ELEKTROTEHNIČKI FAKULTET U BEOGRADU

Katedra za elektroniku

Predmet: 32_bitni mikrokontroleri i primena

Projekat: Ovlašćeno upravljanje step motorom

Student:

Nemanja Kundović 3328/2011

e-mail: psyetf@gmail.com

Profesor: dr Dragan Vasiljević

Asistent: mr Nenad Jovičić

Beograd, januar 2013.

Sadržaj:

1. Uvod

Potrebno je realizovati sistem koji će omogućiti upravljanje step motorom korisnicima koji poseduju *RFID* kartice sa odgovarajućim tagom. Ovakav sistem može naći primenu za svako ovlašćeno upravljanje elektronskim sklopovima. Cilj projekta jeste kontrolisanje pozicije ili brzine step motora u otvorenoj sprezi (*closed loop*). Aktuator kod CNC mašina je najčešće step motor. Zbog velike rezolucije (visoke tačnosti pozicioniranja) i relativno jednostavne kontrole, ovakvi motori nalaze sve češće primenu u industriji. Zbog zahteva tržišta za što bržom proizvodnjom novih proizvoda, velike firme zapošljavaju ogroman broj ljudi koje treba grupisati prema zadacima. Da ne bi došlo do neovlašćene kontrole nad nekim delom sistema, potrebno je uvesti neki nivo zaštite. Jedan način je unos šifri pomoću tastature. Drugi način jeste korišćenje *RFID* kartica. Svaka kartica na svetu ima jedinstven kod, tako da je jedinstvenost pristupa obezbedjena. Ove kartice se danas često koriste za ulaz u odredjene poslovne komplekse, naplatu vožnje, itd.

2. Opis sistema

Potrebno je napraviti sistem koji će očitavati RFID kartice, upravljati step motorom, ispisivati tekst na inteligentnom LCD displeju i primati komande preko tastature. Samo korisnicima čije su kartice registrovane u sistemu će biti omogućeno upravljanje step motorom. Komande za regulaciju pozicije ili brzine motora će se zadavati preko tastature i biće ispisivane na inteligentnom LCD displeju. Cilj sistema jeste očitavanje RFID kartica i tastature, slanje komandi step motoru i slanje teksta za ispis na LCD displej.

Komponente našeg sistema su:

- Mikrokontroler *STM32F100RB* sa svojim (internim) periferijama
- Bipolarni step motor, kao i drajver za bipolarni step motor – integrisano kolo A3967SLB
- PC tastatura koja komunicira sa mikrokontrolerom preko PS/2 porta
- Inteligentni alfanumerički *LC* displej
- RFID čitač

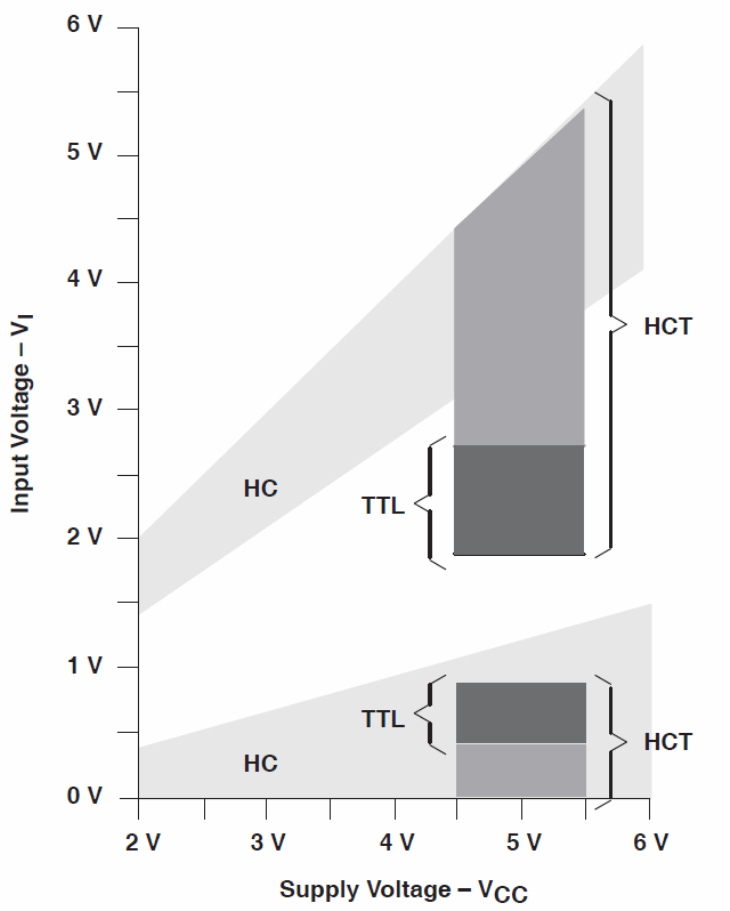
Glavna komponenta našeg sistema jeste mikrokontroler kompanije *ST Microelectronics* - STM32F100RB. Ovaj mikrokontroler ima ugradjen 32-bitni *Cortex-M3* procesor (arhitektura *ARMv7-M*). Glavna karakteristika ovog mikrokontrolera jeste niska cena i prilagodjenost korišćenja za industrijske aplikacije. Instrukcijski set je *Thumb-2*, pomoću kojeg je povećana efikasnost mikrokontrolera. Sa ovim instrukcijskim setom nema gubljenja vremena za prebacivanje stanja iz *ARM state* u *Thumb state* i obratno. Pomenuti mikrokontroler se nalazi na razvojnoj ploči *STM32VLDISCOVERY*.

Ova pločica ima veoma malo eksternih periferija na sebi (2 tastera i 2 *LE* diode), medjutim veoma je pogodna za povezivanje različitih periferija. Pločica se napaja pomoću USB-a (+5V). Na njoj se nalazi integrisano kolo *L1117 - LDO (Low Drop Out)* regulator napona koji pravi referencu napona od 3.3V za napajanje mikrokontrolera. Za debagovanje koda mikrokontrolera koristi se integrisano kolo koje omogućuje *SWD (Serial Wire Debug)*.

Mikrokontroler ima zadatak da očitava podatke koji mu stižu od strane tastature i RFID čitača, obradi dobijene podatke i zatim generiše signale za upravljanje step motorom i ispisivanje na *LCD*.

