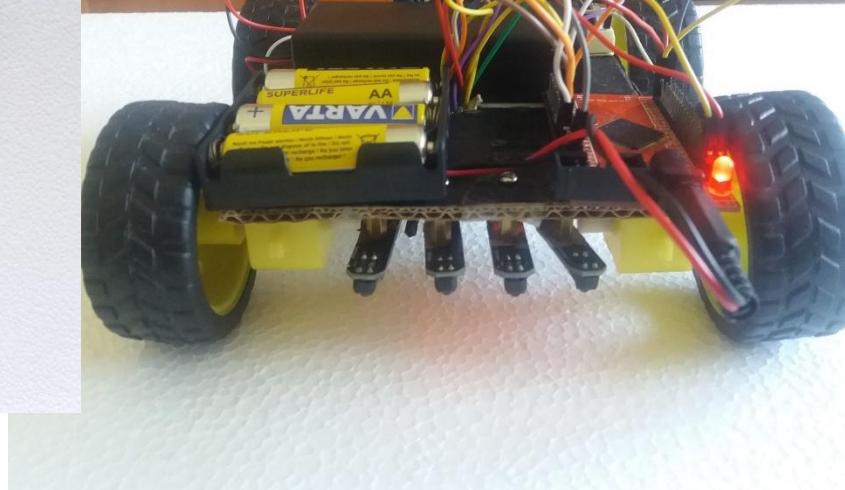
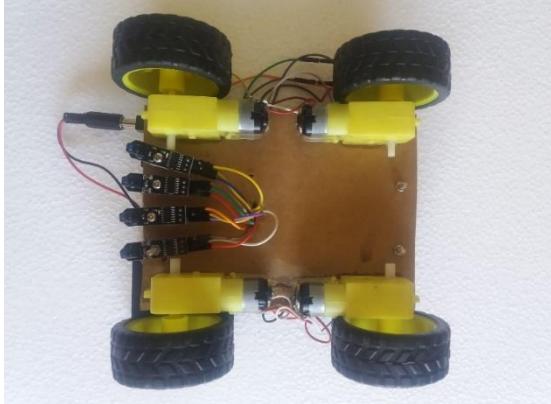


FORWARD			BACKWARD			
	Left motor	Middle motor	Right motor	Left motor	Middle motor	Right motor
1.	Bottom (150)	Middle (300)	Bottom (255)	Bottom (150)	Middle (300)	Bottom (255)
2.	Top (255)	Left (450)	Bottom (255)	Top (255)	Middle (300)	Top (150)
3.	Top (255)	Right (150)	Top (150)	Top (255)	Right (150)	Bottom (255)
4.	Top (255)	Middle (300)	Top (150)	Bottom (150)	Left (450)	Bottom (255)
5.	Bottom (150)	Middle (300)	Bottom (255)	Bottom (150)	Middle (300)	Bottom (255)
TURNING LEFT			TURNING RIGHT			
	Left motor	Middle motor	Right motor	Left motor	Middle motor	Right motor
1.	Bottom (150)	Middle (300)	Bottom (255)	Bottom (150)	Middle (300)	Bottom (255)
2.	Middle (200)	Left (450)	Bottom (255)	Bottom (150)	Right (150)	Middle (200)
3.	Middle (200)	Right (150)	Top (150)	Top (255)	Left (450)	Middle (200)
4.	Middle (200)	Left (450)	Top (150)	Top (255)	Right (150)	Middle (200)

Primjer 3: Slijedenje linije mobilnog robota korištenjem PID kontrolera

- *Zadaci projekta:*
 1. Konstruisati mobilnog robota sa 4 točka.
 2. Ugraditi senzore za praćenje linije.
 3. U programskom jeziku Verilog isprogramirati upravljanje mobilnog robota koristeći programsko okruženje Quartus II i prenijeti ga na pločicu FPGA.
 4. U logiku upravljanja integrisati PID kontroler.
 5. Napraviti maketu koja će omogućiti prikaz funkcionalnosti projekta.

