Elektrotehnički fakultet u Beogradu

Katedra za elektroniku

32-bitni mikrokontroleri i primena

Projekat:

-Analizator Signala-

Mentori:

Prof. Dragan M. Vasiljević

Prof. Nenad Jovičić

Student:

Mladen Cicmil 3210/2012

cicmil.mladen@gmail.com

Beograd, Jun 2013**.**

Sadržaj

1. Uvod

Materijal predstavljen u ovom izveštaju nudi jedno od mogućih rešenja projektnog zadatka, koji se radi iz predmeta 32-bitni mikrokontroleri i primena, na master studijama Elektrotehničkog fakulteta Univerziteta u Beogradu. Glavni zadatak projekta je realizacija jednostavnog analizatota signala. U ovu svrhu je korišćena razvojna pločica STM32VLDISCOVERY [1] koja sadrži mikrokontroler STM32F100RBT6B sa ARM-ovim procesorom Cortex-M3. Specifikacije pomenutih komponenata se mogu naći u [2],[3] i [4]. Aplikacija je napisana u programskom jeziku C u okruženju programskog paketa IAR Embedded Workbench for ARM 6.10 Kickstart [5], a projekat je delimično realizovan u laboratoriji 18. Uz mikrokontroler korišćeni su i uzlazno-izlazni uređaji čiji je opis, način povezivanja i komunikacije sa mikrokontrolerom detaljnije analiziran u narednim poglavljima.

Ceo zadatak, koji je trebalo uraditi, podeljen je na više manjih i konkretnijh problema i zatim se pristupilo njihovom rešavanju. Cilj autora je bio da obezbedi jednostavan, ali siguran i funkcionaln kod, koji implementira analiztor signala na razvojnoj ploči. Ostavljen je i određeni stepen slobode koji je namenjen za poboljšanja i dodatne funkcionalnosti.

Sam izveštaj je podeljen u više celina. Nakon uvodnog dela, dato je drugo poglavlje u kojem je ukratko izložen opis šire oblasti kojoj tema ovog projekta pripada. Treće poglavlje prikazuje idejno rešenje i u kratkim crtama opisuje način funkcionisanja aplikacije. U četvrtom poglavlju je opisana hardverska realizacija sistema. Analizirane su dodatno korišćene komponente i pojašnjeno je njihovo funkcionisanje i povezivanje sa razvojnim sistemom. Peto poglavlje se bavi softverskom realizacijom. U ovom delu su navedeni svi fajlovi koriščeni za realizaciju aplikacije. Za svaki od ovih fajlova navedena je njegova funkcija i opisan je programski kod koji ga prati. Na samom kraju petog poglavlja prikazan je dijagram toka glavnog programa i objašnjeni su svi koraci prilikom njegovog izvršavanja. Rezultati i realni rad programa objašnjeni su u šestom poglavlju. Zaključci autora kao i osvrt na ceo zadatak dati su u sedmom poglavlju, dok je literatura, korišćena za izradu rada, data u osmom poglavlju. Na kraju rada se nalazi prilog sa source kodom aplikacije, pisanom u programskom jeziku C.

2. Opis oblasti kojoj tema projekta pripada

U elektrotehničkoj struci često se javlja potreba za analizom nepoznatog signala kako bi se on učinio poznatim i kako bi se odredili njegovi relevantni parametri. Upravo se u ovu svrhu koriste analizatori signala. Zajedničko za sve uređaje jeste da imaju složenu konstrukciju, koja predstavlja spoj pouzdanog hardvera i naprednog softvera. Danas su sve više u upotrebi digitalni analizatori, koji su istisnuli stare i jednostavnije analoge i čiji je rad zasnovan na korišćenju mikrokontrolera. Signal koji se posmatra, analizatori najpre digitalizuju, filtriraju, a potom i obrađuju digitalnim tehnikama u frekvencijskom domenu. Napredak u ovoj oblasti je omogućio i pogodnosti u vidu memorisanja rezultata i lakšem prenosu podataka na računar. Pored toga, izvrešena merenja i svi relevantni parametri se prikazuju na displeju, koji je sastavni deo ovih uređaja. Analizatori mogu, osim osnovnih merenja (min, max, mean, RMS…) da odrede i prikažu spektar signala, da sračunaju paramete vezane za snagu i distorziju kao i da izvrše analizu šuma i slučajnih procesa. Kao što se može zaključiti, analizatori signala su u mogućnosti da pruže i neke od funkcionalnosti koje se inače sreću kod multimetara, osciloskopa i spektralnih analizatora pa zato i spadaju u skuplju elektrotehničku opremu.

Kako je analiza signala osnovno sredstvo rada u mnogim oblastima tehnike, onda i ne čudi što na tržištu postoji veliki broj ovih uređaja različitih poizviđača. Oni se među sobom razlikuju po parametrima koji direktno utiču na njihovu primenu u pojedinim zadacima. Prilikom izbora analizatora neophodno je voditi računa o opsegu učestanosti ulaznog signala koje je moguće izmeriti, a da ne dođe do izobličenja i distorzije, o kvalitetu integrisanog analogno digitalnog konvertora (rezoluciji, brzini konverzije…), o kvatlitetu procesorske jedinice (učestanost rada, veličina memorije…), kao i o algoritmima i tehnikama koje se koriste za obradu signala. U osmom poglavlju se pod stavkama [13], [14] i [15] može naći prikaz nekih analizatora i njihove specifikacije.

3. Idejno rešenje

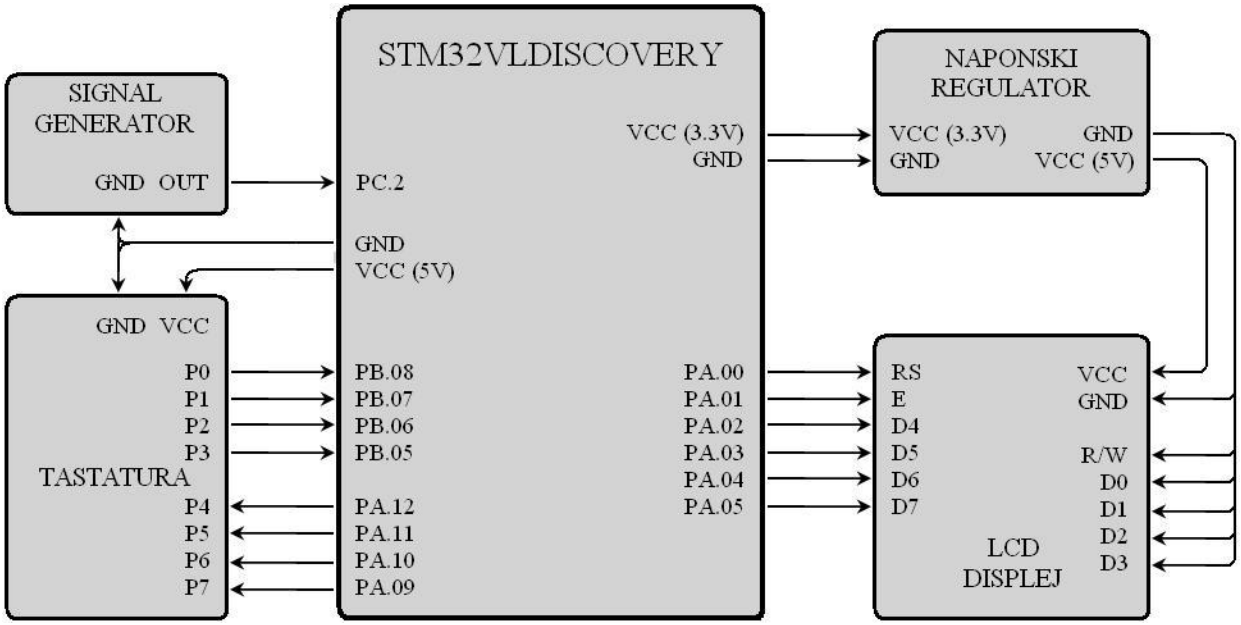
Zadatak ovog projekta je bio da se napravi jednostavan analizator signala. Za razliku od komercijalnih, znatno složenijih i skupljih uređaja, ovaj analizator je realizovan na jednostavnom hardveru i vrši prostiju analizu, odradom u vremenskom domenu. Da bi se analiza uopšte i izvršila neophodno je da ulazni signal bude periodičan sa frekvencijom u opsegu od 4687 do Hz do 18750 Hz, kao i da mu se vrednosti nalaze između 0V i 3V, približno. Ukoliko su prethodno pomenuti uslovi zadovoljeni, analizator može da vrši obradu. Na njemu je implementirano ukupno 6 funkcionalnosti. Moguće je naći minimalnu, maksimalnu, srednju vrednost signala kao i RMS, energiju i frekvenciju signala.

Osnovna ideja na kojoj počiva izrada ovog projekta je jednostavna. Analogno digitalni konvertor (ADC) se konfiguriše tako da neprekidno vrši semplovanje i konverziju ulaznog signala, koji se dobija sa signal generatora. Nakon svake konverzije, direct memory access

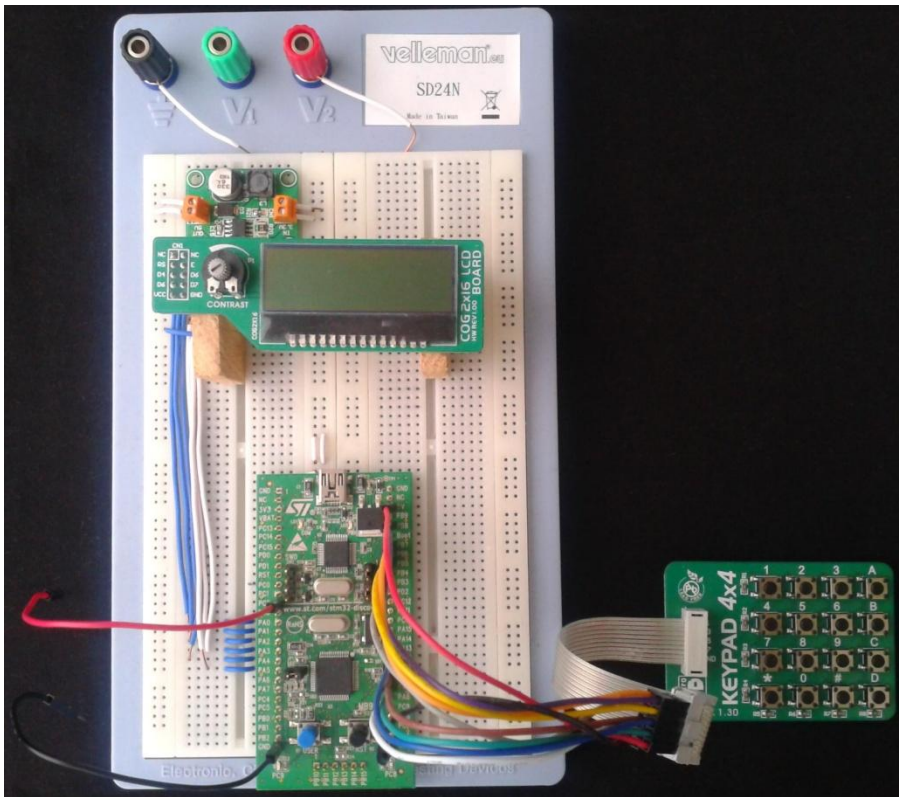
(DMA) kontroler vrši prenos digitalnog reprezenta ulaznog signala u cirkularni bafer. Ovaj preoces se dešava u pozadini i ne remeti izvršavanje glavnog programa, koji je realizovan u vidu mašine stanja. Glavni program najpre vrši očitavanje tastature. Ukoliko korisnik nije pritisnuo nijedan taster ili ako je pritisnuo više njih u isto vreme, tada se prelazi u drugo stanje u kome se vrši kopiranje odbiraka koji se nalaze u cirkularnom baferu. Naime, kada se polovina ovog bafera napuni, sve vrednosti iz te tekuće polovine se kopiraju u pomoćni bafer u memoriji. Na ovaj način je omogućeno da aplikacija neprekidno vrši akviziciju signala, bez propuštanja odbiraka, a da u memoriji uvek postoje „sveži" podaci nad kojima može da se izvrši obrada. Zatim se aplikacija opet vraća na očitavanje tastature. Ako je sada korisnik pritisnuo određeni taster, glavni program prelazi u treće stanje u kome se vrši obrada podataka (koju je korisnik izabrao) nad elementima pomoćnog bafera. Rezultat procesiranja se ispisije na LED displeju i na njemu ostaje sve dok korisnik ne izabere novu obradu.

