A blue and silver logo

Description automatically generated

Korišćenje STM32F103C8 mikrokontrolera za dobijanje udaljenosti između 2 koordinate.

4 AVgust 2023

Prirodno-matematički fakultet

Radoja Domanovića 12, Kragujevac

|  |  |
| --- | --- |
| Autor:  Vildan Čuturić, 118/2018 | Mentor:  Dr. Aleksandar Peulić |

Sadržaj:

[Kratak opis projekta 2](#_Toc142032230)

[Potrebne komponente 2](#_Toc142032231)

[STM32F103C8 2](#_Toc142032232)

[Kreiranje i podešavanje projekta u STM32CUBEIDE 2](#_Toc142032233)

[Tok projekta 2](#_Toc142032234)

[GPS Modul 2](#_Toc142032235)

[Osnovni pojmovi GPS Modula 3](#_Toc142032236)

[Integracija i korišćenje GPS modula 3](#_Toc142032237)

[Proteus simulacija 3](#_Toc142032238)

[STM32F103C8 Firmware 4](#_Toc142032239)

[Primanje podataka na RX pin 4](#_Toc142032240)

[Slanje podataka sa TX pina 6](#_Toc142032241)

[Haversine funkcija udaljenosti 12](#_Toc142032242)

[Zaključak 12](#_Toc142032243)

[Literatura: 13](#_Toc142032244)

# Kratak opis projekta

Simuliraćemo stvarne izlaze GPS Modula u vidu teksta i koristićemo STM32F103C8 mikrokontroler da pronađemo koordinate naše trenutne lokacije i da za unešene koordinate druge lokacije pronađemo njihovu razdaljinu i prikažemo je na virtuelnom terminalu.

# Potrebne komponente

* STM32F103C8 mikrokontroler
* Virtuelni Terminal

# STM32F103C8

Naprednija verzija STM32F103C6 kontrolera, imamo više FLASH memorije, kod STM32F103C6 imamo od nekih 16 do 32KB dok kod STM32F103C8 mikrokontrolera imamo od 64 do 128KB FLASH memorije. Kako ćemo vršiti malo kompleksnija matematička izračunavanja ovaj podatak nam je jako bitan, čak i presudan u odabiru mikrokontrolera. Pored toga imamo i više RAM-a, na STM32F103C6 RAM iznosi od 6 do 10KB dok kod STM32F103C8 imamo 20KB RAM-a. Međutim, RAM nam i nije toliko presudan, jer Proteus, program koji ćemo koristiti za simulaciju ne podržava više od 10KB RAM memorije.

## Kreiranje i podešavanje projekta u STM32CUBEIDE

1. Otvoriti STM32CUBEIDE > New > STM32 Project > Commercial Part Number: STM32F103C8T6
2. Categories > System Core > RCC > High Speed Clock > Crystal/Ceramic Resonator
3. A->Z > USART1 > Mode > Asynchronous
4. Proveriti da li je u Parameter Settings Baud Rate 115200 Bit/s, ako nije, podesiti.
5. A->Z > USART1 > NVIC Settings > Čekirati USART1 Global Interrupt

## Tok projekta

U članku iznad pomenuli smo USART (engl. Universal Synchronous/Asynchronous Receiver/Transmitter). USART je hardver koji omogućava uređaju da komunicira korišćenjem **Serial protokola**.

Serial protokole koristimo kako bi pomerali bitove sekvencijalno, ili po jedan, između komponenti ili različitih uređaja. Serial komunikacije se koriste u skoro svakom digitalnom uređaju.

Mi ćemo USART koristiti kako bi na virtuelnom terminalu prikazali kakvi nam to podaci stižu sa GPS modula, a takođe ćemo ga iskoristiti kako bi uneli koordinate do kojih želimo da preračunamo distancu i naravno, udaljenost između koordinata.

# GPS Modul

Što se tiče GPS Modula, nismo u mogućnosti da nabavimo pravi GPS modul, međutim mi u Proteus-u imamo VGPS tj. simulaciju GPS uređaja i njega ćemo da koristimo kako bi izvukli primer ispisa jednog stvarnog GPS Modula..