Izgled mobilnog robota



Slijedenje linije mobilnog robota korištenjem PID kontrolera

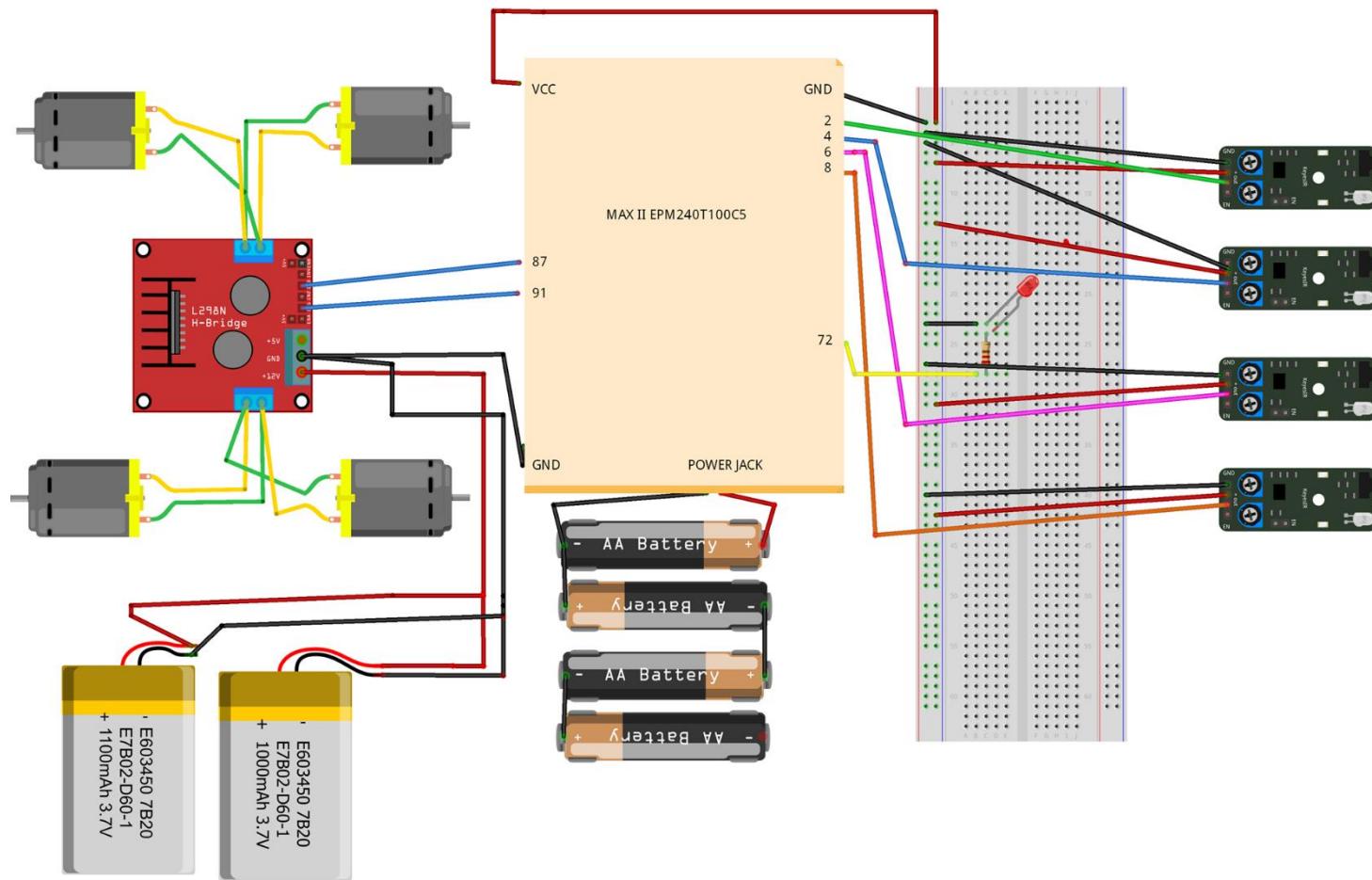
Mobilni robot za kretanje koristi 4 DC motora sa odgovarajućim točkovima. Budući da je korišten i H-most, dva DC motora su povezana na jednu, a druga dva su povezana na drugu stranu H-mosta. Za slučaj kada mobilni robot ide pravo, dva QTI senzora očitavaju crnu liniju, a druga dva očitavaju bijelu podlogu. U trenutku kada sva četiri QTI senzora očitaju bijelu podlogu mobilni robot se zaustavlja.