4. Opis hardvera

Kao što je prethodno rečeno, polazna komponenta korišćena za realizaciju ovog projekta je razvojna pločica STM32VLDISCOVERY. Sama po sebi ona nije bila dovoljna za realizaciju analizatora signala pa su upotrebljeni i drugi moduli, među koje spada multipleksirana tastatura 4x4, inteligentni LCD displej, naponski regulator i signal generator. Pojednostavljena šema svih pomenutih komponenata, ali i prikaz načina na koji su one povezane data je na slici 1. Realni izgled sistema (bez signal generatora) je predstavljen na slici 2.



Slika 1. Blok šema sistema



Slika 2. Prikaz realnog sistema korišćenog za implemetaciju analizatora signala

4.1. STM32VLDISCOVERY

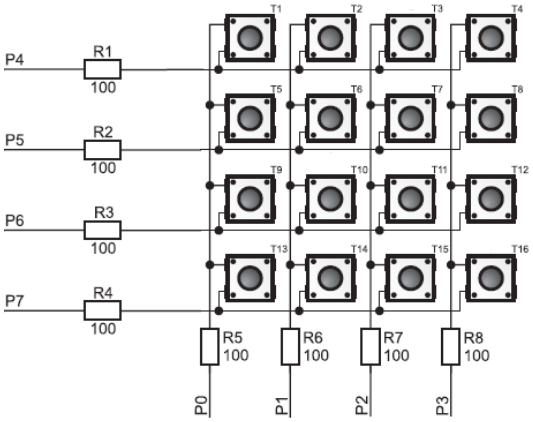
Srce ovog razvojnog sistem čini STM32F100RBT6B mikrokontroler, baziran na ARM-ovom Cortex M3 jezgru. Mikrokontroler ima 128KB flash memorije i 8KB RAM-a, a maksimalna frekvencija rada mu je 24MHz. Programiranje mikrokontrolera se vrši pomoću računara sa kojim se veza ostvaruje korišćenjem USB-a. Preko ove veze se obezbeđuje i napajanje celokupnog razvojnog sistema. Pored pomenutog mikrokontrolera, razvojna pločica sadrži ST-LINK i SWD konektor, koji su namenjeni debagovanju softvera kao i dva tastera i dve diode. Sistem sadrži i 64 GPIO pina koji služe za povezivanje dodatnih periferija sa mikrokontrolerom. Na slici 3 može se videti izgled kompletnog sistema korišćenog u ovom projektnom zadatku.

Što se tiče resursa upotrebljenih prilikom implementacije analizatora signla, korišćeni su I/O porovi, ADC, DMA kontroler, basic tajmer (TIM7), nested vectored interrupt (NVIC) kontroler i dve LED diode.

Slika 3. STM32 discovery evaluation board

4.2. Tastatura

Tastatura predstavlja deo korisničkog interfejsa preko koje se vrši odabir jedne od implementiranih funkcionalnosti analizatora. Njena šema je data na slici 4, dok se na slici 1 može videti način povezivanja tastature sa mikrokontrolerom. Kao što se može primetiti, tastatura je matričnog tipa. Linije označene sa P4, P5, P6 i P7 su uzete kao selekcione linije, dok su P3, P2, P1 i P0 uzete kao linije sa kojih se čita kod pritisnutog tastera. Princip rada tastature je jednostavan. Na izlazne pinove mikrokontrolera PA.12, PA.11, PA.10, PA.09 (konfigurisane kao push-pull) se dovodi šetajuća jedinica, čime se naizmenično selektuje po jedna vrsta (linija) matrične tastature. Na ulaznim pinovima PB.08…PB.05 (konfigurisanim kao input pull-down) se očitavaju naponski nivoi, nakon selektovanja određene linije. Ukoliko je pritisnut neki taster u selektovanoj vrsti, na ulaznom pinu koji odgovara koloni u kojoj se taj pritisnuti taster nalazi će se očitati visok logički nivo, a u suprotnom biće očitana logička nula (zbog pull down konfikuracije ulaznih pinova). Na taj način se određuje kod (vrsta i kolona) pritisnutog tastera.

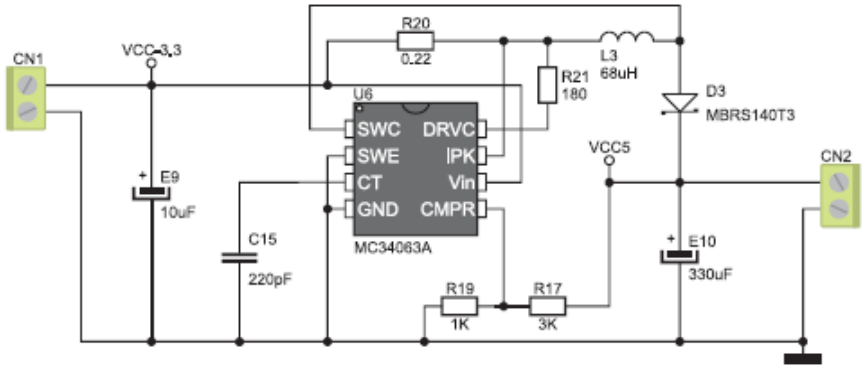


Slika 4. Šema tastature 4x4 [6]

Treba napomenuti da tastatura sadrži i niz otpornika koji predstavljaju zaštitu od kratkog spoja do koga može doći kada se pritisnu taster u selektovanom redu i taster u istoj koloni, ali u nekom drugom redu.

4.3 Naponski regulator

Naponski regulator predstavlja komponentu koja ima ulogu da od napona 3.3V, dostupnog na razvojnom sistemu, napravi napon od 5V, potrebnih za napajanje LCD displeja. Šema ovog naponskog regulatora je prikazana na slici 5, a njegovo povezivanje sa ostatkom hardvera je dato na slici 1.

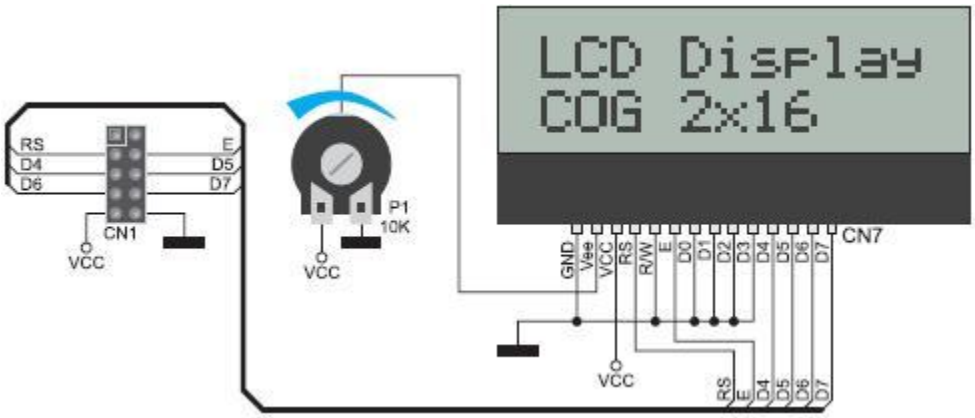


Slika 5. Šema naponskog regulatora [9]

Može se primetiti da je glavni deo ovog regulatora zapravo step-up konvertor koji vrši naponsku translaciju, dok ostatak komponenti služi da obezbedi potrebne uslove za pravilan rad ovog konvertora.

4.4 LCD 2x16

LCD displej čini, zajedno sa tastaturom, deo korisničkog interfejsa. Služi za prikaz rezultata odrade, koju je korisnik zahtevao. Konkretni displej može da prikaže ukupno 32 alfanumerička karaktera, raspoređena u dva reda od po 16. Svaki karakter se predstavlja pomoću matrice od ukupno 5x7 piksela. Na slici 6 je prikazana šema LCD-a na kojoj se mogu uočiti i signali važni za njegov rad.



Slika 6. Šema LCD displeja [7]

Treba napomenuti i činjenicu da je ovo inteligentni LCD, što znači da mikrokontroler šalje heksadecimalnu vrednost alfanumeričkog karaktera (kojim je on predstavljen u ASCII formatu), a ovaj podatak se uzima kao adresa u internoj memoriji LCD-a na kojoj se čuva raspored piksela za željeni karakter.

Veza između LCD displeja i mikrokontrolera je izvršena na malo drugačiji način nego što je to uobičajeno i može se videti na slici 1. Naime, prema specifikaciji displeja, za njegov ispravan rad potrebno je da naponski nivo logičke jedinice iznosi 5V, a mikrokontroler tu vrednost ne može da isporuči na svojim pinovima, već samo 3.3V. Ovo praktično znači da je neophodno koristiti dodatno kolo koje će vršiti translaciju naponskih nivoa svih linija koje se koriste za vezu mikrokontrolera i LCD displeja. Međutim, u praksi se pokazuje da mikrokontroler može da radi i sa naponskim nivoima od 3.3V što eliminiše potrebu za translatorom napona. Zbog svega pomenutog, komandne linije i linije za podatke LCD displeja su direktno povezane sa odgovarajućim pinovima mikrokontrolera (konfigurisanim kao outut push-pull). Kada je reč o glavnom napajanju LCD-a (Vcc), ono mora da iznosi 5V jer od ovog napona direktno zavisi osvetljenje displeja. Kada bi se i ovde uzelo 3.3V alfanumerički karakteri ne bi bili vidljivi na LCD-u. Kako na razvojnoj pločici postoji pin koji može da isporuči nominalno 5V logično bi bilo da se on iskoristi za napajanje displeja. Međutim od realno isporučuje 4.7 volti i potencijalno bi pravio problem u osvetljaju. Zbog toga je on iskorišćen za vezu sa tastaturom dok je stabilno napajanje od 5V za LCD-a obezbeđeno preko naponskog regulatora opisanog u prethodnom poglavlju.

