Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Marko Turšič

Krmiljenje visokonapetostnega generatorja

DIPLOMSKO DELO NA UNIVERZITETNEM ŠTUDIJU

Mentor: izr. prof. dr. Uroš Lotrič

Ljubljana, 2013

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01893/2013
Datum: 07.01.2013

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **MARKO TURŠIČ**

Naslov: **KRMILJENJE VISOKONAPETOSTNEGA GENERATORJA**
CONTROLLING HIGH VOLTAGE GENERATOR

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Visokonapetostni generator se uporablja za test prebojne napetosti na izdelkih za splošno uporabo. Za obstoječi visokonapetostni generator predlagajte nov mikroprocesorski sistem in ustrezno programsko opremo, s katero bo uporabnik preko osebnega računalnika na intuitiven način vnašal potek napetosti skozi čas in izvajal poskuse.

Mentor:
izr. prof. dr.Uroš Lotrič

Dekan:
prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

I Z J A V A O A V T O R S T V U

diplomskega dela

Spodaj podpisani/-a Marko Turšič,

z vpisno številko 63050121,

sem avtor/-ica diplomskega dela z naslovom:

Krmiljenje visokonapetostnega generatorja

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom (naziv, ime in priimek)

  izr. prof. dr. Uroš Lotrič

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela

- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne 29.5.2013 Podpis avtorja/-ice:

Zahvala

Hvala družini za podporo tekom študija.

Mentorju izr. prof. dr. Urošu Lotriču gre zahvala za vse napotke, nasvete in pomoč pri diplomskem delu.

Hvala podjetju Dat-Con d.o.o., ki mi je omogočilo izdelavo diplomskega dela, še posebno gospodoma Domnu Plaskanu in Juretu Zdovcu.

Nenazadnje gre zahvala tudi vsem ostalim, ki so kakorkoli pripomogli k nastanku diplomskega dela.

Kazalo

Povzetek

Visokonapetostni generator se uporablja v industrijskih procesih za test prebojne napetosti na izdelkih za splošno uporabo. Cilj diplomske naloge je bil napisati programsko opremo za obstoječi visokonapetostni generator. Z njo uporabnik na osebnem računalniku, preko grafičnega vmesnika aplikacije, na intuitiven način vnaša potek napetosti skozi čas in izvaja poskuse.

Programska oprema vsebuje računalniško aplikacijo z grafičnim vmesnikom, napisano v programskem jeziku C# in program za mikrokrmilnik STM32f103RB v visokonapetostnem generatorju, napisan v programskem jeziku C. Za povezavo med njima smo implementirali protokol, ki deluje v obliki pošiljanja in sprejemanja prenosnih paketov preko serijskega vmesnika.

Uporabniku programska oprema omogoča vnos napetostne funkcije direktno na izhod ali v notranji pomnilnik visokonapetostnega generatorja. Če je izhod postavljen, se uporabniku na grafičnem vmesniku sproti izrisuje dejanski potek napetosti izhoda skozi čas. Tako uporabnik vidi razliko med željeno izhodno funkcijo in dejansko.

Mikrokrmilnik je glavna elektronska komponenta visokonapetostnega generatorja, ki skrbi za celotno funkcionalnost. Mikrokrmilniški program preko serijskega vmesnika sprejema prenosne pakete in jih ustrezno obdela. Njegove glavne naloge so postavljanje izhoda, zapis napetostne funkcije v notranji pomnilnik in sporočanje dejanske vrednosti izhoda aplikaciji.

Na koncu smo izvedli test programske opreme s katerim smo preverili pravilnost delovanja in dosegli zastavljene cilje diplomske naloge.

Ključne besede:

visokonapetostni generator, mikrokrmilnik STM32f103RB, programski jezik, serijski vmesnik, grafični vmesnik

Abstract

High-voltage generator is used in industrial processes for testing breakthrough voltage on products for general use. The aim of the thesis was to write software for controlling the high-voltage generator. With this software user can input the voltage over time and carries out experiments via a graphical user interface on the PC in an intuitive way.

The software includes computer application with a graphical user interface (GUI) written in the programming language C # and a program for the microcontroller STM32f103RB in the high-voltage generator, written in C programming language. For the connection between the two, we implemented a protocol that works in the form of sending and receiving transmission packages via a serial interface.

The software allows user to input desired voltage function directly on the output or in the internal memory of the high voltage generator. If the output is set, the actual course of the voltage output over time is simultaneously drawn on the GUI. Hence the user can see the difference between the desired and actual output function.