Cilj korištenja PID regulatora u ovom projektu jeste da se ostvari glatko praćenje crne linije kako ne bi došlo do naglih pokreta mobilnog robota. Putanja kretanja je nastala na osnovu promatranja prevoženja opasnih materija ili ljudi kamionima koji se voze po nekim od najopasnijih cesta na svijetu kao što su National Highway 22 ili Kinnaur Road u Indiji. Ove ceste imaju jako oštре zavoje u obliku slova S.

Izgled makete



Šema spajanja



fritzing

Projektovanje sistema
na čipu 3

Copyright: Lejla Banjanovic-
Mehmedovic

Pin planer

Pin Planner - C:/Users/Nerma/Desktop/Nerma podaci/FETIV/PSC/NermaPSC/test - test

File Edit View Processing Tools Window Help 

Report Report not available

Groups Report

Tasks

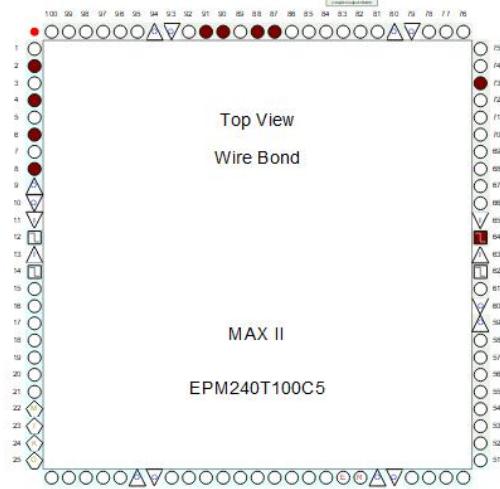
- Run Anal
- Early Pin
- Early
- Run
- Expo
- Change
- Show
- Show >

Named: * Edit: 

All Pins

MAX II
EPM240T100C5

Top View
Wire Bond



Node Name	Direction	Location	I/O Bank	Fitter Location	I/O Standard	Reserved	Current Strength
in_QT1	Input	PIN_2	1	PIN_2	3.3-V LV...default)		16mA (default)
in_QT12	Input	PIN_4	1	PIN_4	3.3-V LV...default)		16mA (default)
in_QT13	Input	PIN_6	1	PIN_6	3.3-V LV...default)		16mA (default)
in_QT14	Input	PIN_8	1	PIN_8	3.3-V LV...default)		16mA (default)
in_clk	Input	PIN_64	2	PIN_64	3.3-V LV...default)		16mA (default)
out_led	Output	PIN_73	2	PIN_73	3.3-V LV...default)		16mA (default)
out_motors_PWM[3]	Output	PIN_88	2	PIN_88	3.3-V LV...default)		16mA (default)
out_motors_PWM[2]	Output	PIN_87	2	PIN_87	3.3-V LV...default)		16mA (default)
out_motors_PWM[1]	Output	PIN_90	2	PIN_90	3.3-V LV...default)		16mA (default)
out_motors_PWM[0]	Output	PIN_91	2	PIN_91	3.3-V LV...default)		16mA (default)

Filter: Pins: all

0% 00:00:00

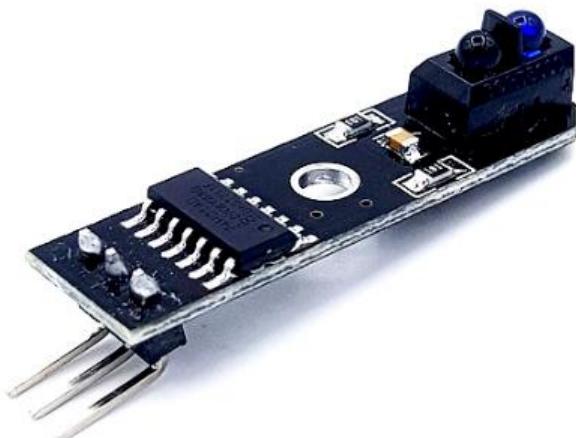
Upravljačka ploča



FPGA MAX II: EPM240T100C5

EPM240 mini ploča je idealna za niskobudžetne eksperimente i projekte. Na sredini ploče se nalazi Alterin MAX II EPM240TC5 CPLD. Zahtijeva USB blaster za programiranje. MAX II porodica CPLD-ova ima od 240 do 2.210 logičkih elemenata (LE). MAX II uređaji nude veliki broj I/O-a te brze performanse. MAX II uređaji dizajnirani su tako da reduciraju trošak i snagu.