LCD displej korišćen u izradi ovog projekta radi isključivo u 4-bitnom režimu i koristi 6 linija za komunikaciju sa mikrokontrolerom. Podaci se prenose linijama D4, D5, D6 i D7, dok su D0, D1, D2, D3 neaktvne i vezane su za masu. Komandne linije su enable (E) - označava da su podaci prisutni na odgovarajućim linijma validni, RS – nosi informaciju o tome da li preneti niz bitova predstavlja instrukciju (logička nula) ili podatak (logička jedinica), i read/write (R/W) koja određuje da li je u pitanju ciklus upisa (logička nula) ili čitanja (logička jedinica). Kako u ovom projektu nije potrebno čitati iz LCD displeja, R/W je vezan za GND.

5. Softverska implementacija rešenja

Softver za ovaj projekat je pisan u programskom jeziku C. Radi bolje organizacije i preglednosti aplikacija je dodeljena u više c fajlova, praćenih odgovarajućih heder fajlovima. U njima su sadržani deklaracije, prtototipovi i kodovi koji se odnose na pojedine periferije ali i logičke celine korišćene u zadatku. Treba dodati i da je prilikom pisanja koda posebna pažnja posvećena tome da kod bude čitljiv i razumljiv. Stoga su korišćeni različiti makroi, konstante, funkcije… U prilogu se mogu videti svi fajlovi korišćeni u ovom projektu.

5.1 Portovi

U odgovarajućem header fajlu se nalaze deklaracije svih portova i pinova korišćenih u zadatku. Za LCD su upotrebljeni pinovi porta A (u output push pull konifiguraciji), za tastaturu pinovi porta A (output push pull) i porta B (input pull down), dok je za ADC (analog input) i LED diode (output push pull) iskoršćen port C. U pratećem C fajlu je napisana funkcija *Inicijalizacija_Portova* kojom se vrši inicijalizacija svih portova i pinova. Ona se poziva prilikom startovanja cele aplikacije, na njenom početku. Kod ove funkcije se nalazi u Prilogu A.1.

5.2 RCC

Podešavanjem Reset and Clock Control grupe registara se definiše CLK signal za pojedine resurse. U C fajlu koji prati RCC (Prilog A.2) definisana je funkcija *CLK_Periferija* kojom se clock signal dovodi do svih periferija. Treba napomenuti da je učestanost CLK signala za ADC preskalirana tako da iznosi 6MHz umesto maksimalnih 12MHz. Ovaj izbor je napravljen kako bi ADC sigurnije i pouzdanije obavljao svoju funkciju.

5.3 NVIC

U fajlu nvic_m.c (Prilog A.3) definisana je funkcija kojom se vrši konfiguracija NVIC kontrolera. Ona se poziva tokom procesa inicijalizacije sistema. Ukupno je definisano 3 prekida, za DMA prenos, ADC i tajmer pri čemu su DMA i ADC su većeg prioriteta. O detaljima prekidnih rutina biće reči u narednim poglavljima.

5.4 ADC

Analogno digitalni konvertor predstavlja ulazni deo sistema. On je zadužen da vrši semplovanje i memorisanje odbiraka ulaznog signala kojeg ispitujemo. Nepoznati signal (dobijen sa signal generatora) se dovodi na 12. kanal konvertora. ADC se startuje softverski i konfigurisan je tako da radi u scan i continous modu. Ovo znači da se nakon izvršene jedne konverzije, odmah se započinje sa narednom i ovaj proces se vrši sve dok se ne prekine izvršavanje celog programa. Takođe, ADC radi u tandemu sa DMA kontrolerom tako da se nakon svake konverzije, podatak iz internog registra ADC-a prebacuje u cirkularni bafer u memoriji mikrokontrolera. Na ovaj način je obezbeđeno da nema propuštanja odbiraka ulaznog signala. ADC je podešen da radi sa sampling time-om od 7.5 ciklusa. Ako tome pridodamo i činjenicu da ADC ima učestanost rada od 6MHz dolazi se do podatka da je učestanost odabiranja ulaznog signala 300KHz. Kako bi se sistem zaštitio od neodgovarajućih vrednosti ulaznog signala aktiviran je analog watchdog. Ukoliko se definisane granice watchdog-a premaše tada se generiše prekid i skače se u odgovarajuću rutinu, pojašnjenu u 5.9.

Sva podešavanja ADC-a su izvršena u funkciji *ADC_Inicijalizacija* (Prilog A.4) koja se poziva prilikom inicijalizacije sistema.

5.5 DMA

DMA kontroler je izuzetno bitan za funkcionisanje cele aplikacije. Podešen je da vrši kopiranje digitalne vrednosti ulaznog signala, koja se nalazi u data registru ADC-a u cirkularni bafer. Ovaj proces se dešava u pozadini i ne zahteva učešće procesora koji je na taj način slobodan da izvršava glavni program. Konfiguracija DMA je definisana funkcijom *DMA_Inicijalizacija* u fajlu dma_m.c (Prilog A.5)

Cirkularni bafer ima ukupno mesta za 128 odbiraka. Ova vrednost je određena eksperimentalno i predstavlja najveću vrednost (stepena dvojke), koju sistem dopušta tj. za koju aplikacija funkcioniše. Ukoliko se izabere neka veća vrednost tada postoji problem sa kompajliranjem koda sa računara u fleš memoriju mikrokontrolera i aplikacija neće da radi. Broj odbiraka u cirkularom baferu je važan i zbog toga što direktno utiče na frekvencijski opseg u kom treba da se nalazi signal koji ispitujemo, kao i na tačnost merenja. U delu koji govori o glavnom programu će biti više reči o ovome.

DMA je podešen i da generiše prekid kada se desi transfer error. Na ovaj način se aplikacija štiti od grešaka i korumpiranih podataka.

5.6 Tajmer

U ovom projektnom zadatku koristi se basic tajmer, TIM7. Ovaj jednostavni tajmer je upotrebljen kako bi se implementirala funkcija čekanja, neophodna za rad LCD displeja. Naime prilikom upisa podataka u memoriju LCD-a potrebno je sačekati određeni vremenski period pre slanja narednog podatka, kako ne bi došlo do kolizije, a samim tim i grešaka unutar displeja. Ovo je posebno važno prilikom inicijalizacije LCD-a.

Tajmer, tačnije njegov preskaler je konifgurisan tako da TIM7 inkrementira svoj brojač na svaku μs. Prilikom pozivanja funkcije *Čekaj* navodi se parametar n, koji indirektno određuje vrednost do koje će brojač tajmera da ide. Ovaj parametar se prvo množi sa faktorom 100, tako da tajmer radi ukupno $n \cdot 100$ μs. Kada doće do kraja, tačnije do overflow-a, generiše se prekid u kome se zaustavlja tajmer. Na prethodno opisani način moguće je precizno kontrolisati vreme čekanja, odnosno pauze između slanja podataka LCD-u. Detaljan prikaz funkcije *Čekaj* se može videti u C fajlu koji prati tajmer (Prilog A.6).

5.7 Tastatura

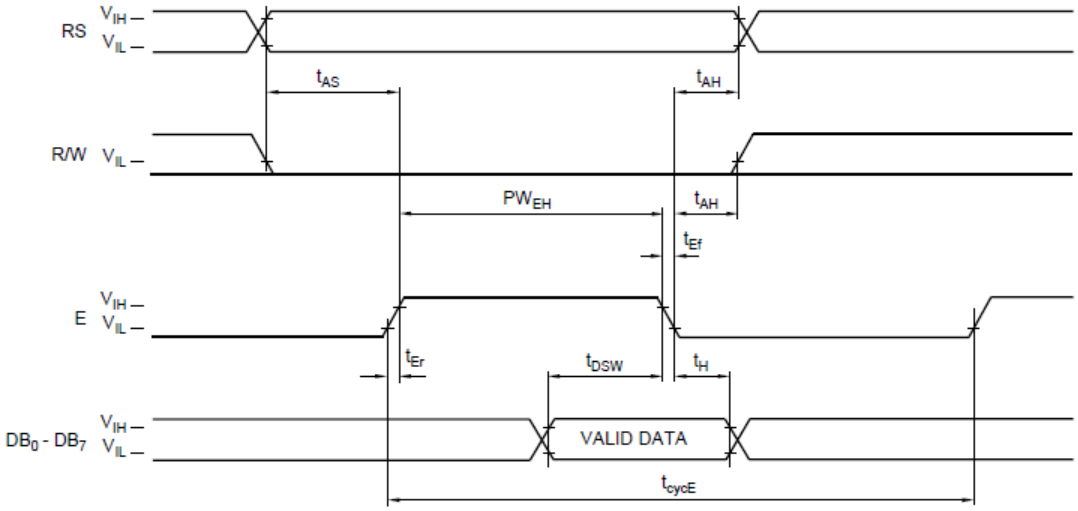
U c fajlu koji prati tastaturu (Prilog A.7) definisana je funkcija *Očitaj_Tastaturu* koja, kao što joj i ime kaže vrši skeniranje tastature i vraća kod pritisnutog tastera. Ukoliko korisnik nije pritisnuo nijedan taster funkcija vraća nulu, u suprotnom vraća informaciju o vrsti i koloni u kojoj se pritisnuti taster nalazi.

Sama funkcija je organizovana u vidu petlje u kojoj se redom selektuju vrste tastature. Kada je na određenu vrstu dovedena logička jedinica, iščitaju se naponski nivoi koji odgovaraju kolonama. Ukoliko je neka kolona na visokom nivou, to ukazuje da postoji pritisnuti taster pa se njegov kod (tekuća vrsta, tekuća kolona) pamti. U okviru funkcije implementirana je jednostavna zaštita od istovremenog pritiska više tastera. Ukoliko dođe do ove pojave, ona se ignoriše i funkcija vraća nulu, baš kao i u slučaju kad korisnik nije pritisnuo nijedan taster.

5.8 LCD

Komunikacija između mikrokontrolera i LCD displeja zahteva implementaciju određenih protokola i poštovanje specifičnih vremenskih rokova. Prema specifikaciji LCD-a [8], postoje razlike u načinu signaliziranja prilikom slanja podataka, komandi, inicijalizacije. Stoga je u okviru LCD.c fajla napisano nekoliko funkcija kako bi komunikacija sa LCD-om bila efikasnija i razumljivija.

Funkcija *Slanje_Informacije* je bazična funkcija na koju se oslanjaju sve ostale. Njoj se prosleđuje 8-bitna informacija (podatak ili komanda) iz koje ona uzima gornjih 4 bita i zatim ih šalje na izlazne pinove mikrokontrolera povezane sa LCD-om.