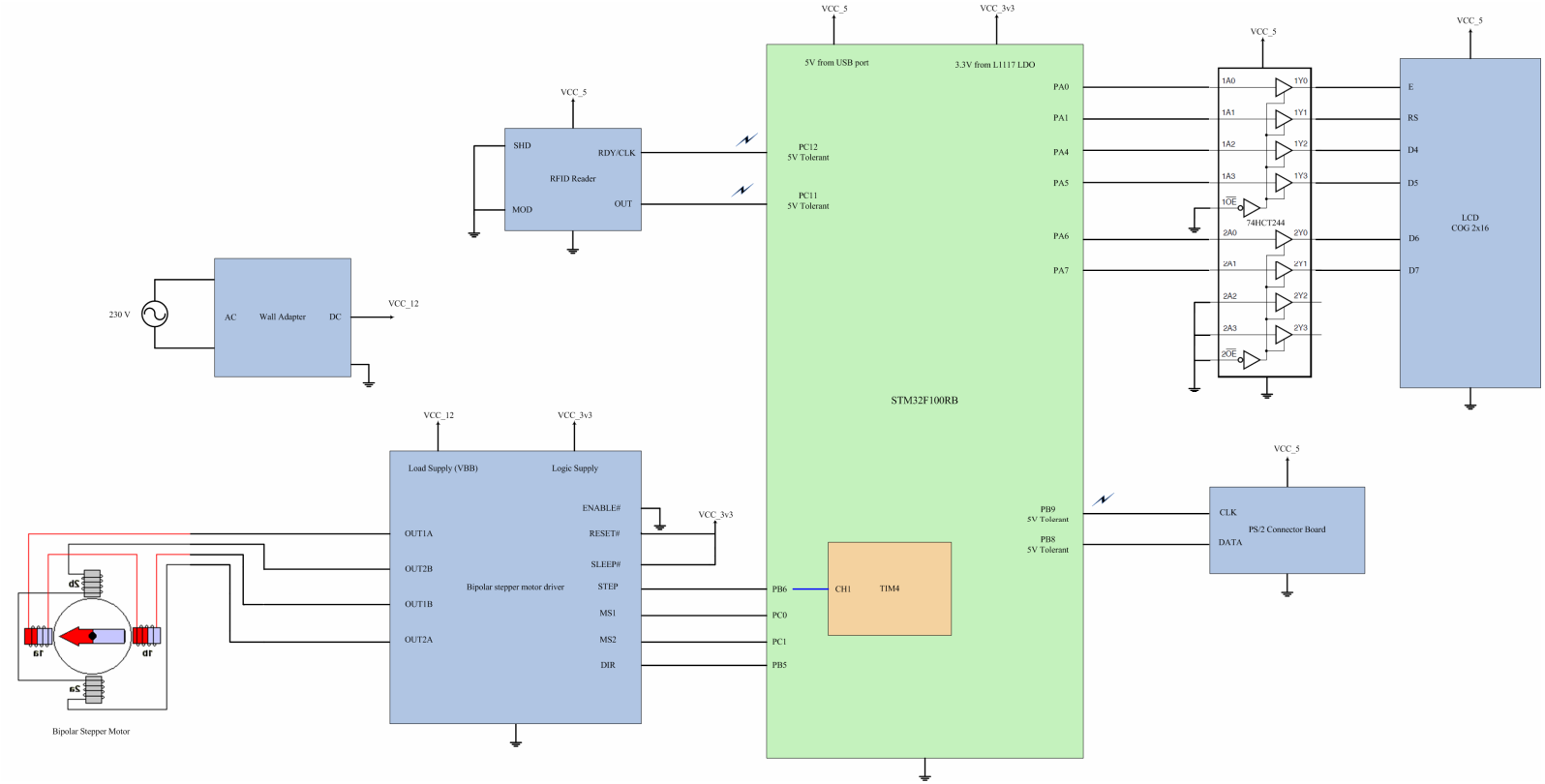
3. Koncept realizacije zadatka

Pomoću pločice *STM32VLDISCOVERY* imamo mogućnost napajanja periferija sa 3.3V ili 5V. Tastatura se napaja sa 5V i naponski nivoi signala koje ona generiše su u skladu sa CMOS(5V). Naš mikrokontroler se napaja sa 3.3V i njegovi naponski nivoi su u skladu sa standardom LVCMOS33. Mikrokontroler STM32F100RB ima mogućnost povezivanja sa periferijama koje su u skladu sa standardom CMOS(5V). Naime odredjeni pinovi su 5V *Tolerant*. Iz ovih razloga, tastaturu povezujemo na 5V *Tolerant* pinove PB.8 i PB.9. RFID čitač se takodje napaja sa 5V, pa su i njegovi signali povezani na 5V *Tolerant* pinove mikrokontrolera - PC.11 i PC.12. Inteligentni alfanumerički LC displej se napaja sa 5V, dok naš mikrokontroler generiše signale u skladu sa LVCMOS33 standardom, pa nam je za pobudu LC displeja potreban translator nivoa. Za ovu svrhu je iskorišćeno integrisano kolo iz familije HCT zbog velike margine šuma, kao što je prikazano na slici 1. Odabrano je kolo HCT244, koje ima mogućnost translacije do 8 ulaza. Bipolarni step motor se napaja sa 12V. Ovaj napon je dobijen pomoću mrežnog adaptera. Digitalna logika za drajvovanje step motora se napaja sa napajanjem od 3.3V pa nam nije potreban translator naponskog nivoa.



Slika 1. Logički nivoi HCT familije

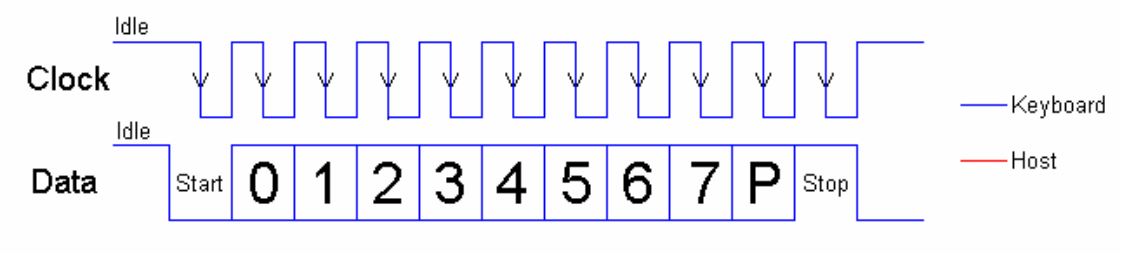
Kompletna električna šema sistema je prikzana na slici 2. Oznaka groma označava ulazne signale na čiju promenu (uzlaznu i/ili silaznu ivicu) dolazi do generisanja prekida u mikrokontroleru. U daljem tekstu su detaljnije opisane periferije a zatim i princip funkcionisanja celog sistema.



Slika 2. Električna šema sistema

3.1 Tastatura

Tastatura pored napajanja, ima još 2 pina. Jedan signal je takt (Clk) koji je povezan na pin PB.9. Takt je aktivan samo prilikom slanja pritisnutog karaktera, inače se nalazi u *idle* stanju (logička jedinica). Pin PB.9 mikrokontrolera je konfigurisan kao ulazni *pull-up*. Omogućeni su prekidi na silaznu ivicu signala PB.9. Drugi signal je podatak (Data). Protokol očitavanja karaktera se sastoji u očitavanju Data linije na silaznu ivicu Clk signala. Format podataka je prikazan na slici 3.

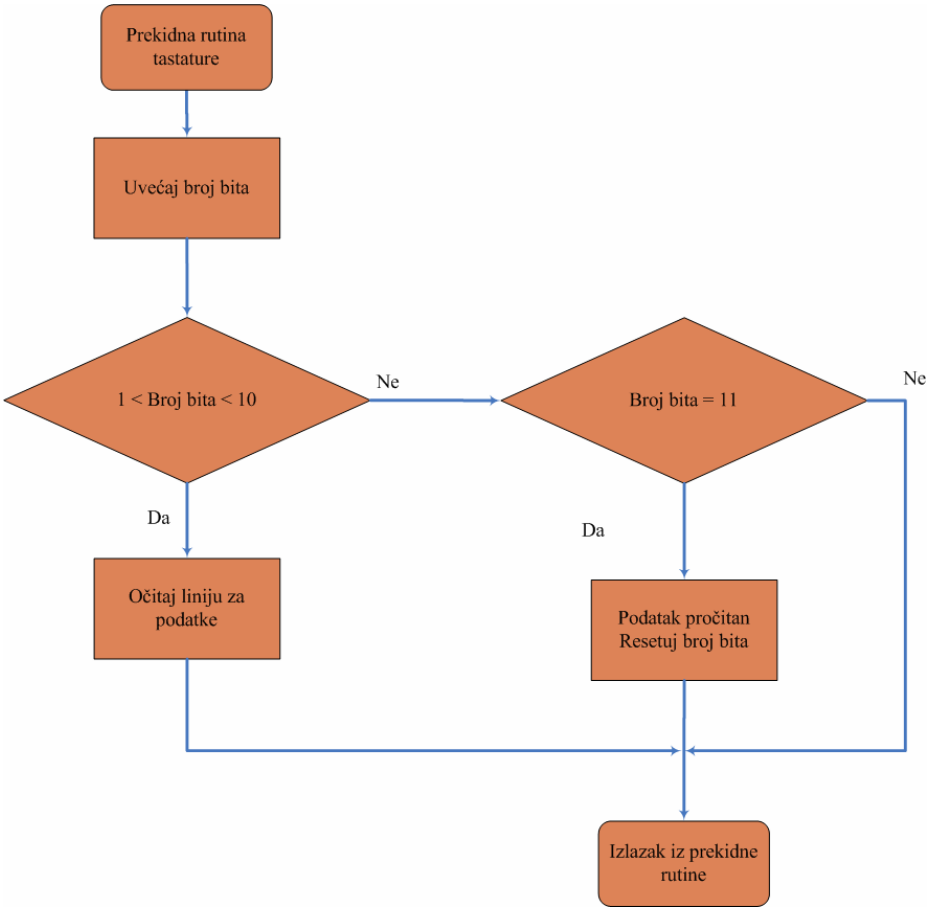


Slika 3. Format podataka tastature

Start *frame*-a se detektuje na silaznu ivicu takta. Nakon toga se u prekidnoj rutini očitavaju podaci D0-D7. Tastatura generiše make i break kodove. U okviru implementirane prekidne rutine, aktivne na silaznu ivicu Clk signala, očitavaju se svi podaci poslati od strane tastature i prosledjuju glavnom programu. Glavni program filtrira break kodove i za validne koristi samo make kodove. Algoritam očitavanja tastature je prikazan na slici 4. Pri ulasku u prekidnu rutinu inkrementira se broj bita. Sa slike vidimo da informacioni biti kreću od indeksa 2 i traju do indeksa 9. Iz ovoga zaključujemo da je potrebno očitavati liniju podataka ukoliko je indeks veći od 1 i manji od 10. Indeks stop bita je 11. Kada je broj bita jednak 11, očitanih 8 informacionih bita se prosledjuje glavnom programu a brojač bita se resetuje. Unutar glavnog programa se filtriraju break kodovi. Ukoliko je prethodni bajt bio 0xF0, sledeći bajt se ignoriše. U suprotnom bi na jedan pritisak tastera dobili informaciju kao da je dvaput pritisnut. U tabeli 1. su prikazana značenja tastera.

Tabela 1. Znacenje tastera

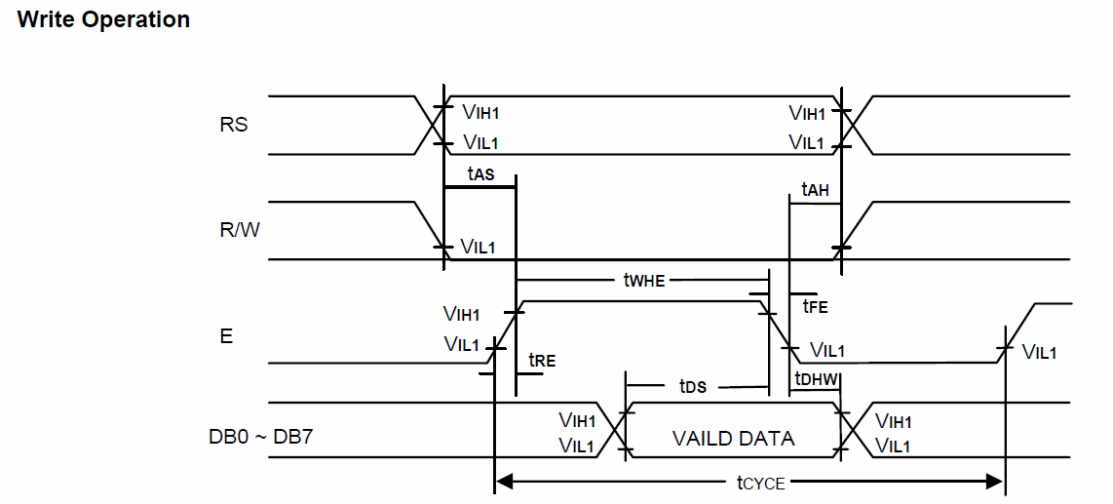
Taster	Znacenje
A	Dekrementiraj poziciju/brzinu
D	Ikrementiraj poziciju/brzinu
+	Definise smer CW okretanja
-	Definise smer CCW okretanja
1-4 (alpha numeric)	Definise korak motora (1, 1/2 , ¼, 1/8)
0-9 (numeric)	Unos zeljene pozicije/brzine
Enter	Potvrda zeljene brzine/pozicije
Esc	Izlazak iz trenutnog menija
C	Brisanje nevalidnog pristupa (plava led)