A screenshot of a computer

Description automatically generated

*Slika 1.*

## Osnovni pojmovi GPS Modula

Kako bi bolje razumeli kakav problem imamo, i šta pokušavamo da uradimo, dobar je početak sam GPS Modul. Kao što vidimo sa *Slike 1.* Imamo TX i RX tu su nam zapravo Transmit i Receive pinovi, respektivno. Transmit pin zapravo izvozi podatke iz GPS modula, videćemo u nastavku i kakve, i šta oni znače. Sledeće šta vidimo jesu neke koordinate, te koordinate su Latitude (koordinata iznad) i Longitude (koordinata ispod). I na kraju, vidimo neka 2 slova, prvo slovo, u trenutnom primeru, N (engl. North), nam označava severnu hemisferu, ova koordinata može biti i S (engl. South) što bi označavalo južnu hemisferu. Isto pravilo važi za slovo pored koordinate ispod, ono nam označava W (engl. West) zapadnu hemisferu i E (engl. East), što bi označavalo istočnu hemisferu.

## Integracija i korišćenje GPS modula

Sada kada znamo šta je šta na GPS modulu, sledeći problem nam je kako da predstavimo nekom uređaju ove podatke koji dalje može da radi sa njima i izvlači neke informacije koje su korisne za čoveka.

U stvarnoj situaciji, da zapravo imamo ovakav uređaj, povezali bi smo TX pin GPS modula na RX pin uređaja koji želimo da primi ove podatke, u našem slučaju bi to bio STM32F103C8 mikrokontroler.

Kako radimo sa simulacijom, naš uređaj nije bio u stanju da pravilno pročita podatke koje je slao GPS modul, pa ćemo da uzmemo samo validne podatke iz ovog uređaja i da ih predstavimo tekstualno u firmware-u.

# Proteus simulacija

1. Otvoriti Proteus > Napraviti novi projekat.
2. Dodati STM32F103C6 ploču, jer trenutno nemamo podršku za našu STM32F103C8.
3. Dodati Virtuelni terminal, može se pronaći na kartici: ,,instrumenti’’.
4. Dodati struju i uzemljenje.
5. Povezati RXD pin Virtuelnog terminala na TX pin STM32F103C8 mikrokontrolera, u našem slučaju to bi bio PA9 pin.
6. Povezati TXD pin Virtuelnog terminala na RX pin STM32F103C8 mikrokontrolera, u našem slučaju to bi bio PA10 pin.
7. Desni klik na Virtuelni terminal > Edit Properties > Baud Rate postaviti na 115200.

Kada smo sve završili Proteus simulacija bi trebala izgledati ovako:

A screen shot of a computer

Description automatically generated

*Slika 2.*

# STM32F103C8 Firmware

Sada kada smo se upoznali sa uređajima i problemom koji treba da rešimo, sada to treba i uraditi.

## Primanje podataka na RX pin

Vraćamo se na korak gde smo generisali kod za STM32F103C8, tačnije, da malo objasnimo korak:

1. A->Z > USART1 > NVIC Settings > Čekirati USART1 Global Interrupt

Kada dobijamo neke podatke sa GPS modula, mi želimo da imamo neku kontrolu nad nadolazećim podacima, to možemo uraditi pomoću prekida (engl. Interrupt), naime, mi ćemo svaki put kada dobijemo bilo kakav karakter pozvati prekid, da taj isti karakter možemo da obradimo.

Čekirali smo da želimo prekid na USART1, međutim kod nije generisan pa to moramo uraditi sami. Al pre toga moramo još definisati na koji način će se desiti prekid.

Prekid bi trebalo da se desi na unos jednog karaktera, ali pre toga moramo definisati gde ćemo primati te karaktere i koliko.

Prvo definisati strukturu koja će primati te podatke.

Pronaći u kodu liniju:

|  |
| --- |
| /\* USER CODE BEGIN PV \*/ |

I ispod nje dodati sledeću liniju koda:

|  |
| --- |
| uint8\_t UART1\_rxBuffer[32] = { 0 }; |

Sa ovim pravimo bafer u koji ćemo smeštati primljene podatke sa RX pina STM32F103C8 ploče.

Sledeće je potrebno definisati kako će se desiti prekid, to možemo da uradimo na sledeći način:

Pronaći u kodu liniju:

|  |
| --- |
| /\* USER CODE BEGIN 2 \*/ |

I ispod nje dodati sledeću liniju:

|  |
| --- |
| HAL\_UART\_Receive\_IT(&huart1, UART1\_rxBuffer, 1); |

Sa ovim smo uradili sledeće: Primi poruke sa UART1 ulaza, smesti ih u UART1\_rxBuffer koji smo definisali iznad, i dužina poruke je 1 dakle, karakter po karakter, i sa \_IT smo rekli da kada dobiješ ovakvo nešto, treba da se dogodi prekid.