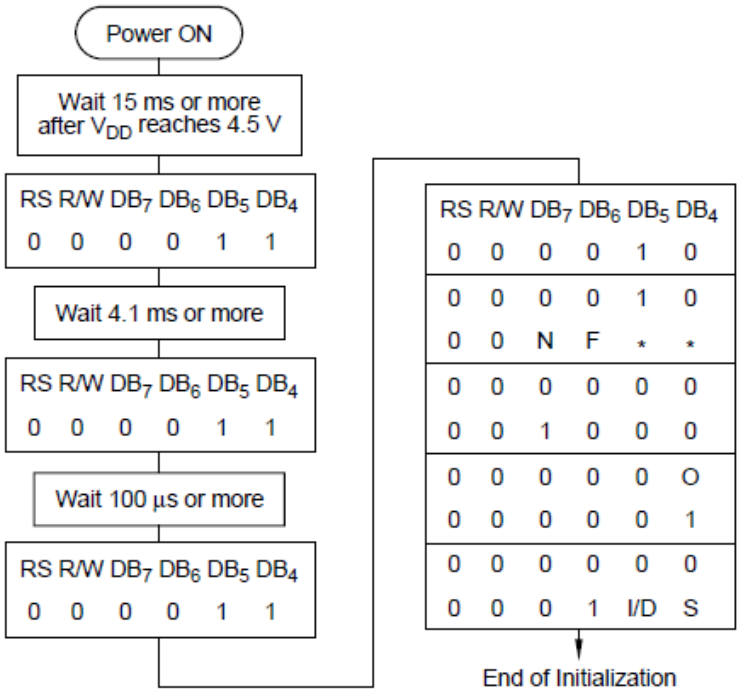
Slika 7. Slanje informacije LCD displeju [8]

Prilikom slanja podataka odnosno komandi potrebno je ispoštovati slične procedure, koje se jasno mogu uočiti na slici 7. Jedina razlika postoji u signalu RS koji nosi informaciju o tome da li preneti niz bitova predstavlja instrukciju (logička nula) ili podatak (logička jedinica). Radi preglednosti, ali i zbog pojednostavljenja komunikacije, napisane su različite funkcije *Slanje_Podatka* i *Slanje_Komande* kako bi se LCD-u dostavile različite informacije. Pomenute funkcije su napisane u skladu sa protokolom sa slike 7, tako da aktiviraju signale RS, E, D4, D5, D6 i D7 prema naznačenom redosledu i u naznačenom vremenskom intervalu (slika 8).

PARAMETER		SYMBOL	VALUE		UNIT
			MIN.	MAX.	
Enable Cycle Time		t_{CYCE}	1000	—	ns
Enable Pulse Width	"High" Level	PW_{EH}	450	—	ns
Enable Rise/Fall Time		t_{Er}, t_{Ef}	—	25	ns
Setup Time	RS, R/W-E	t_{AS}	140	—	ns
Address Hold Time		t_{AH}	10	—	ns
Data Setup Time		t_{DSW}	195	—	ns
Data Hold Time		t_H	10	—	ns

Slika 8. Vremenski parametri upisa u LCD [8]

Da bi LCD displej pravilno radio, neophodno je da se inicijalizuje na odgovarajući način. Koraci koje je potrebno izvršiti prilikom inicijalizacije su dati na slici 9. Funkcija *LCD_Inicijalizacija* implementira ovu proceduru i poziva se na početku izvršavanja programa, pre beskonačne petlje.



Slika 9. Inicijalizacija LCD displeja [8]

Kada se završi obrada koju je korisnik zahtevao, rezultat je potrebno prikazati na LCD displeju. U ovu svrhu se koriste funkcije *Slanje_Teksta* i *Slanje_Broja*. Funkcija *Slanje_Teksta* se koristi i u slučaju nekih neregularnih situacija kada je potrebno izvestiti korisnika o nastalom problemu. Kao parametar prilikom poziva ove funkcije, navodi se red u kome je potrebno ispisati poruku, kao i tekst poruke. Tekst se ne posmatra kao string, već kao skup alfanumeričkih karaktera koji se zatim jedan po jedan šalju LCD-u pomoću funkcije *Slanje_Podatka*. Pošto LCD radi u 4-bitnom režimu, svaki karakter zahteva pozivanje funkcije *Slanje_Podatka* dva puta između čijih pozivanja se vrši šiftovanje niža 4 bita podatka na mesto viša 4 bita.

Funkcija Slanje_Broja služi za ispis brojnih vrednosti odnosno rezultata obrade. Kao parametri, ovoj funkciji se navode red u kome je potrebno ispisati broj kao i sam broj. Cifre, koje želimo da prikažemo se najpre konvertuju u string korišćenjem funkcije *sprintf*. Ovaj string se zatim ispisuje na displeju pozivom funkcije *Slanje_Teksta*.

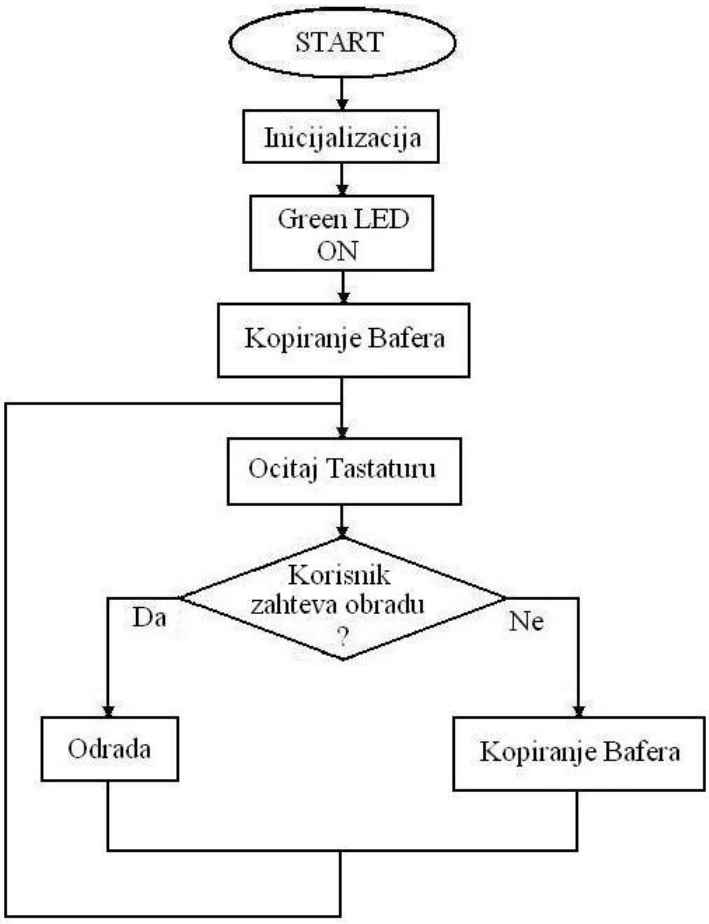
Za brisanje tekućeg sadržaja na displeju, koristi se funkcija *LCD_Brisanje*. Ona zapravo vrši poziv funkcije Slanje_Teksta kojoj se kao parametar za ispis prosleđuje prazan string.

5.9 Prekidi

U realizaciji analizatora signala koriste se 3 prekidne rutine. Jedna rutina je vezana za tajmer TIM7 i ona je najnižeg prioriteta. U njoj se vrednost promenljive state postavlja na nulu čime se zaustavlja rad tajmera. Prekidne rutine za ADC i DMA su potpuno analogne. Ukoliko je došlo do greške prilikom prenosa podataka (DMA transfer error) ili se ulazni signal ne nalazi u okvirima određenim analog watchdog-om generiše se odgovarajući prekid. U okviru prekidnih rutina vrši se inkrementacija brojača zvanih *brojač_greške_ADC* i *brojač_greške_DMA* koji nose informaciju koliko puta se određena greška desila. Vrednost ovih brojača se kasnije analizira u glavnom programu i ako se utvrdi da je prešla određenu granicu, prekida se sa izvršavanjem aplikacije i korisnik se obaveštava o nastaloj situaciji.

5.10 Glavni program

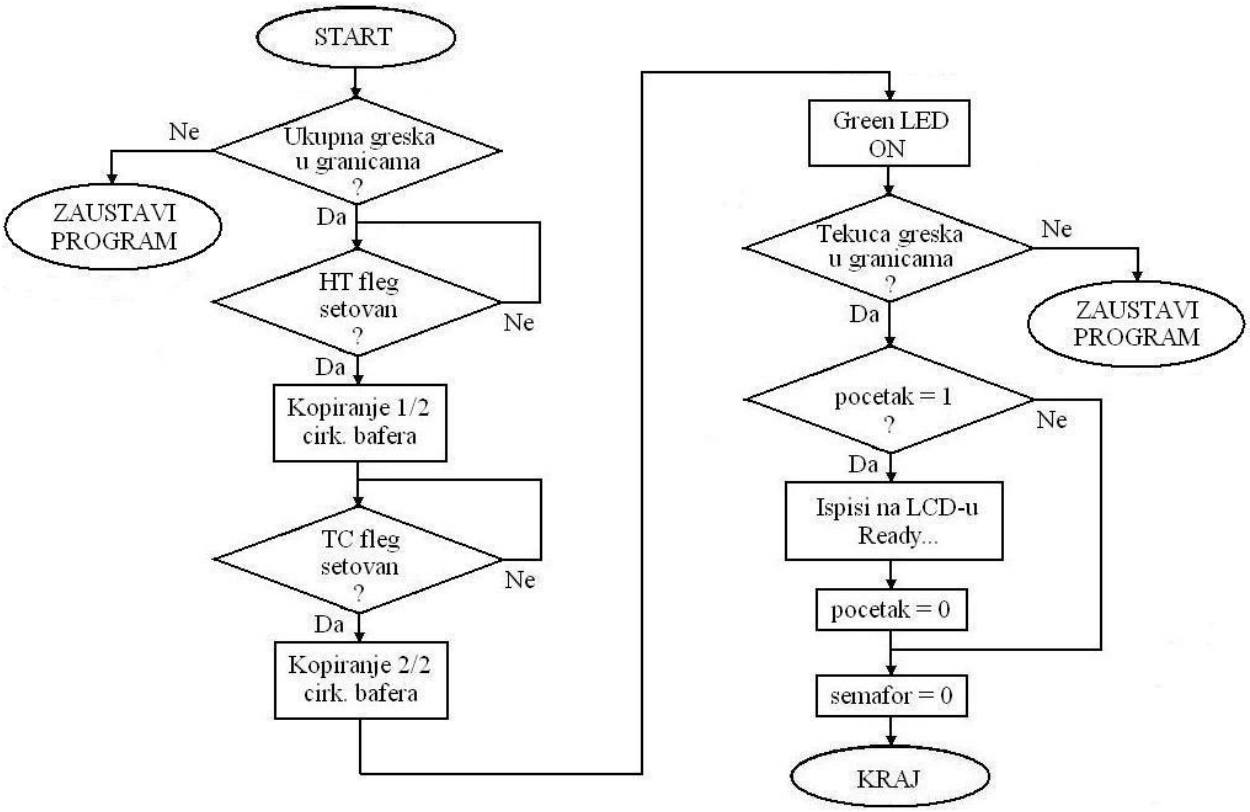
Na samom početku glavnog programa vrši se inicijalizacija svih resursa koji su korišćeni u aplikaciji i njihovo podešavanje. Pod ovim se podrazumeva dovođenje clock signala svim periferijama, konfiguracija portova i pinova upotrebljenih u zadatku, konfiguracija NVIC-a, LCD-a, DMA-a i ADC-a. Kao pokazatelj da se proces inicijalizacije završio i da je sve prošlo kako treba pali se zelena dioda (LED3). Aplikacija zatim prelazi na izvršavanje beskonačne petlje, koja je realizovana u vidu mašine stanja. Prelazak iz jednog u drugo stanje određen je vrednošću promenljive semafor. Dijagram toga izvršavanja čitave aplikacije je dat na slici 10.