The microcontroller is the main electronic component of the high-voltage generator, which is responsible for the overall functionality. Microcontroller program accepts transmission packages through the serial port, and handels them appropriately. Its main tasks are setting output, saving voltage function in the internal memory and notifying the actual output values to the application.

At the end we carried out a software test with which we verified the correctness of the operations and achieved the goals of the thesis.

Key words:

high-voltage generator, microcontroller STM32f103RB, programming language, serial interface, graphical user interface (GUI)

1. Uvod

Visokonapetostni generator se danes uporablja v številnih industrijskih procesih. Z njim izvajamo teste prebojne napetosti bele tehnike, zabavne elektronike in drugih elektronskih naprav. Maksimalna izhodna vrednost napetosti visokonapetostnega generatorja je 2000 V, kar zagotavlja teste za širok spekter naprav. V primeru napačnega delovanja, napačne napetostne vrednosti na izhodu, lahko pride do uničenja bremena priključenega na njegov izhod. Uničeno breme in iskanje vzroka napačnega delovanja pa predstavlja denarni strošek in časovno potratnost, kar je za podjetje zelo slabo. Pravilno delovanje je zato zelo pomembno. Visokonapetostni generator smo podrobneje razložili v poglavju 1.

Pred izdelavo naloge je bila v notranjem spominu visokonapetostnega generatorja zapisana statična funkcija poteka napetosti, ki se je ob sprožitvi pojavila na izhodu. Za izvršitev druge napetostne funkcije na izhodu, je bilo potrebno vsakič znova sprogramirati oziroma vnesti drugo napetostno funkcijo v notranji pomnilnik visokonapetostnega generatorja. To opravilo je bilo zamudno, zlasti pri pogostem spreminjanju izhodne napetostne funkcije. Pojavila se je ideja o računalniškem programu, ki bi omogočal hitreje, predvsem pa preprosteje spremeniti izhodno napetostno funkcijo visokonapetostnega generatorja.

Naša naloga je bila napisati računalniško aplikacijo z grafičnim vmesnikom, preko katerega krmilimo visokonapetostni generator. Njen glavni namen je izris željene napetostne funkcije, prenos funkcije na visokonapetostni generator in izris grafa trenutne napetosti na izhodu. Opis, delovanje, funkcionalnost in implementacijo aplikacije smo opisali v poglavju 5. Glavni del programske kode aplikacije je v prilogi, dodatek B.

Aplikacija preko serijskega vmesnika komunicira z mikrokrmilnikom v generatorju, ki podatke, zapisane v obliki prenosnega paketa, sprejema in ustrezno obdela. Uporabljeni mikrokrmilnik in njegovo delovanje smo predstavili v poglavju 4. Programska koda glavne zanke se nahaja v prilogi, dodatek A.

Komunikacijo med aplikacijo in mikrokrmilnikom smo realizirali s protokolom za izmenjavo podatkov. Izmenjava podatkov poteka v obliki prenosnih paketov. Delovanje serijske komunikacije, sestavo prenosnega paketa in ciklični redundančni kod za zagotavljanje pravilnosti prenosa paketa smo pojasnili v poglavju 2.

Potek izdelave diplomske naloge in nekatere težave pri pisanju programa za krmiljenje visokonapetostnega generatorja smo opisali v 6. poglavju. Navedene so tudi morebitne izboljšave programa.