Slika 4. Algoritam očitavanja tastature

3.2 LCD

Mikrokontroler obavlja komunikaciju sa mikrokontrolerom preko pinova PA.0, PA.1, PA.4-PA7. Komunikacija sa LCD-om je prilično jednostavna i bazira se na vremenskom dijagramu prikazanom na slici 5. Problem predstavljaju kašnjenja koja treba formirati. Ona su formirana pomoću tajmera 2. Napravljen je algoritam da glavni program ne ostaje u petlji i čeka da prodje kašnjenje, već da za to vreme izvršava ostale instrukcije glavnog programa. Kada je potrebno osvežiti displej, umesto brisanja displeja, koristi se komanda vraćanja cursora na početnu poziciju, zato što je za izvršavanje te instrukcije potrebno mnogo manje vremena (40 us) nego za instrukciju brisanja displeja (1.64 ms). Displej se osvežava samo kada neki od potprograma setuje promenljivu refresh_display. Ova promenljiva se setuje u slučaju detekcije RFID kartice ili u slučaju pritiska tastera.



Slika 5. Vremenski dijagram LCD-a

3.3 RFID

RFID signali su povezani na pinove mikrokontrolera PC.12 (Rdy/Clk) i PC.11 (Out). Omogućeni su prekidi za oba signala. Protokol RFID čitača se sastoji u slanju 64 bita koji su kodirani *Manchester Encoding*-om. Svaki bit traje 64 perioda takta. Ukoliko se u tom periodu desila uzlazna ivica očitani bit je 1, a ukoliko se desila silazna ivica, očitani bit je 0. Ukoliko se nije desila ni silazna ni uzlazna ivica kartica nije prisutna. Sinhronizacija sa početkom slanja sekvence se zaključuje na osnovu prvih 9 bita koji su logičke jedinice, kodirane *Manchester Encoding*-om. Nakon očitanog koda, prekidna rutina RFID čitača setuje promenljivu data_read_done i prosledjuje glavnom programu ID kartice. Ukoliko se kartica nalazi u grupi validnih kartica, omogućuje se upravljanje step motorom pomoću tastature i aktivira se zelena LE dioda. U slučaju pokušaja pristupa sa nevalidnom karticom, aktivira se plava LE dioda.

3.4 STEP MOTOR

Step motor je inicijalno neaktivan. On je povezan sa mikrokontrolerom preko pinova: PC.0 (MS1), PC.1 (MS2), PB.5 (Dir), PB6 (Step). Na osnovu bitova MS1 i MS2 podešavamo korak step motora. Pun korak step motora je 7.5 stepeni. Odabir rezolucije step motora se vrši pomoću tastera 1, 2, 3 i 4 (alpha num).

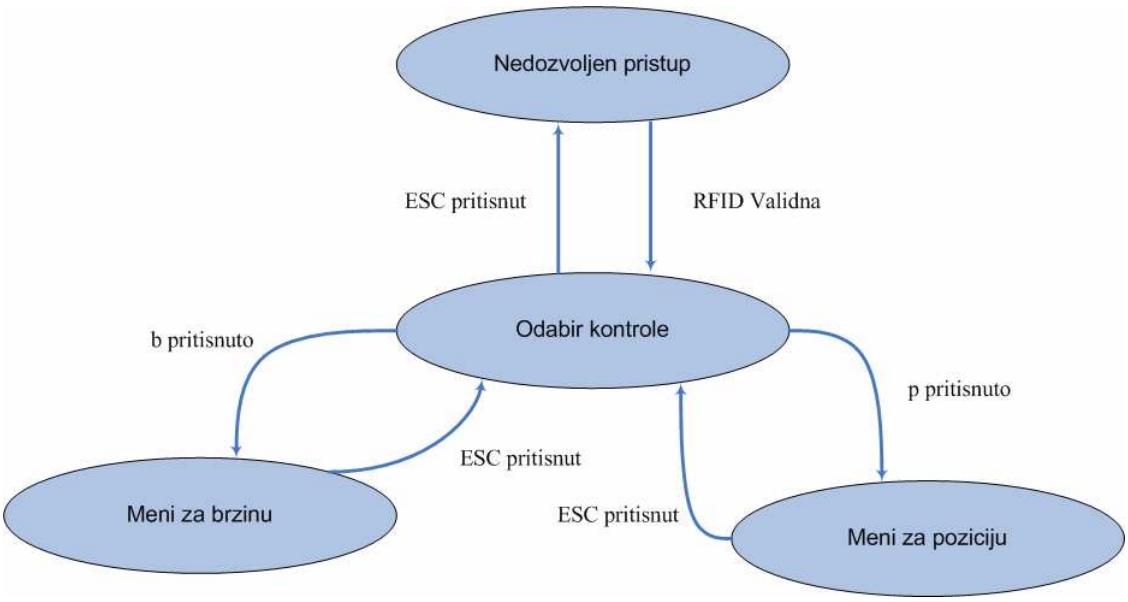
Tabela 2. Odabir rezolucije step motora

MS1	MS2	Rezolucija
0	0	Pun korak
1	0	Polukorak
0	1	¼ koraka
1	1	1/8 koraka

Smer se odredjuje na osnovu znakova + i -. Plus mora da se unese, tj. nije podrazumevan predznak. Ukoliko je zadata brzina dosta veća/manja od trenutne, motor postepeno ubrzava. Kontrola brzine motora se omogućava kontrolisanjem signala Step. Ovaj signal se generiše pomoću tajmera 4 – kanal 1. Kanal 1 je u PWM modu i njegov period se menja pri promeni brzine, što je tražena brzina veća to je period PWM-a kraći i obratno.

4. Mašina stanja

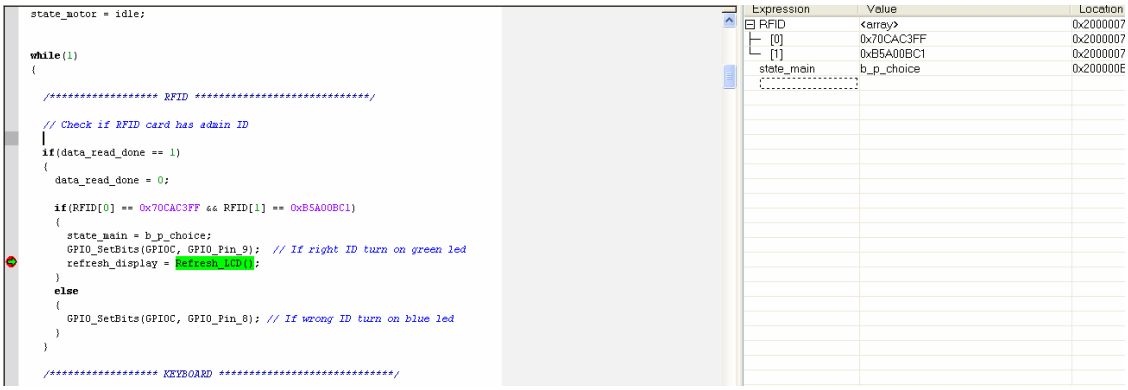
Glavna kontrola se obavlja pomoću mašine stanja. Inicijalno stanje mašine je "Nedozvoljen pristup". Nakon prinošenja validne RFID kartice mašina prelazi u stanje "Odabir kontrole". Ukoliko korisnik pritisne taster p, vrši se upravljanje pozicijom, a ukoliko pritisne taster b, upravlja se brzinom. Nakon postavljene pozicije/brzine korisnik može vratiti mašinu u stanje nedozvoljen pristup pritiskom na dugme Esc dva puta. Na osnovu ove mašine stanja, izmedju ostalog, vršimo osvežavanje displeja.



Slika 6. Glavna mašina stanja

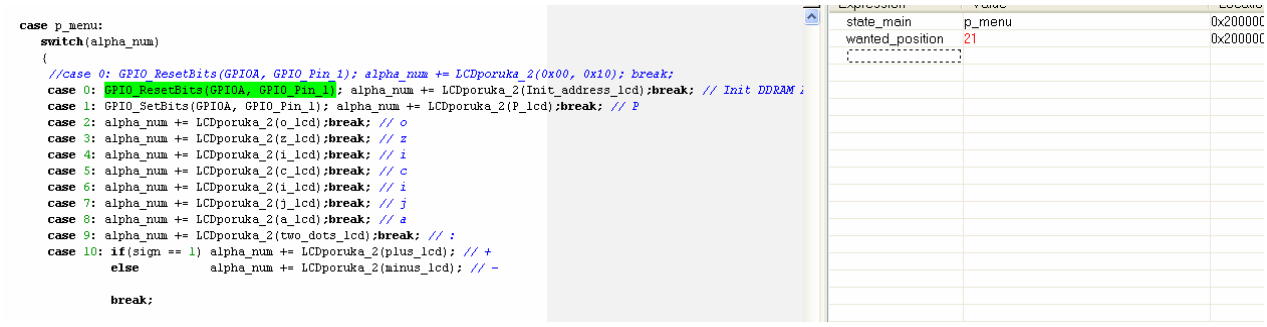
5. Rezultati

Interfejs sa svakim uredjajem je testiran zasebno u cilju bržeg nalaženja grešaka. Nakon toga implementiran je ceo sistem. Na slici 7 prikazana je situacija kada je očitana validna RFID kartica. U Watch prozoru su dodate promenljive – RFID, koja predstavlja ID kartice i promenljiva state_main koja je u stanju b_p_choice (izbor brzine ili pozicije), odakle vidimo da je glavna mašina stanja izašla iz stanja “Nedozvoljen pristup”.