Slika 10. Dijagram toga celog programa

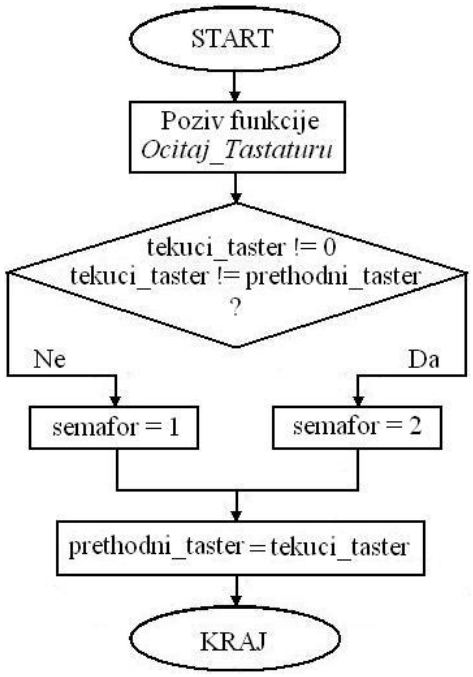
Nakon uspešno izvršene inicijalizacije, semafor ima vrednost 1 i program ulazi u stanje koje je nazvano Kopiranje Bafera. Koraci koji se vrše u ovom stanju su prikazani na slici 11. Na početku ovog stanja vrši se provera ukupne greške DMA prenosa i ukupne greške analog watchdog-a. Svaki put kad se desi odgovarajući prekid izazvan greškom (DMA Transfer Error ili Analog Watchdog), brojači koji se nalaze u prekidnim rutinama i koji se pojašnjeni u poglavlju 5.9 se inkrementiraju. U promenljive nazvane *ukupna_greška_DMA* i *ukupna_greška_ADC* se kopira vrednost ovih brojača, tako da će u njima biti sadržana informacija koliko se grešaka ukupno desilo od početka rada programa. Kako bi se izvršila provera pomenutog broja grešaka kao i da bi se ustanovilo da li je taj broj veći od neke predviđene granice napisana je funkcija *Greška*. Njoj se kao parametri prosleđuju vrednosti *ukupna_greška_DMA* i *ukupna_greška_ADC*, maksimalan broj dozvoljenih grešaka kao i tekst poruke koji se prikazuje na LCD-u ukoliko postoji neregularno stanje. Ako je granica za greške premašena, prekida se rad čitave aplikacije jer su podaci (vrednosti odbiraka ulaznog signala) nepouzdani i nikakva odrada nad ovakvim podacima ne sme da se dozvoli. Ukoliko je greška u dozvoljenim granicama nastavlja se sa izvršavanjem aplikacije. Tada se brišu flegovi Half Transfer (HT) i Transfer Complete (TC) koji ukazuju da je cirkularni bafer bio napunjen do polovine odnosno do kraja. Na ovaj način ignorišemo sve vrednosti ulaznog signala koje su se nalazile u baferu i

posmatramo samo one vrednosti koje DMA prenese nakon brisanja flegova. Program sada čeka da se ponovo generiše HT fleg. Kada se ovo desi, sve vrednosti iz prve polovine cirkularnog bafera se množe sa odgovarajućim faktorom skaliranja i smeštaju se u pomoćni bafer u memoriji, nazvan bafer_1. Kao faktor skaliranja se uzima 2.99/4096 jer se ADC napaja pomoću 0V i 2.99V (utvrđeno direktnim merenjem na razvojnim sistemu) i ima 12-obitnu rezoluciju. Isti postupak se ponavlja i u slučaju kada se napuni ceo cirkularni bafer tj. kao se aktivira TC fleg. Tada se skaliraju i kopiraju vrednosti iz druge polovine cirkularnog bafera u drugu polovinu pomoćnog bafera (bafer_1). Sada je bafer_1 napunjen „svežim" podacima nad kojima je moguće izvršiti obradu. Prethodno opisani način rada se koristi kako bi se relevantni podaci za obradu prikupili samo kada se program nađe u stanju semafor=1, Kopiranje Bafera. U preostala dva stanja se takođe vrši odabiranje i DMA prenos, ali se ti odbirci ignorišu. Na ovaj način se ostvaruje precizna kontrola nad prikupljanjem podataka. Kada je ovaj proces završen vrši se još jedna provera grešaka. Ovoga puta se posmatra tekuća greška tj. broj grešaka koji se desio u toku punjenja pomoćnog bafera, a ne ukupna greška kao što se to čini na početku ovog stanja. Ako nije bilo problema u tekućem ciklusu togluje se plava dioda (LED4) kao naznaka da je sve prošlo bez problema. Na kraju ovog stanja se nalazi deo koda koji se izvršava samo u slučaju da je aplikacija tek pokrenuta. Na displeju se tada ispisuje tekst Ready... koji ukazuje korisniku da je inicijalizacija uspešno urađena kao i da u memoriji postoje validne vrednosti nad kojima može da se izvrši obrada. Promenljiva semafor se potom postavlja na nulu i program prelazi u naredno stanje.



Slika 11. Dijagram toga za stanje Kopiranje Bafera

Nakon kompletiranja stanja semafor=1, aplikacija uvek prelazi u stanje semafor=0, nazvano Očitavanje Tastature. U ovom delu programa se prikuplja kod pritisnutog tastera i na osnovu njega se odlučuje u kom će se pravcu aplikacija dalje izvršavati. Najpre se poziva funkcija *Očitaj_Tastaturu* koja vraća kod (vrsta, kolona) pritisnutog tastera ili vraća 0 ukoliko korisnik nije pritisnuo nijedan taster ili je pritisnuo više tastera istovremeno. Zatim se kod tekućeg tastera poredi sa kodom prethodno pritisnutog tastera, koji se čuva u memoriji. Ako su različiti i ako tekući taster nije 0, to ukazuje da je korisnik zahtevao neku obradu i promenljiva semafor se postavlja na 2. U svim drugim situacijama se semafor postavlja na 1. Ovim se postiže da kada korisnik ništa ne zadaje aplikaciji, ona vrti program između stanja 0 i 1, odnosno očitavanja tastature i prikupljanja podataka za obradu. Tek kada korisnik unese neku komandu, skače se u stanje 2 u kom se vrši željena obrada, slika 12.



Slika 12. Dijagram toka za stanje Ocitaj Tastaturu

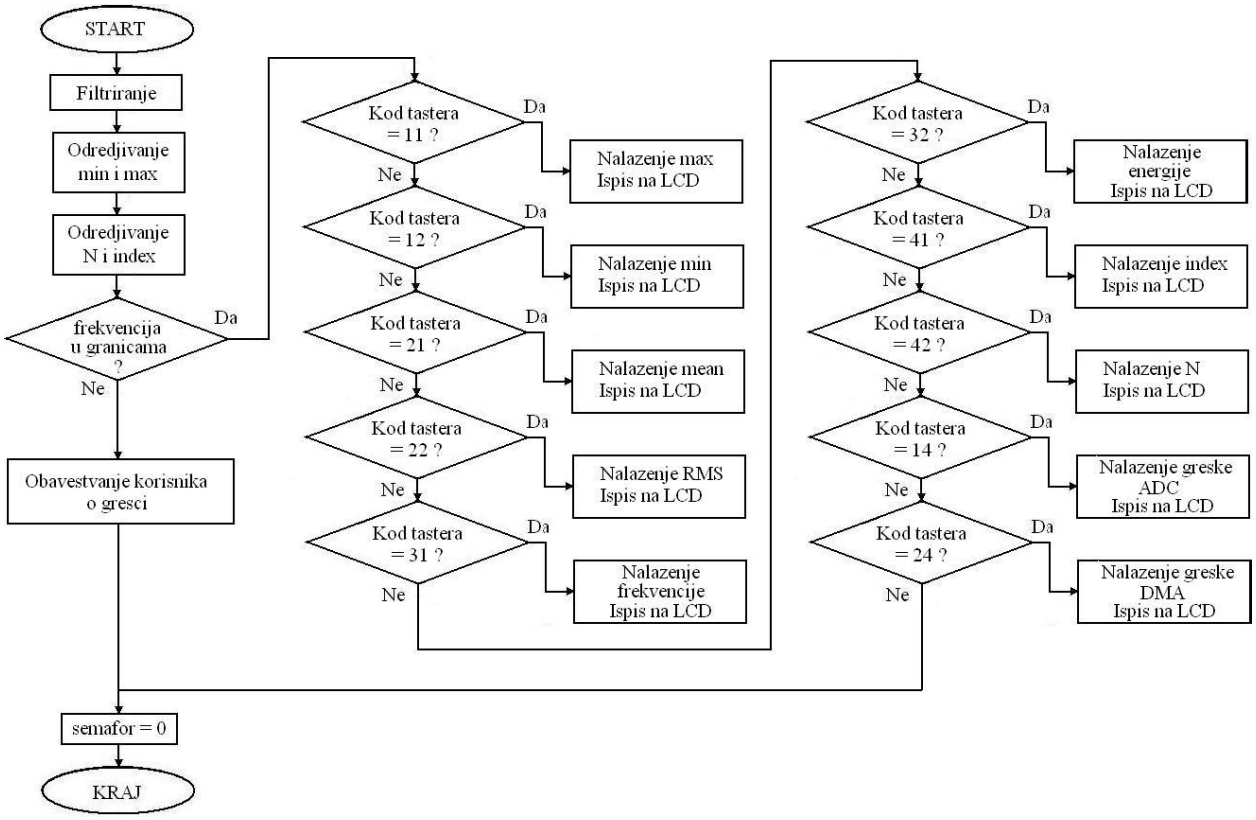
Kao što je više puta pomenuto, obrada podataka se vrši u stanju semafor=2, kreativno nazvanom Obrada. Na početku izvršavanja ovog dela koda najpre se kompletiraju neke opšte akcije kako bi se sistem pripremio za samu odradu. Pod ovim se podrazumeva pre svega filtriranje podataka. U aplikaciji je implementiran jednostavan averaging filter koji se poziva funkcijom *Filtriranje*. On je dizajniran da tekuću vrednost odbirka zameni sa srednjom vrednošću tog odbirka i njemu četiri susedna odbirka, dva se jedne i dva se druge strane. Na ovaj način se značajno umanjuju spike-ovi koji se javljaju usled šuma i drugih smetnji, a koji proizvode greške u vrednostima signala. Nakon filtriranja analizator pouzdanije radi tj. rezultati do kojih se dolazi su tačniji. Aplikacija potom nalazi minimalnu i maksimalna vrednost signala kao i parametar granica= (min+max)/2. Ovaj parametar je izuzetno važan jer se preko njega određuje broj odbiraka N koji staje u jednu periodu ulaznog signala. Naime, u cirkulari bafer