Senzori



4 QTI senzora

Koristi se za praćenje crne linije. Sastoji se od IR LED i IR fotodiode. Infracrvena svjetlost koju emitira LED udara od površinu i odbija se nazad u fotodiodu. Fotodioda tada daje izlazni napon proporcionalan obojnosti površine (visoka vrijednost za bijelu, niska vrijednost za crnu površinu).

Ima 3 pina, a to su:

- Vcc(5V),
- GND,
- Digital output.

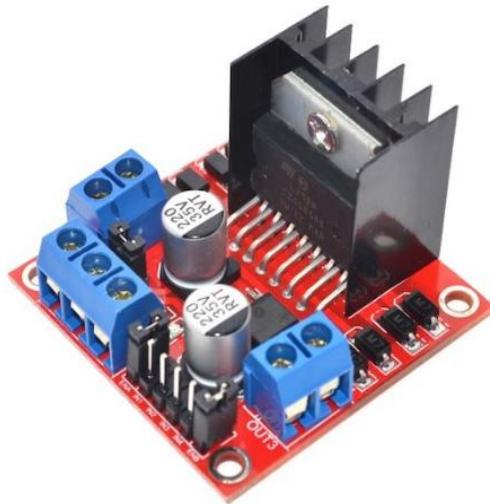
Izlazi



4 DC motora i 4 točka

Svrha ovih motora je pretvaranje električne energije u mehaničku i električne struje u mehaničke pokrete. Istosmjerni motor sastoji se od tri glavna dijela: statora, rotora i komutatora (kolektora i četkica).

Drajveri motora



- Driveri su elektronski sklopovi koji sastavljaju sve potrebne komponente za ispravan rad motora u jedan sklop. **Pomoću drivera moguće je kontrolirati brzinu i smjer okretanja motora.**
- Jedan od njih je i L298N driver koji se sastoji od dva H-mosta koji služe za promjenu smjera svakog pojedinog motora. Sastoji se od 4 prekidača, koji su inače u spojevima tranzistori, a između dva tranzistora je spojen elektromotor.

L298N Motor Driver

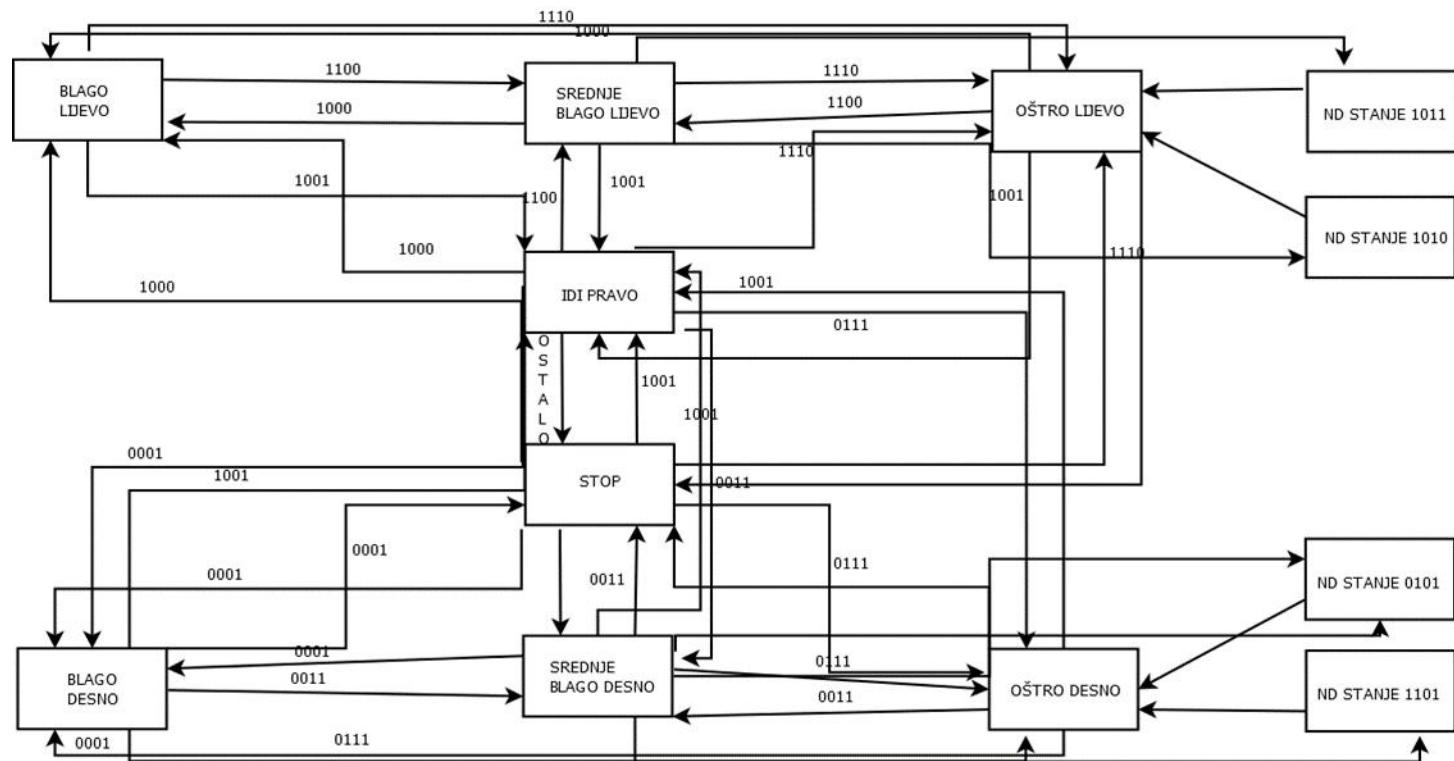
DC motori i L298N driver spojeni su na šasiju mobilnog robota. Pružaju potrebno kretanje na način da su dva desna motora spojena na ulaz IN1, a dva lijeva na ulaz IN3. Motori su prethodno zalemljeni unakrsno i kao takvi spojeni na driver.