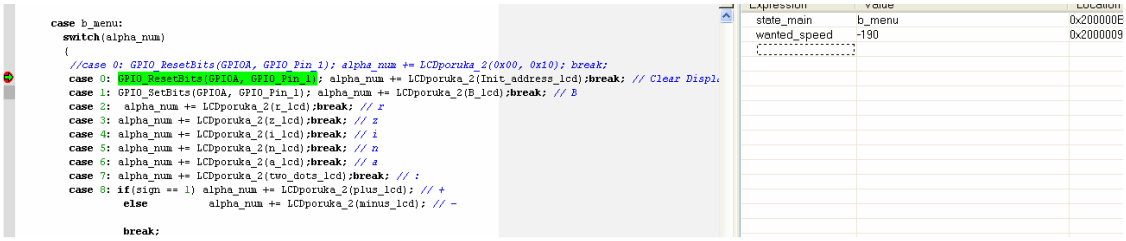
Slika 7. RFID validna

Na slici 8 prikazana je situacija kada je zadata pozicija +21 i displej se osvežava. U Watch prozoru su dodate promenljive – state_main koja je u stanju p_menu (meni za poziciju) i wanted_position = 21, tj. željena pozicija je +21



Slika 8. Biranje pozicije

Na slici 9 prikazana je situacija kada je zadata brzina -190 i displej se osvežava. U Watch prozoru su dodate promenljive – state_main koja je u stanju b_menu (meni za brzinu) i wanted_speed = -190, tj. željena pozicija je -190.



Slika 9. Biranje brzine

6. Zaključak

Na 32-bitnom mikrokontroleru implementiran je sistem za ovlašćeno upravljanje brzinom/pozicijom step motora. Unapredjenje projekta je moguće u smislu povezivanja sa računarom i beleženju logova i vremena, da bi postojao uvid ko je kada kontrolisao motor. Moguće unapredjenje je i dodavanje administratorskih privilegija u cilju modifikacije sistema.

Literatura:

[1] http://retired.beyondlogic.org/keyboard/keybrd.htm
[2] http://www.stepperworld.com/Tutorials/pgBipolarTutorial.htm
[3] http://www.geocities.com/dinceraydin/djlcdsim/djlcdsim.html
[4] http://www.priority1design.com.au/em4100_protocol.html