može ukupno da stane 128 odbiraka ulaznog periodičnog signala, ali se ne zna koja je njegova frekvencija. Da bi se to detektovalo napisan je jednostavan kod koji određuje broj semplova u periodi, N. Ovaj kod funkcioniše tako što detektuje dva susedna sempla: jedan koji se nalazi neposredno iznad granice i jedan koji se nalazi ispod granice. Pošto je ulazni signal periodičan, to znači da moraju da postoje minimim još dva susedna sempla koja ispunjavaju isti uslov. Prebrojavanjem semplova između dva prelaska granice određujemo broj semplova (N) u periodi ulaznog signala. Naravno, N ne može biti bilo koji broj. Minimalnu vrednost broja N određuje tačnost rezultata merenja. Ako je broj semplova mali, npr. 3, 5, 10 itd. nalaženje minimima, maksimima, srednje vrednosti, RMS itd. je neprecizno. Da bi rezultati bili iole pouzdani neophodno je makar 16 semplova, pa je to minimalna vrednost za N. Maksimalna vrednost za broj semplova u periodi je određena kodom koji pronalazi N. Da bi on sigurno funkcionisao potrebno je da među 128 odbiraka ulaznog signala postoje minimum dva periode tog signala, što N ograničava na 64. Sada kada je pronđena vrednost broja N, vrši se provera da li on leži u opsegu [16, 64]. Pošto je učestanost odabiranja 300KHz, onda sledi da za $N \in [16, 64]$. frekvencija ulaznog signala $\in$ [4687.5 Hz, 18750 Hz]. Stoga je moguće posmatrati i analizirati signale čija je učestanost samo u ovom opsegu, što je već pomenuto u poglavlju 3. Dakle pre bilo kakve obrade potrebno je utvrditi da li se N nalazi u željenim granicama. Ukoliko ovo nije slučaj korisniku se šalje poruka da frekvencija signala nije odgovarajuća i program prelazi u stanje semafor=1, Očitavanje Tastature. Ako je učestanost u dozvoljenom opsegu, vrši se obrada nad podacima u skladu sa kodom pritisnutog tastera prema tabeli 1:

KOD TASTERA	FUNKCIJA
11	Maksimum
12	Minimum
21	Mean
22	RMS
31	Frekvencija
32	Energija
41	Index
42	N
14	Brojač greške ADC
24	Brojač greške DMA

Tabela 1. Kod taster i njegova funkcija

Kada je poznat broj sempla (index) od koga perioda signala počinje i kad se zna koliko iznosi N, nije teško izračunati parametre navedene u tabeli. Rezultati odrade se potom ispisuju na LCD-u i ostaju na njemu sve dok korisnik ne zatraži neku drugu funkcionalnost. Za to vreme, promenljiva semafor je postavljena na 0, pa aplikacija nastavlja sa očitavanjem tastature. Treba napomenuti da su poslednje četiri funkcionalnosti, navedene u tabeli 1 korišćene prilikom pisanja aplikacije, da bi se videlo ponašenje pojedinih delova koda i kako bi se otklonili bagovi te nisu značajni za korisnika. Kompletan tok izvršavanja ovog stanja je prikazan na slici 13.



Slika 13. Dijagram toka za stanje Obrada

6. Rezultati

Za testiranje ove aplikacije neophodno je koristiti signal generator. Zbog ove specifičnosti aplikacija je isprobana u laboratoriji 18 u realnim uslovima. Problema pri startovanju i predviđenom radu aplikacije je bilo, što je i u neku ruku i očekivano, pa su debagovanje i različite korekcije, u cilju upaređenja, vršene na licu mesta. Na kraju su svi problemi uspešno otklonjeni i program se dobro pokazao.

U cilju prikaza rada aplikacije napravljena su dva video snimka (dostavljena uz izveštaj) koja prikazuju funkcionisanje analizatora signala.

U prvom slučaju na analizator je doveden sinusoidalni signal sledećih parametara: max = 1.97V; min = 0.71V ; mean = 1,34V; RMS = 1.4121V; E = 1.99 J i f =6925Hz. Ovo je signal čije su karakteristike nisu nameštene na celobrojne i lepe vrednosti. Kao što se može videti na snimku nazvanom Video 1, analizator daje vrlo dobre rezultate. Najveća greška se javlja prilikom određivanja minimuma i iznosi oko 3 % dok je odstupanje ostalih parametara između 1 i 2%. Na snimku se vidi i još jedna interesantna činjenica, naime frekvenciji od 6925Hz odgovara broj odbiraka u periodi ulaznog signala od N≈43.3. Aplikacija u većini slučajeva ovu vrednost zaokriži na 43 mada se neki put desi da zaokruži i na 44. Zbog toga se dobija učestanost od 6918 Hz pa je učinjena greška 1.58%.

Najveća boljka ovog analizatora jesu mali signali tj. malih vrednosti i amplituda čija se frekvencija približava gornjoj dozvoljenoj granici od 18.75 KHz. U video snimku 2, prikazan je worst case. Na analizator je doveden testerasti signal sledećih parametara: max = 250mV; min = 230mV ; mean =240mV; RMS = 0.24V; E = 0.057 J, f =17.341KHz. Greške se sada uvećavaju i se oko 4-5%, ali ipak aplikacija dobro prati ulazni signal. Veliko odstupanje međutim postoji prilikom nalaženja f. Kako je N≈17.3 što predstavlja mali broj semplova, funkcija nekada izračuna da je N manje od 16 pa izbacuje grešku u kojoj navodi da učestanost signala nije odgovarajuća. Za pouzdan rad, očigledno da je potrebo uzeti više odbiraka, odnosno povećati minimalnu granicu za N.

7. Zaključak

Projekat predstavljen u ovom dokumentu prikazuje jedno jednostavnije rešenje analizatora signala. U radu su definisani osnovni zahtevi u pogledu hardvera i softvera, opisani su primenjeni protokoli i objašnjen je način rada kompletnog sistema. Pored toga vodilo se računa i o organizaciji i čitljivosti celokupnog programa.

Funkcionisanje analizatora u realnim uslovima dalo je zadovoljavajuće rezultate. Prilikom realizacije pojavljivali su se problemi, ali su oni uspešno rešeni. Autor veruje i da je napravio dobar koncept rada aplikacije tako da je poboljšanja i eventulane nadogradnje moguće jednostavno implementirati. Zbog svega pomenutog autor smatra da je ispunio zadati projektni zadatak.

Što se tiče unapređenja sistema, mesta za to svakako postoji. Da bi se greška u rezultatima merenja još više smanjila potrebno je implementirati kompleksniji filtar od onog koji je upotrebljen u ovom radu. Njegova karakteristika ne bi bila jednostavna jer on mora da propusti DC komponentu signala kao i AC komponentu na željenom opsegu. To podrazumeva da će red filtra biti jako veliki, pa se može pojaviti problem sa nedostatkom memorije. Još jedno poboljšanje postojećeg rešenja se odnosi na veličinu cirkularnog bafera. Trebalo bi pronaći način da se ovaj bafer poveća pa bi se time povećala i tačnost merenja i analizator bi se učinio širokopojasnijim. Međutim, najveće poboljšanje leži u analizi signala u frekvencijskom domenu, kao što se i radi kod komercijalnih uređaja. Tada bi bilo moguće sračunati parametre vezane za spektar signala pa čak ga i prikazati na nekom malo boljem displeju.

8. Literatura

[1] User Manual - STM32VLDISCOVERY, STMicroelectronics, 2010.
[2] Reference Manual - STM32F100xx advanced ARM-based 32-bit MCUs, STMicroelectronics, 2010.
[3] STM32F100xx Datasheet
[4] The definitive guide to the ARM Cortex-M3, Second edition, Joseph Yiu, 2010.
[5] User Manual - Developing your STM32VLDISCOVERY application using the IAR Embedded Workbench software, STMicroelectronics, 2010.
[6] Keypad 4x4 User Manual, MikroElektronika
[7] COG2x16 LCD Board, MikroElektronika
[8] Display Unit User's Manual, Dot-Matrix LCD Units, Sharp
[9] 3.3V-5V Regulator**,** MikroElektronika
[10] http://www.radio-electronics.com/info/t_and_m/logic_analyzer/logic_analyzer.php
[11] http://en.wikipedia.org/wiki/Signal_analyzer
[12] http://en.wikipedia.org/wiki/Spectrum_analyzer
[13] http://www.aeroflex.com/ats/products/product/Spectrum_Analyzers_and_Signal_Analyzers/S-Series/SVA_Signal_Analyzer~801.html
[14] http://www.home.agilent.com/en/pc-1707666/x-series-signal-analyzers?&cc=RS&lc=eng
[15] http://sine.ni.com/nips/cds/view/p/lang/sr/nid/203041

Prilog

A.1 portovi.c

```
#include "portovi.h"
GPIO_InitTypeDef GPIO_InitStructure;
void Inicijalizacija_Portova(void)
{
   unsigned char i;
   // Konfiguracija pinova za LED diode //
   GPIO_StructInit(&GPIO_InitStructure);
   GPIO_InitStructure.GPIO_Pin   = LED_GREEN | LED_BLUE;
   GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_Out_PP;
   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
   GPIO_Init(GPIOC, &GPIO_InitStructure);
   GPIO_ResetBits(Port_LED, LED_GREEN | LED_BLUE);

   // Konfiguracija PC.2 za ADC //
   GPIO_StructInit(&GPIO_InitStructure);
   GPIO_InitStructure.GPIO_Pin = ADC_IN;
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
   GPIO_Init(Port_ADC, &GPIO_InitStructure);

   // Konfiguracija pinova za LCD //
   GPIO_StructInit(&GPIO_InitStructure);
   GPIO_InitStructure.GPIO_Pin = (RS_LCD | E_LCD);
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
   GPIO_Init(Port_LCD, &GPIO_InitStructure);
   GPIO_WriteBit(Port_LCD, RS_LCD | E_LCD, Bit_RESET);

   for (i=0;i<4;i++)
    {
     GPIO_StructInit(&GPIO_InitStructure);
     GPIO_InitStructure.GPIO_Pin = Pin_LCD[i]; //D4,D5,D6,D7
     GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
     GPIO_Init(Port_LCD,&GPIO_InitStructure);
     GPIO_WriteBit(Port_LCD, Pin_LCD[i], Bit_RESET);
    }

   // Konfiguracija pinova za tastaturu //
   for (i=0;i<4;i++)
    {
     GPIO_StructInit(&GPIO_InitStructure);
     GPIO_InitStructure.GPIO_Pin = Pin_Tast_In[i]; //P4,P5,P6,P7
     GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
     GPIO_Init(Port_Tast_In,&GPIO_InitStructure);
     GPIO_WriteBit(Port_Tast_In, Pin_Tast_In[i], Bit_RESET);
    }
```

```
   for (i=0;i<4;i++)
     {
      GPIO_StructInit(&GPIO_InitStructure);
      GPIO_InitStructure.GPIO_Pin = Pin_Tast_Out[i]; //P0,P1,P2,P3
      GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
      GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPD;
      GPIO_Init(Port_Tast_Out,&GPIO_InitStructure );
      GPIO_WriteBit(Port_Tast_Out, Pin_Tast_Out[i], Bit_RESET);
     }
}
```