I na kraju, treba definisati funkciju u kojoj će se obrađivati prekid, to možemo uraditi na sledeći način:

Pronaći u kodu liniju:

|  |
| --- |
| /\* USER CODE BEGIN 4 \*/ |

I ispod nje dodati funkciju:

|  |
| --- |
| void HAL\_UART\_RxCpltCallback(UART\_HandleTypeDef \*huart) {  } |

Na kraju bi to trebali izgledati ovako:

|  |
| --- |
| /\* USER CODE BEGIN 4 \*/ void HAL\_UART\_RxCpltCallback(UART\_HandleTypeDef \*huart) {  } /\* USER CODE END 4 \*/ |

Ova funkcija će se pozivati prema gore definisanim kriterijumima, dakle ako stigne 1 karakter.   
**Važno!** Zapamtiti da treba da nastavimo da primamo karaktere kada ih obradimo.

Dodati samo:

|  |
| --- |
| /\* USER CODE BEGIN 4 \*/ void HAL\_UART\_RxCpltCallback(UART\_HandleTypeDef \*huart) {  HAL\_UART\_Receive\_IT(&huart1, UART1\_rxBuffer, 1); } /\* USER CODE END 4 \*/ |

## Slanje podataka sa TX pina

Definisali smo prekid, definisali smo kako će da se desi, definisali smo strukturu podataka u koju ćemo da smestimo podatke koji su pristigli. Sada je potrebno poslati neke podatke.

Pronaći u kodu liniju:

|  |
| --- |
| /\* USER CODE BEGIN PV \*/ |

I ispod nje dodati podatke koje želimo da pošaljemo:

|  |
| --- |
| char \*listOfGpsData[] =  {  "$GPRMC,093121.691,A,44.017320,N,20.907265,W,000.0,000.0,300723„,A\*7C\r\n",  "$GPGGA,093121.691,44.017320,N,20.907265,W,1,10,4.00,100.0,M,S0.0,M„\*7A\r\n",  "$GPGSA,A,3,13,11,20,28,14,18,16,21,22,19„,4.00,3.20,2.40\*OB\r\n"  }; |

Ovo su podaci koje bi generisao jedan stvarni GPS modul. U suštini razlikujemo neke 3 vrste podataka:

* GPRMC – (engl. Global Positioning System Recommended Minimum) Sadrži neke podatke kao što su UTC vreme (engl. Coordinated Universal Time), Latitude, Longitude i on nam više daje neke podatke koje su u relaciji sa navigacijom kao što su brzina I kurs na zemlji koji nam nisu baš bitni za naš zadatak.
* GPGGA – Sadrži sve podatke koje sadrži i GPRMC takođe sadrži poziciju, altitude, broj satelita u opsegu, i GPS fix indikatore. Nama će najviše odgovarati ovaj tip.
* GPGSA – Ovaj tip sadrži podatke o satelitima, njihovom preciznošću i slično, on nam takođe neće biti potreban za naš zadatak.

Pošto ćemo se uglavnom fokusirati na GPGGA, objasnićemo redom šta znači koji broj.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| "$GPGGA, 093121.691, 44.017320, N, 20.907265, W, 1, 10,4.00,100.0,M,S0.0,M„\*7A\r\n" | | | | | | | |
| Tip podatka | Vreme, u formatu: HHMMSS,SSS | Latitude | North\_South karakter. | Longitude | West\_East  karakter. | Fix | … |

Sve podatke smo do sada objasnili osim Fix-a, Fix je podatak koji nam u suštini govori da li je GPGGA rečenica validna ili ne.

Pošto nismo u mogućnosti da pročitamo podatke brzinom koje bi jedan GPS modul slao, mi smo sami definisali deo podataka kakve bi jedan GPS modul slao.

Za radi bolje vizualizacije problema, mi ćemo te podatke poslati na virtuelni terminal.

Pronaći u kodu liniju:

|  |
| --- |
| /\* USER CODE BEGIN 0 \*/ |

I u toj sekciji dodati:

|  |
| --- |
| void UART\_SEND(char \*message)  {  HAL\_UART\_Transmit(&huart1, (uint8\_t\*) message, strlen(message), 100);  } |

Kod Transmit funkcije argumenti redom su: na koji UART set se šalje poruka (obično ploče imaju 2 UART seta, dok npr. malo jače ploče kao što je i ploča koju mi koristimo imaju 3 ili više UART setova). Sledeći tj. drugi argument funkcije predstavlja zapravo ta poruka, treći argument je njena dužina, i četvrti argument je timeout predstavljen u milisekundima.