Opis rada mobilnog robota

Četiri QTI senzora: prilikom kretanja mobilnog robota pravo naprijed oni rade po sljedećoj logici: dva QTI senzora očitavaju bijelu podlogu, dok druga dva prate crnu liniju. U zavisnosti od očitanja senzora mijenjaju se i brzine lijevih i desnih motora. U nastavku je data tabela koja prikazuje sva moguća stanja.

QTI1	QTI2	QTI3	QTI4	Desni motori	Lijevi motori	Opis
1	0	0	1	45	45	Ide pravo
0	0	0	1	50.6	0	Blago desno skreće s linije
0	0	1	1	61.8	0	Srednje blago desno skreće s linije
0	1	1	1	67.4	0	Oštro desno skreće s linije
1	1	1	1	0	0	U potpunosti skreće s linije
1	0	0	0	0	50.6	Blago lijevo skreće s linije
1	1	0	0	0	61.8	Srednje blago lijevo skreće s linije
1	1	1	0	0	67.4	Oštro lijevo skreće s linije
1	1	0	1	61.8	0	Skreće lijevo s linije
1	0	1	1	0	61.8	Skreće desno s linije
1	0	1	0	0	61.8	Skreće lijevo s linije
0	1	0	1	61.8	0	Skreće desno s linije

Mašina konačnog stanja



PID kontroler

$$u(t) = G_{PID}(s) = K_p e(t) + K_d \frac{de(t)}{dt} + K_i \int_{-\infty}^t e(t) dt$$
$$u[k] = u[k - 1] + K_1 e[k] + K_2 e[k - 1] + K_3 e[k - 2]$$

$$K_1 = K_p + K_i + K_d$$

$$K_2 = -K_p - 2K_d$$

$$K_3 = K_d$$

Mi smo odabrali sledeće vrijednosti koeficijenata:

$$K_p = 3 \quad K_d = 2 \quad K_i = 6/10$$

Te u skladu sa formulama dobijamo:

$$K_1 = 5.6$$

$$K_2 = -7$$

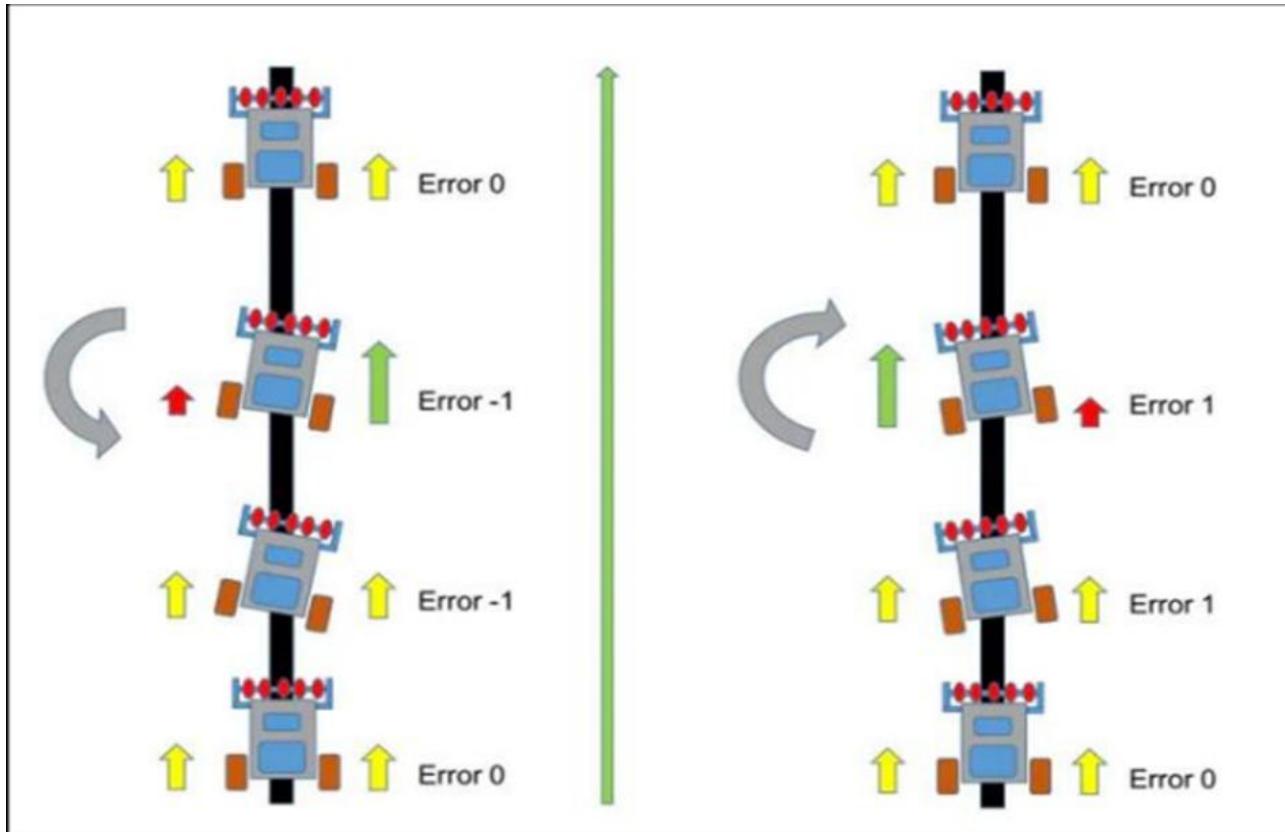
$$K_3 = 2$$

Opis rada mobilnog robota

Svakom očitanju senzora dodjeljuje se vrijednost greške $e(t)$ i to tako da manju grešku imaju očitanja koja manje skreću sa linije, a ona koja više skreću imaju veću vrijednost greške. Upravo ovdje možemo vidjeti prednost korištenja 4 QTI senzora umjesto npr. 3, jer omogućavaju veći broj različitih stanja i veći broj vrijednosti grešaka. Tada postoje manje razlike brzina između npr. skretanje blago desno i skretanje srednje blago desno te glatke prelaze sa jednog na drugo stanje. U nastavku je prikazana tabela sa očitanjima senzora i njima pridružene vrijednosti greške.

Očitanje senzora	Opis kretanja robota	Vrijednost greške
1 0 0 1	Ide pravo	0
0 0 0 1	Blago skreće desno s linije	1
0 0 1 1	Srednje blago skreće desno s linije	3
0 1 1 1	Oštro skreće desno s linije	4
1 0 0 0	Blago skreće lijevo s linije	1
1 1 0 0	Srednje blago skreće lijevo s linije	3
1 1 1 0	Oštro skreće lijevo s linije	4
1 1 0 1	Skreće lijevo s linije	3
1 0 1 1	Skreće desno s linije	3
1 0 1 0	Skreće lijevo s linije	3
0 1 0 1	Skreće desno s linije	3
Ostale kombinacije	Zaustavlja se	-

Opis rada mobilnog robota



Rezultati navigacije