A.2 rcc_m.c

```
#include "rcc_m.h"
#include "stm32f10x_gpio.h"
#include "Tajmer.h"
 void CLK_Periferija(void)
 {
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_GPIOB | RCC_APB2Periph_GPIOC,
                                                                                    ENABLE); // Portovi A,B,C //
    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE); // DMA //
    RCC_ADCCLKConfig(Division); // Ucestanost ADC-a 6 MHz//
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE); // ADC //
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM, ENABLE);  // TIM //
 }
```

A.3 nvic_m.c

```
#include "nvic_m.h"
 NVIC_InitTypeDef NVIC_InitStruct;
 void NVIC_Konfiguracija(void)
 {
    NVIC_InitStruct.NVIC_IRQChannel = DMA1_Channel1_IRQn;
    NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStruct);

    NVIC_InitStruct.NVIC_IRQChannel = ADC1_IRQn;
    NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStruct.NVIC_IRQChannelSubPriority = 1;
    NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStruct);

    NVIC_InitStruct.NVIC_IRQChannel = TIM_IRQn;
    NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 1;
    NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStruct);
}
```

A.4 adc_m.c

```
#include "adc_m.h"
ADC_InitTypeDef ADC_InitStructure;
```

```
void ADC_Inicijalizacija(void)
 {
    ADC_InitTypeDef ADC_InitStructure; // Deklaracija strukture //

    ADC_DeInit(ADC1); // Inicijalizacija ADC-a default vrednostima //
    ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
    ADC_InitStructure.ADC_ScanConvMode = ENABLE; // Mora jer DMA jedino radi sa SCAN modom //
    ADC_InitStructure.ADC_ContinuousConvMode = ENABLE; // Continous Mode
    ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
    ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
    ADC_InitStructure.ADC_NbrOfChannel = 1;
    ADC_Init(ADC1, &ADC_InitStructure); // Inicijalizacija ADC-a novim vrednostima //

    ADC_RegularChannelConfig(ADC1, ADC_Channel, 1 , Sample_Time);

    ADC_DMACmd(ADC1, ENABLE); // Dozvola za DMA prenos //

    // Konfiguracija Watchdog-a //
    ADC_AnalogWatchdogThresholdsConfig(ADC1, ADC_max, ADC_min);
    ADC_AnalogWatchdogSingleChannelConfig(ADC1, ADC_Channel);
    ADC_AnalogWatchdogCmd(ADC1, ADC_AnalogWatchdog_SingleRegEnable);

    ADC_ITConfig(ADC1, ADC_IT_AWD, ENABLE);

    ADC_Cmd(ADC1, ENABLE); // Enable-ovan ADC //

    // Kalibracija ADC-a //
    ADC_ResetCalibration(ADC1);
    while(ADC_GetResetCalibrationStatus(ADC1));
    ADC_StartCalibration(ADC1);
    while(ADC_GetCalibrationStatus(ADC1));

    // Softversko startovanje ADC-a //
    ADC_SoftwareStartConvCmd(ADC1, ENABLE);
 }
```

A.5 dma_m.c

```
#include "dma_m.h"
#include "adc_m.h"
DMA_InitTypeDef DMA_InitStructure; // Deklaracija strukture //
extern __IO uint16_t cirk_bafer[N_odbiraka];
  void DMA_Inicijalizacija(void)
 {
    DMA_DeInit(DMA1_Channel1); // Inicijalizacija kanal 1 default vrednostima //
    DMA_InitStructure.DMA_PeripheralBaseAddr = ADC1_DR_Address; // Izvorisna adresa za DMA prenos //
    DMA_InitStructure.DMA_MemoryBaseAddr = (u32)&cirk_bafer; // Odredisna adresa za DMA prenos //
    DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
    DMA_InitStructure.DMA_BufferSize = N_odbiraka;
    DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
    DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
    DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord;
    DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
    DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;
    DMA_InitStructure.DMA_Priority = DMA_Priority_High;
```

```
    DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;
    DMA_Init(DMA1_Channel1, &DMA_InitStructure);
    //DMA_ITConfig(DMA1_Channel1, DMA_IT_HT | DMA_IT_TC,ENABLE); // Dozvola HalfTransfer i TransferComplite
prekida
    DMA_ITConfig(DMA1_Channel1, DMA1_IT_TE1, ENABLE); // Dozvola Transfer Error prekida
    DMA_Cmd(DMA1_Channel1, ENABLE); // Dozvoljen DMA prenos //
 }
```

A.6 tajmer.c

```
#include "tajmer.h"
TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
extern char state;
void Cekaj(int n)
{
   TIM_TimeBaseStructure.TIM_Period = ((n*100)-1); // n*100 microsec
   TIM_TimeBaseStructure.TIM_Prescaler = (24-1);  // 1 microsec
   TIM_TimeBaseInit(TIM, &TIM_TimeBaseStructure);
   TIM_ITConfig(TIM, TIM_IT_Update, ENABLE);
   TIM_Cmd(TIM, ENABLE);
   state = 1;
   while(state)
   {
   }
   TIM_Cmd(TIM, DISABLE);
}
```

A.7 tastatura.c

```
#include "stm32f10x.h"
#include "tastatura.h"
#include "portovi.h"
// Funkcija za ocitavanje tastature //
unsigned char Ocitaj_tastaturu(void)
{
   unsigned char  ocitan_pin[] = {0,0,0,0};
   unsigned char  kontrola=0;
   unsigned char  kolona,vrsta,pritisnut_taster=0;

   for (vrsta=0; vrsta <=3; vrsta++)
   {
     GPIO_WriteBit(Port_Tast_In, Pin_Tast_In[vrsta], Bit_SET);
     for (kolona=0; kolona <=3; kolona++)
       {
        ocitan_pin[kolona]=GPIO_ReadInputDataBit(Port_Tast_Out,Pin_Tast_Out[kolona]);
        if (ocitan_pin[kolona]==1)
         {
           kontrola+=1;
           pritisnut_taster=(Kod_Tastera[vrsta][kolona]);
         }
       }
     GPIO_WriteBit(Port_Tast_In, Pin_Tast_In[vrsta], Bit_RESET);
   }
   if (kontrola==1) // Zastita protiv istovremenog pritiska vise tastera //
     {
```

```
      return(pritisnut_taster);
     }
   else return 0;
}
```

A.8 lcd.c

```
#include "lcd.h"
#include "string.h"
#include "stdio.h"
// Slanje informacija (podatak ili instrukcija) LCD-u //
void LCD_Informacija(unsigned char info)
{
   unsigned char maska = 0x80, informacija;
   int i;

   for (i = 0; i<=3; i++)
   {
     informacija = info & maska;
     if (informacija == 0x00)
         GPIO_WriteBit(Port_LCD, Pin_LCD[i], Bit_RESET);
     else GPIO_WriteBit(Port_LCD, Pin_LCD[i], Bit_SET);

     maska >>= 1;
   }
}

// Inicijalizacija LCD-a //
void LCD_Inicijalizacija(void)
{
   Cekaj(310);
   LCD_Slanje_Komande(Function_set_U);
   Cekaj(45);
   LCD_Slanje_Komande(Function_set_U);
   Cekaj(3);
   LCD_Slanje_Komande(Function_set_U);
   Cekaj(2);
   LCD_Slanje_Komande(Function_set_D);
   LCD_Slanje_Komande(Function_set_D);
   LCD_Slanje_Komande(Function_set_D2);
   Cekaj(2);
   LCD_Slanje_Komande(Display_OFF_U);
   LCD_Slanje_Komande(Display_OFF_D);
   Cekaj(2);
   LCD_Slanje_Komande(Clear_Display_U);
   LCD_Slanje_Komande(Clear_Display_D);
   Cekaj(25);
   LCD_Slanje_Komande(Entry_Mode_Set_U);
   LCD_Slanje_Komande(Entry_Mode_Set_D);
   Cekaj(3);
   LCD_Slanje_Komande(Display_ON_U);
   LCD_Slanje_Komande(Display_ON_D);
   Cekaj(3);
   LCD_Slanje_Komande(Display_Home_U);
   LCD_Slanje_Komande(Display_Home_D);
```

```
}

// Slanje komande ili adrese LCD-u //
void LCD_Slanje_Komande(unsigned char komanda)
{
   Cekaj(3);
   GPIO_WriteBit(Port_LCD, E_LCD, Bit_RESET);
   GPIO_WriteBit(Port_LCD, RS_LCD, Bit_RESET);
   Cekaj(1);
   GPIO_WriteBit(Port_LCD, E_LCD, Bit_SET);
   Cekaj(1);
   LCD_Informacija(komanda);
   Cekaj(1);
   GPIO_WriteBit(Port_LCD, E_LCD, Bit_RESET);
   Cekaj(1);
   LCD_Informacija(Clear);
   Cekaj(3);
}

// Slanje podatka LCD-u //
void LCD_Slanje_Podatka(unsigned char podatak)
{
   Cekaj(3);
   GPIO_WriteBit(Port_LCD, E_LCD, Bit_RESET);
   GPIO_WriteBit(Port_LCD, RS_LCD, Bit_RESET);
   Cekaj(1);
   GPIO_WriteBit(Port_LCD, RS_LCD, Bit_SET);
   Cekaj(1);
   GPIO_WriteBit(Port_LCD, E_LCD, Bit_SET);
   Cekaj(1);
   LCD_Informacija(podatak);
   Cekaj(1);
   GPIO_WriteBit(Port_LCD, E_LCD, Bit_RESET);
   Cekaj(1);
   GPIO_WriteBit(Port_LCD, RS_LCD, Bit_RESET);
   Cekaj(1);
   LCD_Informacija(Clear);
   Cekaj(3);
}

// Slanje teksta LCD-u //
void LCD_Slanje_Teksta(unsigned char red, char tekst[])
{
   unsigned char i,pozicija=0x00,pomoc;
   char *pomocni_tekst=tekst;
   int duzina=strlen(tekst);

   if (duzina<=16)
   {
     for(i=0;i<duzina;i++)
     {
       LCD_Slanje_Komande(red);
       LCD_Slanje_Komande(pozicija);
       LCD_Slanje_Podatka(pomocni_tekst[i]);
```

```
        pomoc=pomocni_tekst[i]<<4;
        LCD_Slanje_Podatka(pomoc);
        pozicija+=16;
      }
    }
}

// Slanje realnog broja LCD-u //
void LCD_Slanje_Broja(unsigned char red, float broj)
{
   char realni_broj[12];

   sprintf(realni_broj,"%5.5f",broj);
   LCD_Slanje_Teksta(red,realni_broj);
}

// Brisanje sadrzaja ispisanog na displeju //
void LCD_Brisanje(void)
{
  char praznine[]={"               "};
  LCD_Slanje_Teksta(Prvi_red,praznine);
  LCD_Slanje_Teksta(Drugi_red,praznine);
}
```