PRILOG – Izvorni kod programa

main.c

```
/* Includes ------------------------------------------------------------------*/
#include "stm32f10x.h"
#include "STM32vldiscovery.h"

#include "ConstsAndPrototypes.h"

/* Private typedef -----------------------------------------------------------*/

/* Private define ------------------------------------------------------------*/
/* Private macro -------------------------------------------------------------*/
/* Private consts ------------------------------------------------------------*/
/* Private variables ---------------------------------------------------------*/
/* Private function prototypes -----------------------------------------------*/
/* Private functions ---------------------------------------------------------*/


extern char karakter, new_data;

char komanda, activate_digits = 0, direction = 0, same_direction = 1, alpha_num = 0,
refresh_display = 1;

int broj, wanted_position = 0, wanted_speed = 0, current_speed = 0, sign = 1,
wanted_period = default_step_period, current_period = default_step_period;

int current_position=0, rel_position = 0, lcd_num = 0;

char cifra[4];


enum
{
   idle_key, new_command, ignore_next
} state_keyboard;

enum
{
   access_denied, b_p_choice, b_menu, p_menu
} state_main;
```

```
state_motor_type state_motor;

char Timer_IF = 0;

extern uint32_t RFID[2];

extern char data_read_done;


int main (void)
{

  Init_keyboard();
  Init_stepper_motor();
  Init_LCD();
  Init_RFID();

  state_keyboard = idle_key;
  state_main = access_denied;
  state_motor = idle;


  while(1)
  {

    /****************** RFID ****************************/

    // Check if RFID card has admin ID

    if(data_read_done == 1)
    {
      data_read_done = 0;

      if(RFID[0] == 0x70CAC3FF && RFID[1] == 0xB5A00BC1)
      {
        state_main = b_p_choice;
        GPIO_SetBits(GPIOC, GPIO_Pin_9);  // If right ID turn on green led
        refresh_display = Refresh_LCD();
      }
      else
      {
        GPIO_SetBits(GPIOC, GPIO_Pin_8); // If wrong ID turn on blue led
      }
    }
```

```
/****************** KEYBOARD *****************************/


switch(state_keyboard)
{
case idle_key:
 if(state_main == access_denied) break;

  if(new_data == 1)
  {
    state_keyboard = new_command;
    new_data = 0;
    komanda = karakter;
    karakter = 0x00;
  }

  break;

case new_command:

        refresh_display = Refresh_LCD();

  switch(komanda)
  {

   case Break_Code: state_keyboard = ignore_next; break;

   case C_key:

     GPIO_ResetBits(GPIOC, GPIO_Pin_8);
     state_keyboard = idle_key;
     break;

   case P_key:


     if(state_main == b_p_choice)
     {
       state_main = p_menu;
       TIM_Cmd( TIM4, DISABLE );
       current_speed = 0;
       wanted_speed = 0;
     }

     state_keyboard = idle_key;
```

```
  break;

case B_key:


  if(state_main == b_p_choice)
  {
    state_main = b_menu;
    wanted_position = 0;
    current_position = 0;
  }

  state_keyboard = idle_key;
  break;

case D_key:


  state_keyboard = idle_key;

 // if(rel_position != 0) break;

  if(activate_digits == 0)
  {

  if(state_main == p_menu)
    {
      wanted_position++;

      if(wanted_position > max_pos_speed) wanted_position = max_pos_speed;

      if(wanted_position > 0) sign = 1;

      state_motor = update_pos;

    }

  if(state_main == b_menu)
    {
      wanted_speed+=10;

      if(wanted_speed > max_pos_speed) wanted_speed = max_pos_speed;

      if(wanted_speed > 0) sign = 1;
```

```
    state_motor = update_speed;


   }
  }


  break;

case A_key:

  state_keyboard = idle_key;

 // if(rel_position != 0) break;

  if(activate_digits == 0)
  {

  if(state_main == p_menu)
    {
     wanted_position--;

     if(wanted_position< -max_pos_speed) wanted_position = -max_pos_speed;

      if(wanted_position < 0) sign = -1;

     state_motor = update_pos;


    }

  if(state_main == b_menu)
    {
     wanted_speed-=10;

     if(wanted_speed < -max_pos_speed) wanted_speed = -max_pos_speed;

     if(wanted_speed < 0) sign = -1;

     state_motor = update_speed;


    }
  }
```

```
  break;

case Plus_key:


  if((state_main == b_menu || state_main == p_menu) && activate_digits == 0)
  {
    activate_digits = 1;
    sign = 1;
    broj = 0;

  }

  state_keyboard = idle_key;
  break;

case Minus_key:


  if((state_main == b_menu || state_main == p_menu) && activate_digits == 0)
  {
    activate_digits = 1;
    sign = -1;
    broj = 0;

  }

  state_keyboard = idle_key;
  break;

case Enter_key:


  state_keyboard = idle_key;

 // if(rel_position != 0) break;

  if(activate_digits == 1)
  {
    activate_digits = 0;

    if(broj > max_pos_speed) broj = max_pos_speed;
```

```
    broj *= sign;

    if(state_main == p_menu)
    {
      wanted_position = broj;
      state_motor = update_pos;
    }

    if(state_main == b_menu)
    {
      wanted_speed = broj;
      state_motor = update_speed;
    }


  }


  break;

case Esc_key:



  state_keyboard = idle_key;

 // if(rel_position != 0) break;


  if(activate_digits == 1)
  {
    activate_digits = 0;
    if(wanted_position < 0 || wanted_speed < 0)
    {
      sign = -1;
    }
    else
    {
      sign = 1;
    }

  }
  else
  {

  if(state_main == b_p_choice)
```

```
  {
     state_main = access_denied;
     GPIO_ResetBits(GPIOC, GPIO_Pin_9);
  }

  if(state_main == b_menu || state_main == p_menu)
  {
   activate_digits = 0;
   broj = 0;
   state_main = b_p_choice;
   //direction = 0;
   //GPIO_ResetBits(GPIOB, GPIO_Pin_5);

   state_motor = idle;
   TIM_ITConfig(TIM4,TIM_IT_Update, ENABLE);
  }


  }



  break;

case Bkspc_key:



  if(activate_digits == 1)
  {
   broj /= 10;

   //if(broj == 0)
   //{
   //  activate_digits = 0;
   //}

  }

  state_keyboard = idle_key;
  break;

case an1_key:

  GPIO_ResetBits(GPIOC, GPIO_Pin_0 | GPIO_Pin_1);
```

```
  state_keyboard = idle_key;
  break;

case an2_key:

  GPIO_ResetBits(GPIOC, GPIO_Pin_1);
  GPIO_SetBits  (GPIOC, GPIO_Pin_0);

  state_keyboard = idle_key;
  break;

case an3_key:


 GPIO_ResetBits(GPIOC, GPIO_Pin_0);
 GPIO_SetBits  (GPIOC, GPIO_Pin_1);

  state_keyboard = idle_key;
  break;

case an4_key:

  GPIO_SetBits  (GPIOC, GPIO_Pin_0 | GPIO_Pin_1);

  state_keyboard = idle_key;
  break;


 case d0_key:



  if(activate_digits == 1)
  {
    broj *= 10;
    broj += 0;

    if(broj > max_pos_speed) broj = max_pos_speed;
  }

  state_keyboard = idle_key;
  break;

 case d1_key:
```

```
  if(activate_digits == 1)
  {
    broj *= 10;
    broj += 1;
    if(broj > max_pos_speed) broj = max_pos_speed;
  }

  state_keyboard = idle_key;
  break;

case d2_key:



  if(activate_digits == 1)
  {
    broj *= 10;
    broj += 2;
    if(broj > max_pos_speed) broj = max_pos_speed;
  }

  state_keyboard = idle_key;
  break;

case d3_key:



  if(activate_digits == 1)
  {
    broj *= 10;
    broj += 3;

    if(broj > max_pos_speed) broj = max_pos_speed;
  }

  state_keyboard = idle_key;
  break;

case d4_key:



  if(activate_digits == 1)
  {
```

```
   broj *= 10;
   broj += 4;
   if(broj > max_pos_speed) broj = max_pos_speed;
  }

  state_keyboard = idle_key;
  break;

case d5_key:



  if(activate_digits == 1)
  {
   broj *= 10;
   broj += 5;
   if(broj > max_pos_speed) broj = max_pos_speed;
  }

  state_keyboard = idle_key;
  break;

case d6_key:



  if(activate_digits == 1)
  {
   broj *= 10;
   broj += 6;
   if(broj > max_pos_speed) broj = max_pos_speed;
  }

  state_keyboard = idle_key;
  break;

case d7_key:



  if(activate_digits == 1)
  {
   broj *= 10;
   broj += 7;
   if(broj > max_pos_speed) broj = max_pos_speed;
  }
```

```
    state_keyboard = idle_key;
    break;

   case d8_key:



    if(activate_digits == 1)
    {
     broj *= 10;
     broj += 8;
     if(broj > max_pos_speed) broj = max_pos_speed;
    }

    state_keyboard = idle_key;
    break;

   case d9_key:



    if(activate_digits == 1)
    {
     broj *= 10;
     broj += 9;
     if(broj > max_pos_speed) broj = max_pos_speed;
    }

    state_keyboard = idle_key;
    break;



  default: state_keyboard = idle_key; break;

   }

 case ignore_next:
   if(new_data == 1)
   {
    new_data = 0;
    karakter = 0x00;
    state_keyboard = idle_key;
   }
   break;
```

```
default: break;

}

/****************** STEPPER MOTOR *****************************/

switch(state_motor)
{
  case idle: break;

  case update_pos:

      rel_position = wanted_position - current_position;

      current_position = wanted_position;

      if(rel_position > 0)
      {
        GPIO_ResetBits(GPIOB, GPIO_Pin_5);
        direction = 0;

      }
      else
      {
        rel_position = - rel_position;
        GPIO_SetBits(GPIOB, GPIO_Pin_5);
        direction = 1;
      }

      if(rel_position != 0)
      {
      TIM_SetAutoreload(TIM4, 3125 - 1);
      state_motor = change_pos;
      TIM_Cmd( TIM4, ENABLE );
      }
      else
      {
        state_motor = idle;
      }


      break;

  case update_speed:
```

```
        same_direction = Same_direction(wanted_speed, direction);

        if(wanted_speed < 0) sign =-1;
        else sign = 1;

        if(wanted_speed == 0)
        {
          TIM_Cmd( TIM4, DISABLE );
          state_motor = idle;
        }
        else
        {
        wanted_period = 50000/wanted_speed*sign;

        //if(current_speed == 0)

        Enable_TIM4_and_TIM4_INT();

        state_motor = change_speed;
        }

        break;

     default: break;

   }


   /****************** LCD ****************************/

   if(refresh_display == 1)
   {
     switch(state_main)
     {

     case access_denied:
       switch(alpha_num)
       {
        //case 0: GPIO_ResetBits(GPIOA, GPIO_Pin_1); alpha_num +=
LCDporuka_2(0x00, 0x10); break;
        case 0: GPIO_ResetBits(GPIOA, GPIO_Pin_1); alpha_num +=
LCDporuka_2(Init_address_lcd);break; // Init DDRAM Address
        case 1: GPIO_SetBits(GPIOA, GPIO_Pin_1); alpha_num +=
LCDporuka_2(P_lcd);break; // P
        case 2:  alpha_num += LCDporuka_2(r_lcd);break; // r
```

```
        case 3: alpha_num += LCDporuka_2(i_lcd);break; // i
        case 4: alpha_num += LCDporuka_2(n_lcd);break; // n
        case 5: alpha_num += LCDporuka_2(e_lcd);break; // e
        case 6: alpha_num += LCDporuka_2(s_lcd);break; // s
        case 7: alpha_num += LCDporuka_2(i_lcd);break; // i
        case 8: alpha_num += LCDporuka_2(t_lcd);break; // t
        case 9: alpha_num += LCDporuka_2(e_lcd);break; // e
        case 10: alpha_num += LCDporuka_2(space_lcd);break; // space
        case 11: alpha_num += LCDporuka_2(R_lcd);break; // R
        case 12: alpha_num += LCDporuka_2(F_lcd);break; // F
        case 13: alpha_num += LCDporuka_2(I_lcd);break; // I
        case 14: alpha_num += LCDporuka_2(D_lcd);break; // D
        case 15: alpha_num += LCDporuka_2(C_lcd);break; // C
          //case 4: alpha_num += LCDporuka_2(0x50, 0x00);break;

        case 16: refresh_display = 0; alpha_num = 0; TIM_Cmd( TIM2, DISABLE );
         default: break;

       }

       break;

     case b_p_choice:
       switch(alpha_num)
       {
        //case 0: GPIO_ResetBits(GPIOA, GPIO_Pin_1); alpha_num +=
LCDporuka_2(0x00, 0x10); break;
        case 0: GPIO_ResetBits(GPIOA, GPIO_Pin_1); alpha_num +=
LCDporuka_2(Init_address_lcd);break; // Init DDRAM Address
        case 1: GPIO_SetBits(GPIOA, GPIO_Pin_1); alpha_num +=
LCDporuka_2(P_lcd);break; // P
        case 2:  alpha_num += LCDporuka_2(o_lcd);break; // o
        case 3: alpha_num += LCDporuka_2(z_lcd);break; // z
        case 4: alpha_num += LCDporuka_2(space_lcd);break; // space
        case 5: alpha_num += LCDporuka_2(dash_lcd);break; // -
        case 6: alpha_num += LCDporuka_2(space_lcd);break; // space
        case 7: alpha_num += LCDporuka_2(p_lcd);break; // p
        case 8: alpha_num += LCDporuka_2(space_lcd);break; // space
        case 9: alpha_num += LCDporuka_2(B_lcd);break; // B
        case 10: alpha_num += LCDporuka_2(r_lcd);break; // r
        case 11: alpha_num += LCDporuka_2(z_lcd);break; // z
        case 12: alpha_num += LCDporuka_2(space_lcd);break; // space
        case 13: alpha_num += LCDporuka_2(dash_lcd);break; // -
        case 14: alpha_num += LCDporuka_2(space_lcd);break; // space
        case 15: alpha_num += LCDporuka_2(b_lcd);break; // b
          //case 4: alpha_num += LCDporuka_2(0x50, 0x00);break;
```

```
        case 16: refresh_display = 0; alpha_num = 0; TIM_Cmd( TIM2, DISABLE );
         default: break;

       }

       break;

     case p_menu:
       switch(alpha_num)
       {
        //case 0: GPIO_ResetBits(GPIOA, GPIO_Pin_1); alpha_num +=
LCDporuka_2(0x00, 0x10); break;
        case 0: GPIO_ResetBits(GPIOA, GPIO_Pin_1); alpha_num +=
LCDporuka_2(Init_address_lcd);break; // Init DDRAM Address
        case 1: GPIO_SetBits(GPIOA, GPIO_Pin_1); alpha_num +=
LCDporuka_2(P_lcd);break; // P
        case 2: alpha_num += LCDporuka_2(o_lcd);break; // o
        case 3: alpha_num += LCDporuka_2(z_lcd);break; // z
        case 4: alpha_num += LCDporuka_2(i_lcd);break; // i
        case 5: alpha_num += LCDporuka_2(c_lcd);break; // c
        case 6: alpha_num += LCDporuka_2(i_lcd);break; // i
        case 7: alpha_num += LCDporuka_2(j_lcd);break; // j
        case 8: alpha_num += LCDporuka_2(a_lcd);break; // a
        case 9: alpha_num += LCDporuka_2(two_dots_lcd);break; // :
        case 10: if(sign == 1) alpha_num += LCDporuka_2(plus_lcd); // +
                 else          alpha_num += LCDporuka_2(minus_lcd); // -

                 break;

        case 11: if(activate_digits == 0) lcd_num = sign*wanted_position;
                 else lcd_num = broj;
                 cifra[0] = lcd_num/1000;
                 lcd_num = lcd_num - cifra[0]*1000;

                 if(cifra[0] == 0)
                 {
                   alpha_num+=2;
                 }
                 else
                 {
                   alpha_num++;
                 }

                 break;
```

```
case 12: alpha_num += LCDporuka_2(digit_base_lcd | cifra[0]); break;

case 13: cifra[1] = lcd_num/100;
        lcd_num = lcd_num - cifra[1]*100;
        if(cifra[0] == 0 && cifra[1] == 0)
        {
          alpha_num+=2;
        }
        else
        {
          alpha_num++;
        }

        break;

case 14: alpha_num += LCDporuka_2(digit_base_lcd | cifra[1]); break;

case 15: cifra[2] = lcd_num/10;
        lcd_num = lcd_num - cifra[2]*10;
        if(cifra[0] == 0 && cifra[1] == 0 && cifra[2] == 0)
        {
          alpha_num+=2;
        }
        else
        {
          alpha_num++;
        }

        break;

case 16: alpha_num += LCDporuka_2(digit_base_lcd | cifra[2]); break;

case 17: cifra[3] = lcd_num;
        alpha_num++;

        break;

case 18: alpha_num += LCDporuka_2(digit_base_lcd | cifra[3]);

        break;

case 19: case 20: case 21: case 22:
alpha_num += LCDporuka_2(space_lcd);break; // space

case 23: refresh_display = 0;  alpha_num = 0; TIM_Cmd( TIM2, DISABLE );
 default: break;
```

```
      }

      break;

     case b_menu:
      switch(alpha_num)
      {
       //case 0: GPIO_ResetBits(GPIOA, GPIO_Pin_1); alpha_num +=
LCDporuka_2(0x00, 0x10); break;
       case 0: GPIO_ResetBits(GPIOA, GPIO_Pin_1); alpha_num +=
LCDporuka_2(Init_address_lcd);break; // Clear Display
       case 1: GPIO_SetBits(GPIOA, GPIO_Pin_1); alpha_num +=
LCDporuka_2(B_lcd);break; // B
       case 2:  alpha_num += LCDporuka_2(r_lcd);break; // r
       case 3: alpha_num += LCDporuka_2(z_lcd);break; // z
       case 4: alpha_num += LCDporuka_2(i_lcd);break; // i
       case 5: alpha_num += LCDporuka_2(n_lcd);break; // n
       case 6: alpha_num += LCDporuka_2(a_lcd);break; // a
       case 7: alpha_num += LCDporuka_2(two_dots_lcd);break; // :
       case 8: if(sign == 1) alpha_num += LCDporuka_2(plus_lcd); // +
               else          alpha_num += LCDporuka_2(minus_lcd); // -

               break;

       case 9: if(activate_digits == 0) lcd_num = sign*wanted_speed;
               else lcd_num = broj;
               cifra[0] = lcd_num/1000;
               lcd_num = lcd_num - cifra[0]*1000;

               if(cifra[0] == 0)
               {
                 alpha_num+=2;
               }
               else
               {
                 alpha_num++;
               }

               break;

       case 10: alpha_num += LCDporuka_2(digit_base_lcd | cifra[0]); break;

       case 11: cifra[1] = lcd_num/100;
               lcd_num = lcd_num - cifra[1]*100;
               if(cifra[0] == 0 && cifra[1] == 0)
```

```
        {
          alpha_num+=2;
        }
        else
        {
          alpha_num++;
        }

        break;

 case 12: alpha_num += LCDporuka_2(digit_base_lcd | cifra[1]); break;

 case 13: cifra[2] = lcd_num/10;
        lcd_num = lcd_num - cifra[2]*10;
        if(cifra[0] == 0 && cifra[1] == 0 && cifra[2] == 0)
        {
          alpha_num+=2;
        }
        else
        {
          alpha_num++;
        }

        break;

 case 14: alpha_num += LCDporuka_2(digit_base_lcd | cifra[2]); break;

 case 15: cifra[3] = lcd_num;
        alpha_num++;

        break;

 case 16: alpha_num += LCDporuka_2(digit_base_lcd | cifra[3]);

        break;

case 17: case 18: case 19: case 20: case 21: case 22:
 alpha_num += LCDporuka_2(space_lcd);break; // space



 case 23: refresh_display = 0; alpha_num = 0; TIM_Cmd( TIM2, DISABLE );
  default: break;

}
```

```
      break;

    }

  }

 }
}

void assert_failed(uint8_t* file, uint32_t line)
{
 /* User can add his own implementation to report the file name and line number,
    ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */

 /* Infinite loop */
 while (1)
 {
 }
}
```