Timeout u smislu: Timeout u slučaju greške, pomaže npr. ako UART nije spreman, nema potrebe slati poruku.

Kada bi koristili Receive funkciju sa timeout-om, npr. možemo da kažemo: ,,E, čekaj mi poruku 5 sekundi, ako je ne dobiješ ne treba da je čekaš više’’.

UART set se sastoji iz jednog RX pina i jednog TX pina.

Sada kada smo omogućili slanje poruka sa TX pina, možemo zapravo i poslati te poruke.

Pronaći sledeću liniju u kodu:

|  |
| --- |
| /\* Infinite loop \*/  /\* USER CODE BEGIN WHILE \*/  while (1) { |

Jer želimo da šaljemo poruke kontinualno.

|  |
| --- |
| while (1) {  UART\_SEND(listOfGpsData[0]);  UART\_SEND(listOfGpsData[1]);  UART\_SEND(listOfGpsData[2]);  HAL\_Delay(500);  }; |

Sa ovim kodom šaljemo poruke na svakih 500ms. Dakle uređaj koji je povezan na naš TX pin bi trebao da vidi ove poruke sada.

Kao što smo već pomenuli nemamo uređaje da pravilno pročitamo duži tekst sa RX pina pa ćemo odmah preskočiti do dola gde obrađujemo ove podatke.

Pošto će nam trebati par funkcija kako bi obradili ove podatke, najbolje bi bilo da napravimo zaseban fajl, biblioteku u kojoj će se to dešavati a onda samo da je uvezemo.

Dodati u *Inc* folder header fajl, *nmea\_parse.h*. A u *Src* folder dodati nmea\_parse.c fajl.

A screenshot of a computer program

Description automatically generated

Kada to sve uradimo imaćemo nešto ovako u header fajlu:

|  |
| --- |
| /\*  \* nmea\_parse.h  \*  \* Created on: Aug 1, 2023  \* Author: vcutu  \*/  #ifndef INC\_NMEA\_PARSE\_H\_  #define INC\_NMEA\_PARSE\_H\_  #endif /\* INC\_NMEA\_PARSE\_H\_ \*/ |

Dok će nam .C fajl biti prazan.

Dodati .h fajlu sledeći kod:

|  |
| --- |
| /\*  \* nmea\_parse.h  \*  \* Created on: Aug 1, 2023  \* Author: vcutu  \*/  #ifndef INC\_NMEA\_PARSE\_H\_  #define INC\_NMEA\_PARSE\_H\_  typedef struct {  float time;  float latitude;  char northSouth;  float longitude;  char eastWest;  int fix;  } GGASTRUCT;  typedef struct {  float time;  float latitude;  char northSouth;  float longitude;  char eastWest;  int fix;  } RMCSTRUCT;  typedef struct {  GGASTRUCT ggaStruct;  RMCSTRUCT rmcStruct;  }GPSSTRUCT;  void nmea\_parse(char \*gps\_data[], GPSSTRUCT \*gpsStruct);  #endif /\* INC\_NMEA\_PARSE\_H\_ \*/ |

Kreirali smo strukture podataka koje ćemo koristiti da smeštamo podatke koje pročitamo. I takođe smo definisali funkciju **nmea\_parse** koju ćemo pozivati u glavnom programu.

Neće nam biti potrebni RMC podaci kako oni sadrže razne stvari poput brzine kretanja, ali u slučaju da nam zatrebaju podaci poput brzine, možemo lako da doradimo kod pa da radimo i sa tim podacima.

Sada preostaje samo da se naprave funkcije za obradu pristiglih podataka, to možemo uraditi ovako,  
u nmea\_parse.C fajlu dodati sledeće funkcije:

|  |
| --- |
| int parse\_GPGGA(char \*GGAbuffer, GGASTRUCT \*ggaStruct) {  int currentData = 0;  //"$GPGGA,093121.691,5404.2675,N,00159.7569,W,1  char \*token = strtok(GGAbuffer, ",");  // token = $GPGGA 093121.691 5404.2675 N 00159.7569 W 1  // irelevant time latitude NS longitude EW fix  char \*marker = token;  while (marker != NULL) {  if(currentData == TIME)  ggaStruct->time = strToFloat(marker);  if(currentData == LATITUDE)  ggaStruct->latitude = strToFloat(marker);  if(currentData == NORTH\_SOUTH)  ggaStruct->northSouth = charToNumber((char)marker[0]);  if(currentData == LONGITUDE)  ggaStruct->longitude = strToFloat(marker);  if(currentData == EAST\_WEST)  ggaStruct->eastWest = (char)marker[0];  if(currentData == FIX)  ggaStruct->fix = charToNumber((char)marker[0]);  marker = strtok(NULL, ",");  currentData++;  }  return 1;  } |

Ukratko, **parse\_GPGGA** funkcija prima GGA rečenicu i strukturu u koju treba da smesti izvučene podatke. Funkcija **strtok** će nam izdeliti rečenicu po zarezima. I onda ćemo samo da prođemo kroz dobijene reči i smestiti ih u odgovarajući podatak u strukturi.