A.9 stm32f10x_it.c

```
/******************************************************************************/
/*               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 TIM7_IRQHandler(void)
{
   state = 0;
   TIM_ClearFlag(TIM, TIM_FLAG_Update);
}

void DMA1_Channel1_IRQHandler(void)
{
   if(DMA_GetITStatus(DMA1_IT_TE1)==SET)
   {
     brojac_greske_DMA++;
     DMA_ClearITPendingBit(DMA1_IT_TE1);
   }
}

void ADC1_IRQHandler(void)
```

```
{
   if(ADC_GetITStatus(ADC1,ADC_IT_AWD)==SET)
   {
    brojac_greske_ADC++;
    ADC_ClearITPendingBit(ADC1, ADC_IT_AWD);
   }
}
```

A.10 main.c

```
/*********************************************************************************************
                          Analizator signala
                        Mladen Cicmil 2012/3210
*********************************************************************************************/

#include "stm32f10x.h"
#include "STM32vldiscovery.h"
#include "portovi.h"
#include "tajmer.h"
#include "tastatura.h"
#include "lcd.h"
#include "rcc_m.h"
#include "nvic_m.h"
#include "adc_m.h"
#include "dma_m.h"
#include "math.h"
#include "stdlib.h"

void Greska(char greska, char broj, char poruka);
void Filtriranje(float *pok1, float *pok2);

__IO uint16_t cirk_bafer[N_odbiraka];
float bafer_1[N_odbiraka],*pokb_1=&bafer_1[0], bafer_2[N_odbiraka], *pokb_2=&bafer_2[0];
int N_bafer=N_odbiraka;
unsigned char i, semafor=1, pocetak=1;
unsigned char prethodni_taster=0,taster=0;
char brojac_greske_ADC=0,brojac_greske_DMA=0;

char tekst[13][16]={ "Ready...",                    // 0
                     "Min[V] =",    "Max[V] =",     // 1,2
                     "Mean[V] =",   "RMS[V] =",     // 3,4
                     "f[KHz] =",    "Energija[J] =", // 5,6
                     "N =",         "Index =",      // 7,8
                     "GRESKA !!",   "f van opsega",  // 9,10
                     "ADC watchdog", "DMA transfer"   // 11,12
                    };
float fs=300000, faktor_skaliranja=(2.99 / 4096);

int main(void)
{
 // Dovodjenje CLK-a signala periferijama //
   CLK_Periferija();

 // Inicijalizavija portova za ADC, tastaturu, displej //
   Inicijalizacija_Portova();
```

```
// Konfiguracija prekida //
  NVIC_Konfiguracija();

// Inicijalizacija LCD-a //
  LCD_Inicijalizacija();

// Inicijalizacija DMA kontrolera //
  DMA_Inicijalizacija();

// Inicijalizacija i startovanje AD konvertora //
  ADC_Inicijalizacija();

// Signalizira da je inicijalizacija zavrsena //
  STM32vldiscovery_LEDToggle(LED3); // GREEN //

  while (1)
 {
   switch (semafor)
     {
       case 0:  // Ocitavanje tastature //
        {
          taster=Ocitaj_tastaturu();

          if ((taster != 0) && (taster != prethodni_taster))
              {semafor = 2;}
          else {semafor = 1;}

          prethodni_taster = taster;

         break;
        }// od case-a nula

       case 1:  // Kopiranje bafera //
        {
          char ukupna_greska_DMA = brojac_greske_DMA, ukupna_greska_ADC = brojac_greske_ADC;
          char greska_DMA_1 = brojac_greske_DMA, greska_ADC_1 = brojac_greske_ADC;

          Greska(ukupna_greska_ADC, 30, 11);
          Greska(ukupna_greska_DMA, 50, 12);

          DMA_ClearFlag((DMA1_FLAG_HT1) | (DMA1_FLAG_TC1));

          while(DMA_GetFlagStatus(DMA1_FLAG_HT1)==RESET);

          DMA_ClearFlag(DMA1_FLAG_HT1);
          for (i=0; i<=((N_bafer/2)-1); i++)
            {
             bafer_1[i]=(cirk_bafer[i])*faktor_skaliranja;
            }

          while(DMA_GetFlagStatus(DMA1_FLAG_TC1)==RESET);

          DMA_ClearFlag(DMA1_FLAG_TC1);
```

```
  for (i=(N_bafer/2); i<(N_bafer); i++)
   {
    bafer_1[i]=(cirk_bafer[i])*faktor_skaliranja;
   }

  char greska_ADC_2 = brojac_greske_ADC, greska_DMA_2 = brojac_greske_DMA;

  Greska((greska_ADC_2-greska_ADC_1), 5, 11);
  Greska((greska_DMA_2-greska_DMA_1), 5, 12);


  STM32vldiscovery_LEDToggle(LED4); // BLUE, sve je proslo bez gresaka //

   if (pocetak==1)
   {
    LCD_Brisanje();
    LCD_Slanje_Teksta(Prvi_red,tekst[0]);   // Ready, sad je program spreman da primi komandu sa tastature //
    pocetak=0;
   }
  semafor = 0;
 break;
}// od case-a jedan

case 2:   // Obrada //
 {
   unsigned char brojac=0;
   int N=0,index=0;
   float min,max,granica,f,mean,RMS,Energija,suma=0,suma_kvadrata=0;

   min = bafer_2[0];
   max = bafer_2[0];

   Filtriranje(pokb_2,pokb_1);
   //Filtriranje(pokb_1,pokb_2);

   for (i=1; i<N_bafer; i++)
    {
     if (bafer_2[i] < min)
        min=bafer_2[i];
     if (bafer_2[i] > max)
        max=bafer_2[i];
    }

   granica=(min+max)/2;

   for (i=0; i<N_bafer; i++)
    {
      if ((bafer_2[i+1] < granica) && (bafer_2[i] >= granica))
        brojac++;
      if (brojac==1)
        {  N++;
          if (index==0)
            index=i;
        }
```

```
   }

  if ((N > 15) && (N < N_bafer/2)) // Frekvencija signala je odgovarajuca
   {
      switch (taster)
        {
          case 11: // taster 1 - MAX
           {
             LCD_Brisanje();
             LCD_Slanje_Teksta(Prvi_red,tekst[2]);
             LCD_Slanje_Broja(Drugi_red,max);
           break;
           }
          case 12:  // taster 2 - MIN
           {
            LCD_Brisanje();
            LCD_Slanje_Teksta(Prvi_red,tekst[1]);
            LCD_Slanje_Broja(Drugi_red,min);
           break;
           }
//-------------------------------------------------------------//
          case 21:  // taster 4 - MEAN
           {
            for (i=index; i<=(index+N-1); i++)
              {
               suma+=bafer_2[i];
              }
            mean=suma/N;

            LCD_Brisanje();
            LCD_Slanje_Teksta(Prvi_red,tekst[3]);
            LCD_Slanje_Broja(Drugi_red,mean);

           break;
           }
          case 22:  // taster 5 - RMS
           {
            for(i=index; i<=(index+N-1); i++)
              {
               suma_kvadrata+=(bafer_2[i]*bafer_2[i]);
              }
            RMS=sqrt(suma_kvadrata/N);

            LCD_Brisanje();
            LCD_Slanje_Teksta(Prvi_red,tekst[4]);
            LCD_Slanje_Broja(Drugi_red,RMS);
           break;
           }
//-------------------------------------------------------------//
          case 31: // taster 7 - f
           {
            f=(fs/N)/1000; //Da bi se dobili KHz
            LCD_Brisanje();
            LCD_Slanje_Teksta(Prvi_red,tekst[5]);
```

```
              LCD_Slanje_Broja(Drugi_red,f);
             break;
             }
            case 32:  // taster 8 - Energija
             {
              for(i=index; i<=(index+N-1); i++)
                {
                 suma_kvadrata+=(bafer_2[i]*bafer_2[i]);
                }
              Energija=suma_kvadrata/N;

              LCD_Brisanje();
              LCD_Slanje_Teksta(Prvi_red,tekst[6]);
              LCD_Slanje_Broja(Drugi_red,Energija);
             break;
             }
//-------------------------------------------------------------//
            case 41:  // taster * - index
             {
              LCD_Brisanje();
              LCD_Slanje_Teksta(Prvi_red,tekst[8]);
              LCD_Slanje_Broja(Drugi_red,index);
             break;
             }
            case 42:  // taster 0 - N( broj odbiraka u periodi)
             {
              LCD_Brisanje();
              LCD_Slanje_Teksta(Prvi_red,tekst[7]);
              LCD_Slanje_Broja(Drugi_red,N);
             break;
             }
//-------------------------------------------------------------//
           case 14:  // taster A - greska ADC
             {
              float greskaADC;
              greskaADC = brojac_greske_ADC *1.00;
              LCD_Brisanje();
              LCD_Slanje_Teksta(Prvi_red,tekst[11]);
              LCD_Slanje_Broja(Drugi_red,greskaADC);
             break;
             }
            case 24:  // taster B - greska DMA
             {
              float greskaDMA;
              greskaDMA = brojac_greske_DMA *1.00;
              LCD_Brisanje();
              LCD_Slanje_Teksta(Prvi_red,tekst[12]);
              LCD_Slanje_Broja(Drugi_red,greskaDMA);
             break;
             }
          } //od switch-a unutar case 2
    } // od If-a

  else    // Frekvencija signala nije odgovarajuca
```

```
            {
              LCD_Brisanje();
              LCD_Slanje_Teksta(Prvi_red,tekst[9]);
              LCD_Slanje_Teksta(Drugi_red,tekst[10]);
            }
           semafor = 0;
          break;
         }//od case-a 2
     }// od switch-a
  }//od while-a
}//od main-a*/

/*******************************************************************************************
                              END OF MAIN
*******************************************************************************************/
#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
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)
  {
  }
}
#endif

// Funkcija odradu gresaka ADC-a i DMA-a //
  void Greska(char greska, char broj, char poruka)
  {
     if (greska > broj)
       {
         LCD_Brisanje();
         LCD_Slanje_Teksta(Prvi_red,tekst[9]);
         LCD_Slanje_Teksta(Drugi_red,tekst[poruka]);
         //exit(1); // Prekid programa
       }
  }

// Funkcija za filtriranje //
  void Filtriranje(float *pok1, float *pok2)
  {
     pok1[0]=(pok2[0] + pok2[1]) / 2;
     pok1[1]=(2 * pok1[0] + pok2[2]) / 3;
     pok1[2]=(3 * pok1[1] + pok2[3] + pok2[4]) / 5;
        for (i=3; i<(N_bafer-2); i++)
```

```
        {
         pok1[i]=(5 * pok1[i-1] - pok2[i-3] + pok2[i+2]) / 5;
         }
    pok1[N_bafer-2]=(pok2[N_bafer-3]+pok2[N_bafer-2]+pok2[N_bafer-1])/3;
    pok1[N_bafer-1]=(pok2[N_bafer-1]+pok2[N_bafer-2])/2;
}
```