stm32f10x_it.c

```
/**
******************************************************************************
  * @file    Demo/src/stm32f10x_it.c
  * @author  MCD Application Team
  * @version V1.0.0
  * @date    09/13/2010
  * @brief   Main Interrupt Service Routines.
  *          This file provides template for all exceptions handler and peripherals
  *          interrupt service routine.

******************************************************************************
  * @copy
  *
  * THE PRESENT FIRMWARE WHICH IS FOR GUIDANCE ONLY AIMS AT
PROVIDING CUSTOMERS
  * WITH CODING INFORMATION REGARDING THEIR PRODUCTS IN ORDER
FOR THEM TO SAVE
  * TIME. AS A RESULT, STMICROELECTRONICS SHALL NOT BE HELD
LIABLE FOR ANY
  * DIRECT, INDIRECT OR CONSEQUENTIAL DAMAGES WITH RESPECT TO
ANY CLAIMS ARISING
```

```
  * FROM THE CONTENT OF SUCH FIRMWARE AND/OR THE USE MADE BY
CUSTOMERS OF THE
  * CODING INFORMATION CONTAINED HEREIN IN CONNECTION WITH
THEIR PRODUCTS.
  *
  * <h2><center>&copy; COPYRIGHT 2010 STMicroelectronics</center></h2>
  */

/* Includes ------------------------------------------------------------------*/
#include "stm32f10x_it.h"

#include "ConstsAndPrototypes_Int.h"

/** @addtogroup Demo
  * @{
  */

/* Private typedef -----------------------------------------------------------*/
/* Private define ------------------------------------------------------------*/
/* Private macro -------------------------------------------------------------*/
/* Private variables ---------------------------------------------------------*/

/* Private function prototypes -----------------------------------------------*/
/* Private functions ---------------------------------------------------------*/

/******************************************************************************
*******/
/*            Cortex-M3 Processor Exceptions Handlers                         */
/******************************************************************************
*******/

/**
  * @brief  This function handles NMI exception.
  * @param  None
  * @retval None
  */
void NMI_Handler(void)
{
}

/**
  * @brief  This function handles Hard Fault exception.
  * @param  None
  * @retval None
  */
void HardFault_Handler(void)
```

```
{
  /* Go to infinite loop when Hard Fault exception occurs */
  while (1)
  {
  }
}

/**
  * @brief  This function handles Memory Manage exception.
  * @param  None
  * @retval None
  */
void MemManage_Handler(void)
{
  /* Go to infinite loop when Memory Manage exception occurs */
  while (1)
  {
  }
}

/**
  * @brief  This function handles Bus Fault exception.
  * @param  None
  * @retval None
  */
void BusFault_Handler(void)
{
  /* Go to infinite loop when Bus Fault exception occurs */
  while (1)
  {
  }
}

/**
  * @brief  This function handles Usage Fault exception.
  * @param  None
  * @retval None
  */
void UsageFault_Handler(void)
{
  /* Go to infinite loop when Usage Fault exception occurs */
  while (1)
  {
  }
}
```

```
/**
  * @brief  This function handles SVCall exception.
  * @param  None
  * @retval None
  */
void SVC_Handler(void)
{
}

/**
  * @brief  This function handles Debug Monitor exception.
  * @param  None
  * @retval None
  */
void DebugMon_Handler(void)
{
}

/**
  * @brief  This function handles PendSV_Handler exception.
  * @param  None
  * @retval None
  */
void PendSV_Handler(void)
{
}

/**
  * @brief  This function handles SysTick Handler.
  * @param  None
  * @retval None
  */
void SysTick_Handler(void)
{

}

/****************** KEYBOARD *****************************/

char karakter = 0x00, bit_karakter, cnt = 0, new_data = 0;

void EXTI9_5_IRQHandler(void)
{

  if ( EXTI_GetITStatus( EXTI_Line9 ) == SET )
  {
```

```
    cnt++;
    switch(cnt)
    {
      case 1: break;
      case 2: case 3: case 4: case 5: case 6: case 7: case 8: case 9:

       bit_karakter = GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_8);
       karakter = karakter | (bit_karakter << (cnt - 2));
       break;

      case 10: break;
      case 11: cnt = 0; new_data = 1; break;
      default: break;
    }

     EXTI_ClearFlag(EXTI_Line9);
  }

}


/****************** STEPPER MOTOR *******************************/

extern int rel_position, current_period, wanted_period;

extern char same_direction, direction;

extern state_motor_type state_motor;

void TIM4_IRQHandler(void)
{
  switch(state_motor)
  {
    case change_pos:


        rel_position--;

        if(rel_position == 0)
        {
          state_motor = idle;
          TIM_Cmd( TIM4, DISABLE );
        }
        break;

    case change_speed:
```

```
if(same_direction == 1)
{
 if(wanted_period > limit_slow_fast && current_period > limit_slow_fast)
 {
  current_period = wanted_period;

 }
 else if (wanted_period < limit_slow_fast && current_period > limit_slow_fast)
 {
  current_period = current_period/2;
 }
 else if (wanted_period > limit_slow_fast && current_period < limit_slow_fast)
 {
  current_period += increment_count;
 }
 else if (wanted_period > current_period)
 {
  current_period ++;
 }
 else if (wanted_period < current_period)
 {
  current_period --;
 }

 TIM_SetAutoreload(TIM4, current_period - 1);

 if(current_period == wanted_period)
 {
  TIM_ITConfig(TIM4,TIM_IT_Update, DISABLE);
  state_motor = idle;
 }
}
else
{
  if(current_period > limit_direction)
  {
   direction = direction ^ 0x01;
   same_direction = 1;

  if(direction == 0)
   {
    GPIO_ResetBits(GPIOB, GPIO_Pin_5);

   }
   else
```

```
            {
             GPIO_SetBits(GPIOB, GPIO_Pin_5);
            }

          }
          else
          {
            current_period = current_period*period_slope;
          }
        }

       break;

    default: state_motor = idle; break;
  }

  //Brisemo prekidni fleg
  TIM_ClearFlag(TIM4, TIM_FLAG_Update);
}


/****************** RFID *****************************/


char falling_data = 0, cnt_clk = 0, sync = 0, data_index = 0, b[2], xored, Mem_array[64],
data_read_done = 0, out;

//char samples[500];

int indice = 0;

uint32_t RFID[2] = {0,0};

void EXTI15_10_IRQHandler(void)
{
  // OUT

  if ( EXTI_GetITStatus( EXTI_Line11 ) == SET )
  {
    out = GPIO_ReadInputDataBit(GPIOC, GPIO_Pin_11);

    if(out == 0 && sync == 0)
    {

    falling_data = 1;
    //Mask_Interrupt(Data_Line);
```

```
   //EXTI->IMR &= 0x3F7FF;
   }

   if(sync == 1 && (cnt_clk < RFID_period_offset || cnt_clk > RFID_period-
RFID_period_offset))
   {
     cnt_clk = 0;
   }

   EXTI_ClearFlag(EXTI_Line11);
  }


  // RDY/CLK

  if ( EXTI_GetITStatus( EXTI_Line12 ) == SET )
  {
   if(falling_data == 1)
   {
      cnt_clk = 0;
      falling_data = 0;
      sync = 1;
      data_index = 0;
      RFID[0] = 0;
      RFID[1] = 0;
      indice = 0;
   }

   if(sync == 1)
   {

     cnt_clk++;

     //samples[indice] = GPIO_ReadInputDataBit(GPIOC, GPIO_Pin_11);

     //indice++;

     //if(indice > 500) indice = 0;

     switch(cnt_clk)
     {
       case RFID_period/4-RFID_offset_first: b[0] = GPIO_ReadInputDataBit(GPIOC,
GPIO_Pin_11); break;
       case RFID_period*3/4-RFID_offset_second:
           b[1] = GPIO_ReadInputDataBit(GPIOC, GPIO_Pin_11);
```

```
        xored = b[1] ^ b[0];
        Mem_array[data_index] = b[1];
        if(data_index < RFID_elementary)  RFID[0] |= b[1] << data_index;
        else                RFID[1] |= b[1] << (data_index-RFID_elementary);


        if(xored == 0 || (data_index < (N_ones - 1) && Mem_array[data_index] != 1))
        {
            sync = 0;
          //EXTI->IMR |= 0x800;
        }
        if(data_index == (N_RFID_bits-1))
        {
          data_read_done = 1;
          //RFID = 0;
          sync = 0;
          //EXTI->IMR |= 0x0000;
        }
        data_index++;

        break;

      case RFID_period: cnt_clk = 0;     break;
      default: break;
     }

   }



   EXTI_ClearFlag(EXTI_Line12);
  }



  //EXTI_ClearFlag(EXTI_Line11);

}

/*******************************************************************************
*******/
/*               STM32F10x Peripherals Interrupt Handlers                  */
/*  Add here the Interrupt Handler for the used peripheral(s) (PPP), for the  */
/*  available peripheral interrupt handler's name please refer to the startup */
/*  file (startup_stm32f10x_xx.s).                                            */
```

```
/*******************************************************************************
*******/

/**
  * @brief  This function handles PPP interrupt request.
  * @param  None
  * @retval None
  */
/*void PPP_IRQHandler(void)
{
}*/


/**
  * @}
  */

/******************** (C) COPYRIGHT 2010 STMicroelectronics *****END OF
FILE****/
```