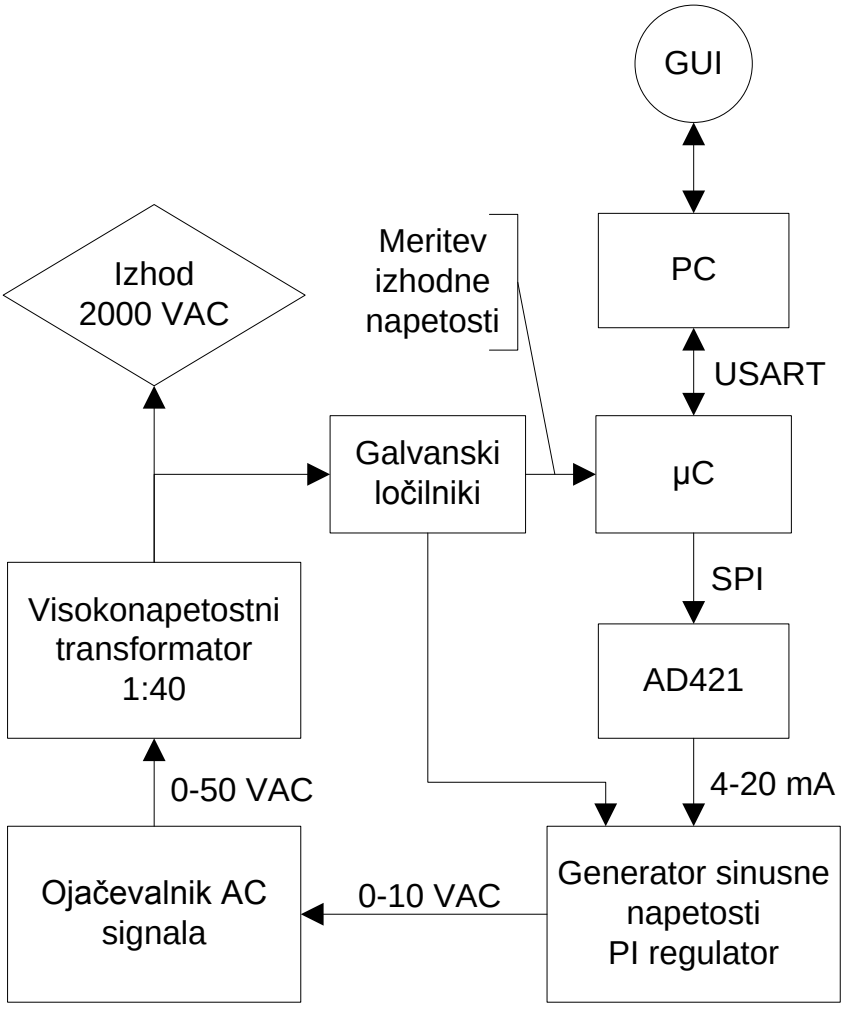
2. Visokonapetostni generator

Generator visoke napetosti [1] (slika 1) generira visoko napetost do 2000 V za namen testiranja prebojnih napetosti v industrijskih procesih. Uporabnik preko nadzornega programa nastavi potek izhodne napetosti visokonapetostnega generatorja. Generator sinusne napetosti, ki je sestavni del naprave, generira sinusno napetost. To prejme ojačevalni del naprave, ki poleg prvostopenjskega ojačenja napetosti poskrbi tudi za tokovno zmogljivost izhoda naprave. Končna stopnja, katere glavna komponenta je visokonapetostni transformator, ojača napetost na končno izhodno vrednost.

Slika 1 - visoko napetostni generator

Glavne komponente:

- nadzorni mikrokrmilniški modul,
- generator analognega tokovnega izhoda AD421 [2] v območju 4-20 mA,
- generator sinusne napetosti v območju 0-10 V izmenične napetosti,
- proporcionalno-integrirni regulator izhodne napetosti,
- ojačevalnik izmeničnega signala v območju 0-50 V izmenične napetosti,
- visoko napetostni transformator in
- galvanski ločilniki.



Slika 2 - shema vezja

2.1. Povratna informacija izhoda

Izhodno napetost merimo na nizkonapetostni strani galvanskih ločilnikov in jo zajemamo preko 12-bitnega AD pretvornika. Mikrokrmilnik najprej preveri stanje izhoda in nato omogoči prenos vrednosti na izhod. Diagnostika sistema se izvede takoj ob vklopu generatorja in traja približno pet sekund.

S povratno zanko preverjamo dva osnovna pogoja za delovanje sistema. Prvi pogoj je sklenjen tokovni krog. Drugi pogoj preverja moč bremena, priključenega na visoko napetostni generator. Z zanko preverjamo tudi splošno delovanje sistema in diagnostiko sistema, s katero ugotavljamo stanje izhodne napetosti, odzivni čas generatorja in stabilnost izhoda.

Napetost na izhodu merimo preko galvanskih ločilnikov. Služi kot povratna informacija, ki jo potrebujemo za grafični prikaz dejanske vrednosti napetosti na izhodu visokonapetostnega generatorja in za proporcionalno-integrirni regulator. Preden se na vhodu sinusnega generatorja pojavi nova vrednost, nov tok, mora biti regulacija zaključena.