Sledeća funkcija koja nam je potrebna jeste:

|  |
| --- |
| char \*prepareData(char \*nmea\_data) {  int commmaCounter = 0;  int idx = 0;  char \*str = (char\*)malloc(MAX\_GGA\_SIZE \* sizeof(char));  strcpy(str, nmea\_data);  while(commmaCounter < 7) {  if(str[idx] == ',')  commmaCounter++;  idx++;  }  str[--idx] = '\0';  return str;  } |

Ukratko, kao i što sama funkcija kaže pripremamo podatke, radi se o GGA rečenici, mi ćemo na početku da imamo surovu GGA rečenicu:

|  |
| --- |
| "$GPGGA, 093121.691, 44.017320, N, 20.907265, W, 1, 10,4.00,100.0,M,S0.0,M„\*7A\r\n" |

Ova funkcija će samo da izbroji do sedmog zareza a ostatak će da odseče jer nam posle Fix-a ostali podaci nisu potrebni.

I **nmea\_parse** funkcija, koja se poziva u glavnom programu:

|  |
| --- |
| void nmea\_parse(char \*gps\_data[], GPSSTRUCT \*gpsStruct) {  int dataLen = 3;  for (int i = 0; i < dataLen; i++) {  if (strstr(gps\_data[i], "\r\n") != NULL) {  if (strstr(gps\_data[i], "GPGGA") != NULL) {  char \*str = prepareData(gps\_data[i]);  parse\_GPGGA(str, &gpsStruct->ggaStruct);  }  }  }  } |

I da kažemo nešto i o njoj, prihvata niz rečenica, i GPS strukturu, **primetiti** da postoji razlika u strukturama koje ove funkcije primaju, **parse\_GPGGA** prima samo GGA strukturu dok ova funkcija prima čitavu GPS strukturu, kako bi mogli u buduće da dodamo neki novi tip GPS podataka, kao što je npr. RMC.

Ova funkcija prolazi kroz niz rečenica, proverava da li je rečenica u pitanju, zatim koji tip rečenice je u pitanju, ako je GPGGA, onda pripremiti datu rečenicu i poslati **parse\_GPGGA** funkciji da je dalje obradi.

## Haversine funkcija udaljenosti

Potrebna nam je još samo sledeća funkcija a to je Haversine funkcija udaljenosti:

|  |
| --- |
| #define EARTH\_RADIUS\_KM 6371.0  #define TO\_RADIANS(deg) ((deg) \* M\_PI / 180.0)  double haversineDistance(double lat\_1, double lon\_1, double lat\_2, double lon\_2) {  // Convert latitude and longitude from degrees to radians  lat\_1 = TO\_RADIANS(lat\_1);  lon\_1 = TO\_RADIANS(lon\_1);  lat\_2 = TO\_RADIANS(lat\_2);  lon\_2 = TO\_RADIANS(lon\_2);  // Calculate the haversine distance  double dlat = lat\_2 - lat\_1;  double dlon = lon\_2 - lon\_1;  double a = sin(dlat / 2.0) \* sin(dlat / 2.0) +  cos(lat\_1) \* cos(lat\_2) \* sin(dlon / 2.0) \* sin(dlon / 2.0);  double c = 2.0 \* atan2(sqrt(a), sqrt(1 - a));  double distance = EARTH\_RADIUS\_KM \* c;  return distance;  } |

Ovaj kod predstavlja funkciju za izračunavanje udaljenosti između dve tačke na Zemlji, koristimo decimalne stepene geografske širine i dužine tačnije, latitude i longitude. Funkcija uzima četiri ulazna argumenta (geografska širina i dužina za obe tačke) i vraća udaljenost između njih u kilometrima.

# Zaključak

Naveli smo sve najbitnije delove projekta: podešavanje projekta, primanje i slanje podataka pomoću USART-a, parsiranje pristiglih GPS podataka, sada je sve manje-više korisnički doživljaj.

# Literatura:

https://www.theengineeringprojects.com/  
<https://www.st.com>