ConstAndPrototypes.h

```
/* Custom types */

typedef enum {
   idle, update_pos, change_pos, update_speed, change_speed
} state_motor_type;


/* Keyboard */

#define Break_Code    0xF0
#define C_key         0x21
#define P_key         0x4D
#define B_key         0x32
#define D_key         0x23
#define A_key         0x1C
#define Plus_key      0x79
#define Minus_key     0x7B
#define Enter_key     0x5A
#define Esc_key       0x76
#define Bkspc_key     0x66

#define an1_key       0x16
#define an2_key       0x1E
```

```
#define an3_key       0x26
#define an4_key       0x25

#define d0_key        0x70
#define d1_key        0x69
#define d2_key        0x72
#define d3_key        0x7A
#define d4_key        0x6B
#define d5_key        0x73
#define d6_key        0x74
#define d7_key        0x6C
#define d8_key        0x75
#define d9_key        0x7D

/* Step motor */

#define max_pos_speed          500
#define default_step_period    5000


/* LCD */

#define Init_address_lcd       0x80
#define P_lcd                  0x50
#define r_lcd                  0x72
#define i_lcd                  0x69
#define n_lcd                  0x6E
#define e_lcd                  0x65
#define s_lcd                  0x73
#define t_lcd                  0x74
#define space_lcd              0x20
#define R_lcd                  0x52
#define F_lcd                  0x46
#define I_lcd                  0x49
#define D_lcd                  0x44
#define C_lcd                  0x43
#define o_lcd                  0x6F
#define p_lcd                  0x70
#define z_lcd                  0x7A
#define dash_lcd               0xB0
#define B_lcd                  0x42
#define b_lcd                  0x62
#define c_lcd                  0x63
#define j_lcd                  0x6A
#define a_lcd                  0x61
#define two_dots_lcd           0x3A
```

```
#define plus_lcd              0x2B
#define minus_lcd              0x2D
#define digit_base_lcd         0x30



/* Prototipovi funkcija */

void Init_keyboard();

void Init_stepper_motor();

void Init_LCD();

void Init_RFID();

char Same_direction(int speed, char dir);

void Enable_TIM4_and_TIM4_INT();

void Delay();

char Refresh_LCD();

void LCDporuka(unsigned char MSBprim,unsigned char LSBprim);

char LCDporuka_2(unsigned char LCD_char);
```