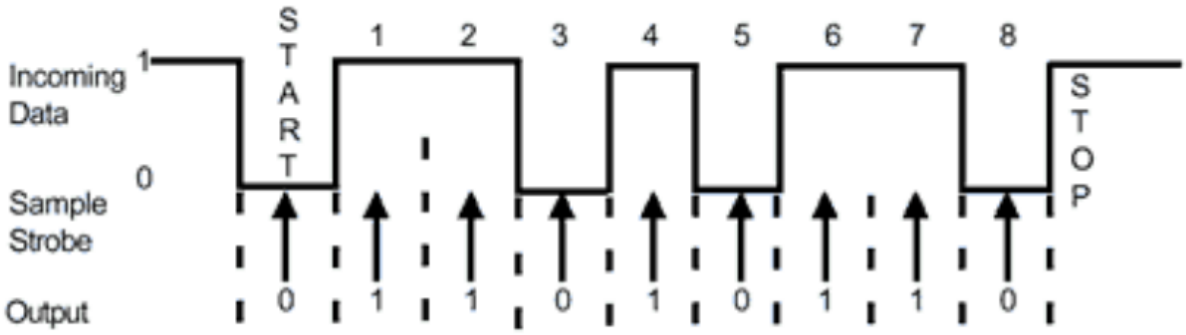
2.2. Način delovanja

Uporabnik zažene aplikacijo za krmiljenje visokonapetostnega generatorja na računalniku. Na risalni panel grafičnega vmesnika izriše željeno napetostno funkcijo in jo prek serijskega vmesnika pošlje na mikrokrmilnik. Mikrokrmilnik preveri osnovna pogoja za delovanje visokonapetostnega generatorja, sklenjen tokovni krog in priključeno breme na izhodu generatorja. Če sta pogoja izpolnjena, se vrednost preko serijskega perifernega komunikacijskega vodila (ang. Serial Peripheral Interface Bus, SPI) prenese na generator analognega tokovnega izhoda AD421. Generator analognega tokovnega izhoda AD421 generira izhod, ki predstavlja skalirano vrednost napetosti, 4-20 mA. Tokovni izhod je hkrati vhod v generator sinusne napetosti. Izhodna vrednost sinusnega generatorja je predstavljena z amplitudami od 0-10 V izmenične napetosti. V ojačevalniku izmeničnega signala se ta napetost še dodatno ojača za faktor pet. V končni stopnji visokonapetostni transformator dvigne napetost na izhodno vrednost, 0-2000 V izmenične napetosti.

3. Komunikacijski protokol

3.1. Serijska komunikacija

Serijska komunikacija je proces pošiljanja podatkov preko komunikacijskega kanala ali računalniškega vodila. Podatki se pošiljajo zaporedno, vsak bit posebej (slika 3). Nasprotno se pri paralelni komunikaciji pošilja več bitov hkrati. Serijska komunikacija se uporablja v večini računalniških omrežij, saj je zaradi cene in sinhronizacije dosti primernejša od paralelne.



Slika 3 - zaporedno pošiljanje bitov pri serijski komunikaciji

Periferna enota mikrokrmilnika, ki skrbi za serijsko komunikacijo, se imenuje univerzalni sinhroni-asinhroni sprejemnik-oddajnik (ang. Universal synchronous/asynchronous receiver/transmitter, USART). Mikrokrmilnik je preko paralelnega komunikacijskega vodila povezan z enoto USART. Podatek se na izhodu enote USART generira z napetostnimi nivoji od nič do napajalne napetosti mikrokrmilnika. Logično ničlo predstavlja območje od 0 do 0,4 V, logično enico pa napajalna napetost z do 0,4 V odstopanja.

Eden bolj razširjenih industrijskih standardov za serijsko komunikacijo je RS232. Podpira ga velika večina mikrokrmilnikov. V najbolj okrnjeni različici uporablja dve signalni liniji in maso. Prva signalna linija je namenjena branju podatka, druga pošiljanju. Dodatne signalne linije omogočajo usklajevanje komunikacije. Sočasno podpira dvosmerni prenos in detekcijo napak.

Včasih se je standard RS232 veliko uporabljal v osebnih računalnikih, vendar ga je danes zaradi nizke prenosne hitrosti, velikega nihanja napetosti med logičnimi nivoji in velikih povezovalnih priključkov nadomestil standard USB.

3.2. Prenosni paket

Aplikacija na računalniku komunicira z mikrokrmilnikom v visokonapetostnem generatorju preko serijskega vmesnika. Podatki se iz računalnika pošiljajo na mikrokrmilnik v obliki prenosnega paketa (slika 4). Vsebino prenosnega paketa lahko na grobo razdelimo na dva dela, podatkovni in prenosni del. Podatkovni del vsebuje polje bajtov, v katerem je zapisana napetostna funkcija. Prenosni del paketa tvorijo posamezni bajti pripeti na začetek in konec podatkovnega dela, ki kot celota skrbijo za pravilen prenos paketa.