ConstAndPrototypes.c

```
#include "stm32f10x.h"
#include "STM32vldiscovery.h"

//#include "ConstsAndPrototypes.h"

GPIO_InitTypeDef GPIO_InitStruct;
EXTI_InitTypeDef EXTI_InitStruct;
NVIC_InitTypeDef NVIC_InitStruct;


GPIO_InitTypeDef GPIO_InitStructure;
TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStruct;
TIM_OCInitTypeDef TIM_OCInitStruct;
NVIC_InitTypeDef NVIC_InitStructure;
```

```
EXTI_InitTypeDef EXTI_InitStructure;

/* Staticne varijable */

static char korak = 0;


/* Prototipovi funkcija */

void Delay()
{
   while(TIM_GetFlagStatus(TIM2, TIM_FLAG_Update) == RESET)
    {

    }
    TIM_ClearFlag(TIM2, TIM_FLAG_Update);

}

void Enable_TIM4_and_TIM4_INT()
{

  TIM_Cmd( TIM4, ENABLE );
  TIM_ITConfig(TIM4,TIM_IT_Update, ENABLE);

}

char Same_direction(int speed, char dir)
{
     if((speed > 0 && dir == 0) || (speed < 0 && dir == 1))
     {
       return 1;
     }
     else
     {
       return 0;
     }
}

char Refresh_LCD()
{
  TIM_Cmd( TIM2, ENABLE );
  return 1;
}

void LCDporuka(unsigned char MSBprim,unsigned char LSBprim)
```

```
{
 GPIO_SetBits(GPIOA, GPIO_Pin_0); //postavljanje E bita
 Delay();

 GPIO_SetBits(GPIOA, MSBprim);      //upis gornjeg nibla
 Delay();
 GPIO_ResetBits(GPIOA, GPIO_Pin_0);  //upis komande na silaznu ivicu E bita
 Delay();
 GPIO_ResetBits(GPIOA, MSBprim);
 Delay();

 GPIO_SetBits(GPIOA, GPIO_Pin_0);        //postavljanje E bita
 Delay();
 GPIO_SetBits(GPIOA, LSBprim);    //upis donjeg nibla
 Delay();
 GPIO_ResetBits(GPIOA, GPIO_Pin_0);  //upis komande na silaznu ivicu E bita
 Delay();
 GPIO_ResetBits(GPIOA, LSBprim);
 Delay();

}

char LCDporuka_2(unsigned char LCD_char)
{
 char Timer_IF, MSBprim = 0x00, LSBprim = 0x00;

 MSBprim = LCD_char & 0xF0;
 LSBprim = (LCD_char & 0x0F) << 4;

 Timer_IF = TIM_GetFlagStatus(TIM2, TIM_FLAG_Update);

 if(Timer_IF == SET)
 {
  TIM_ClearFlag(TIM2, TIM_FLAG_Update);
  TIM_SetCounter(TIM2, 0);

  switch(korak)
 {
  case 0:
  GPIO_SetBits(GPIOA, GPIO_Pin_0); //postavljanje E bita
  korak++;
  break;

 case 1:

  GPIO_SetBits(GPIOA, MSBprim);      //upis gornjeg nibla
```

```
  korak++;
  break;

 case 2:

  GPIO_ResetBits(GPIOA, GPIO_Pin_0);  //upis komande na silaznu ivicu E bita
  korak++;
  break;

 case 3:

  GPIO_ResetBits(GPIOA, MSBprim);
  korak++;
  break;

case 4:
  GPIO_SetBits(GPIOA, GPIO_Pin_0);        //postavljanje E bita
  korak++;
  break;


  case 5:

  GPIO_SetBits(GPIOA, LSBprim);    //upis donjeg nibla
  korak++;
  break;

  case 6:

  GPIO_ResetBits(GPIOA, GPIO_Pin_0);  //upis komande na silaznu ivicu E bita
  korak++;
  break;

  case 7:

  GPIO_ResetBits(GPIOA, LSBprim);
  korak = 0;
  return 1;

  //break;

  default: break;
}

}
```

```
 return 0;

}

void Init_keyboard()
{
 // Inicijalizacija tastature

 RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);

 GPIO_InitStruct.GPIO_Pin   = GPIO_Pin_8 | GPIO_Pin_9;
 GPIO_InitStruct.GPIO_Mode  = GPIO_Mode_IPU;
 GPIO_Init(GPIOB, &GPIO_InitStruct);
 GPIO_StructInit(&GPIO_InitStruct);

 // KAD HOCES INTERRUPTS UVEK AKTIVIRAJ CLOCK ZA AFIO !!!

 RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);

 GPIO_EXTILineConfig(GPIO_PortSourceGPIOB, GPIO_PinSource9);

 EXTI_InitStruct.EXTI_Line = EXTI_Line9;
 EXTI_InitStruct.EXTI_Mode = EXTI_Mode_Interrupt;
 EXTI_InitStruct.EXTI_Trigger = EXTI_Trigger_Falling;
 EXTI_InitStruct.EXTI_LineCmd = ENABLE;
 EXTI_Init(&EXTI_InitStruct);
 EXTI_StructInit(&EXTI_InitStruct);

 NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0);

 NVIC_InitStruct.NVIC_IRQChannel = EXTI9_5_IRQn;
 NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0x0F;
 NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0x0F;
 NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;

 NVIC_Init(&NVIC_InitStruct);
}

void Init_stepper_motor()
{
 // Inicijalizacija step motora

  RCC_APB1PeriphClockCmd( RCC_APB1Periph_TIM4, ENABLE );

 //inicijalizacija portova
   GPIO_StructInit(&GPIO_InitStructure); // Reset init structure
```

```
   GPIO_InitStructure.GPIO_Pin =  GPIO_Pin_6;//plava LED dioda
   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;           // konfiguracija za
alternativnu funkciju - Push Pull
   //GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;          // ovo bi bila
konfiguracija da se koristi kao obican pin
   GPIO_Init( GPIOB, &GPIO_InitStructure );
   //GPIO_PinRemapConfig( GPIO_PartialRemap_TIM3, ENABLE );       // Map
TIM3_CH3 to GPIOC.Pin8, TIM3_CH4 to GPIOC.Pin9

   GPIO_StructInit(&GPIO_InitStructure); // Reset init structure
   GPIO_InitStructure.GPIO_Pin =  GPIO_Pin_5;//plava LED dioda
   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
   GPIO_Init( GPIOB, &GPIO_InitStructure );

   GPIO_ResetBits(GPIOB, GPIO_Pin_5);

   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);

   GPIO_StructInit(&GPIO_InitStructure); // Reset init structure
   GPIO_InitStructure.GPIO_Pin =  GPIO_Pin_0 | GPIO_Pin_1;//plava LED dioda
   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
   GPIO_Init( GPIOC, &GPIO_InitStructure );

   GPIO_ResetBits(GPIOC, GPIO_Pin_0 | GPIO_Pin_1);


   TIM_TimeBaseStructInit( &TIM_TimeBaseInitStruct );
   //TIM_TimeBaseInitStruct.TIM_ClockDivision = TIM_CKD_DIV4;
   TIM_TimeBaseInitStruct.TIM_Period = 50000 - 1;   // 0..999
   TIM_TimeBaseInitStruct.TIM_Prescaler = 480 - 1; // Div 240
   TIM_TimeBaseInitStruct.TIM_CounterMode = TIM_CounterMode_Up;
   TIM_TimeBaseInit( TIM4, &TIM_TimeBaseInitStruct );
   TIM_ARRPreloadConfig(TIM4, ENABLE);

   //Inicijalizacija OC strukture tajmera TIM3
   TIM_OCStructInit( &TIM_OCInitStruct );
   TIM_OCInitStruct.TIM_OutputState = TIM_OutputState_Enable;
   TIM_OCInitStruct.TIM_OCMode = TIM_OCMode_PWM1;
   //Initial duty cycle equals 0%. Value can range from zero to 1000.
   TIM_OCInitStruct.TIM_Pulse = 1; // 0 .. 1000 (0=Always Off, 1000=Always On)
   TIM_OC1Init( TIM4, &TIM_OCInitStruct ); // Channel 1 -- OVDE DEFINISE DA JE
U PITANJU KANAL 1 IZLAZ - TIM_OC1Init !!!
```

```
   //Dozvola generisanja prekida na nivou periferije
   TIM_ITConfig(TIM4,TIM_IT_Update, ENABLE);

   TIM_SetAutoreload(TIM4, 50000 - 1);

   TIM_Cmd( TIM4, DISABLE ); // stopiranje tajmera

   //DOZVOLA GENERISANJA PREKIDA
   //Dozvola generisanja prekida od strane TIM4 periferije
   NVIC_InitStructure.NVIC_IRQChannel = TIM4_IRQn;
   NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x0F;
   NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x0E;
   NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
   //Inicijalizacija NVIC periferije
   NVIC_Init(&NVIC_InitStructure);

}

void Init_LCD()
{
  // Inicijalizacija LCD-a

   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);

   GPIO_StructInit(&GPIO_InitStruct);
   GPIO_InitStruct.GPIO_Pin   = GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_4 |
GPIO_Pin_5 | GPIO_Pin_6 | GPIO_Pin_7;
   GPIO_InitStruct.GPIO_Mode  = GPIO_Mode_Out_PP;
   GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;
   GPIO_Init(GPIOA, &GPIO_InitStruct);
   GPIO_StructInit(&GPIO_InitStruct);

   //GPIO_SetBits(GPIOA, GPIO_Pin_8);

   RCC_APB1PeriphClockCmd( RCC_APB1Periph_TIM2, ENABLE );

    TIM_TimeBaseStructInit( &TIM_TimeBaseInitStruct );
   //TIM_TimeBaseInitStruct.TIM_ClockDivision = TIM_CKD_DIV4;
   TIM_TimeBaseInitStruct.TIM_Period = 3000 - 1;   // to 3000*1us = 3ms
   TIM_TimeBaseInitStruct.TIM_Prescaler = 24 - 1; // 1 us
   TIM_TimeBaseInitStruct.TIM_CounterMode = TIM_CounterMode_Up;
   TIM_TimeBaseInit( TIM2, &TIM_TimeBaseInitStruct );
   TIM_ARRPreloadConfig(TIM2, ENABLE);


    // LCD Init
```

```
   TIM_Cmd( TIM2, ENABLE );

   GPIO_ResetBits(GPIOA, GPIO_Pin_1);
   Delay();
   GPIO_SetBits(GPIOA, GPIO_Pin_0);
   Delay();
   GPIO_SetBits(GPIOA, 0x20);
   Delay();
   GPIO_ResetBits(GPIOA, GPIO_Pin_0);
   Delay();
   GPIO_ResetBits(GPIOA, 0x20);
   Delay();
   //LCDporuka(0x20,0x00);
   //Delay();
   LCDporuka(0x20,0x00);
   Delay();
   LCDporuka(0x00,0xE0);
   Delay();
   LCDporuka(0x00,0x10);
   Delay();
   LCDporuka(0x00,0x60);
   Delay();

   TIM_SetAutoreload(TIM2, 50 - 1);

   //TIM_Cmd( TIM2, DISABLE );

  // LCDporuka(0x20,0x80);
   //Delay();

   //LCDporuka(0x80,0x00);
  // Delay();


  // Delay();

}

void Init_RFID()
{
     // Inicijalizacija RFID-a

   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);

  GPIO_StructInit(&GPIO_InitStruct);
```

```
GPIO_InitStruct.GPIO_Pin   = GPIO_Pin_11 | GPIO_Pin_12;
GPIO_InitStruct.GPIO_Mode  = GPIO_Mode_IN_FLOATING;
GPIO_Init(GPIOC, &GPIO_InitStruct);
GPIO_StructInit(&GPIO_InitStruct);

GPIO_StructInit(&GPIO_InitStruct);
GPIO_InitStruct.GPIO_Pin   = GPIO_Pin_8 | GPIO_Pin_9;
GPIO_InitStruct.GPIO_Mode  = GPIO_Mode_Out_PP;
GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOC, &GPIO_InitStruct);
GPIO_StructInit(&GPIO_InitStruct);

GPIO_ResetBits(GPIOC, GPIO_Pin_8 | GPIO_Pin_9);

RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);

// RDY/CLK

GPIO_EXTILineConfig(GPIO_PortSourceGPIOC, GPIO_PinSource12);

EXTI_InitStruct.EXTI_Line = EXTI_Line12;
EXTI_InitStruct.EXTI_Mode = EXTI_Mode_Interrupt;
EXTI_InitStruct.EXTI_Trigger = EXTI_Trigger_Falling;
EXTI_InitStruct.EXTI_LineCmd = ENABLE;
EXTI_Init(&EXTI_InitStruct);
EXTI_StructInit(&EXTI_InitStruct);



// OUT

GPIO_EXTILineConfig(GPIO_PortSourceGPIOC, GPIO_PinSource11);

EXTI_InitStruct.EXTI_Line = EXTI_Line11;
EXTI_InitStruct.EXTI_Mode = EXTI_Mode_Interrupt;
EXTI_InitStruct.EXTI_Trigger = EXTI_Trigger_Rising_Falling;
EXTI_InitStruct.EXTI_LineCmd = ENABLE;
EXTI_Init(&EXTI_InitStruct);
EXTI_StructInit(&EXTI_InitStruct);



NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0);

NVIC_InitStruct.NVIC_IRQChannel = EXTI15_10_IRQn;
NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0x0F;
```

```
  NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0x0F;
  NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;

  NVIC_Init(&NVIC_InitStruct);

}
```

ConstAndPrototypes_Int.h

```
/* Init Structures */



/* Custom types */

typedef enum {
   idle, update_pos, change_pos, update_speed, change_speed
} state_motor_type;


/* Keyboard */


/* Step motor */

#define limit_slow_fast     400
#define increment_count      10
#define limit_direction   12500
#define period_slope          2


/* RFID */

#define RFID_period          64
#define RFID_period_offset    9
#define RFID_offset_first     4
#define RFID_offset_second    3
#define N_RFID_bits          64
#define N_ones                9
#define RFID_elementary      32
```