Začetek paketa 0x02	Način delovanja 0x55/0x66	Napetostna funkcija 0x00 - 0xFF	Ciklična koda 0x00 – 0xFF	Ciklična koda 0x00 – 0xFF	Konec paketa 0x03

Slika 4 - struktura prenosnega paketa

Začetek paketa je enolično določen z vrednostjo, ki se ne more pojaviti znotraj paketa. To smo zagotovili z maskiranjem podatkov, opisanem v naslednjem poglavju. Po začetku paketa sledi način delovanja, ki mikrokrmilniku pove ali gre za programski ali kontrolni način. Zatem sledi podatkovni del paketa, v katerem je zapisana napetostna funkcija. Prvi podatek v tem polju je vzorčna frekvenca, s katero smo vzorčili napetostno funkcijo. Potem si zaporedno sledijo posamični vzorci, ki jih predstavimo z vrednostjo napetosti. Zatem pride 16-bitna ciklična koda, ki omogoča detekcijo napak pri prenosu paketa na mikrokrmilnik. Ker je prenosni paket sestavljen iz polja bajtov, smo jo morali razdeliti na dva dela. Na koncu imamo prav tako kot na začetku enolično določen konec paketa.

3.3. Maskiranje podatkov

Maskiranje podatkov je nujno potrebno zaradi enolično določenega začetka in konca prenosnega paketa. Če maskiranja podatkov ne bi bilo, bi se lahko znotraj paketa pojavili vrednosti, določeni za začetek ali konec (slika 5). Prišlo bi do napake, saj bi mikrokrmilnik zaznal predčasen konec paketa ali začetek novega paketa še pred koncem prejšnjega. Začetek in konec paketa smo določili z bajtoma 0x02 oziroma 0x03, zapisanima v šestnajstiški obliki. Ti dve vrednosti se tako nikoli ne smeta pojaviti znotraj paketa, saj bi to privedlo do nepravilnega delovanja.

Tako smo celoten podatkovni del paketa, maskirali in s tem zagotovili pravilno strukturo (slika 6).

Do maskiranja podatka pride, ko se v podatkovnem delu paketa pojavi ena od sledečih vrednosti:

- začetek paketa 0x02,
- konec paketa 0x03,
- maskirni bajt 0x7D.

Nad vsako od teh vrednosti znotraj paketa izvedemo operacijo XOR z bajtom 0x20. Tako dobimo nove vrednosti:

- 0x02 XOR 0x20 = 0x22,
- 0x03 XOR 0x20 = 0x23,
- 0x7D XOR 0x20 = 0x5D.

Pred vsako od tako dobljenih vrednosti nato vstavimo maskirni bajt 0x7D. Maskirni bajt je potrebno vstaviti zato, da mikrokrmilnik ve, da sledi maskirni bajt, ki ga nato z operacijo XOR z vrednostjo 0x20 pretvori nazaj v pravo vrednost.

Primer maskiranja prenosnega paketa:

0x02	0x55	0x32	0x02	0xA5	0x7D	0x03	0xFF	0x03

Slika 5 - prenosni paket pred maskiranjem

0x02	0x55	0x32	0x7D	0x22	0xA5	0x7D	0x5D	0x7D	0x23	0xFF	0x03

Slika 6 - prenosni paket po maskiranju

Kot vidimo na zgornji sliki se znotraj paketa pojavljajo vrednosti, ki jih je potrebno maskirati. Na spodnji sliki je prikazan isti paket z maskiranimi vrednostmi.

3.4. Ciklični redundančni kod

Ciklični redundančni kod (ang. Cyclic Redundancy Check, CRC) je namenjen odkrivanju napak pri prenosu po komunikacijskem kanalu. Pogosto se uporablja v digitalnih omrežjih in napravah za shranjevanje podatkov za detekcijo naključnih napak, do katerih lahko pride pri prenosu. Vsakemu podatkovnemu paketu, ki vstopa v komunikacijski kanal se dodajo varnostni biti. Te se izračuna z uporabo polinomskega deljenja nad podatki vsakega paketa posebej pred vstopom v komunikacijski kanal. Na sprejemni strani se na sprejetem paketu ponovno izračunajo. Če se vrednosti ujemajo, je prenos paketa pravilen. V nasprotnem primeru, ko so si različni, je potrebno prenos ponoviti ali ukrepati kako drugače.

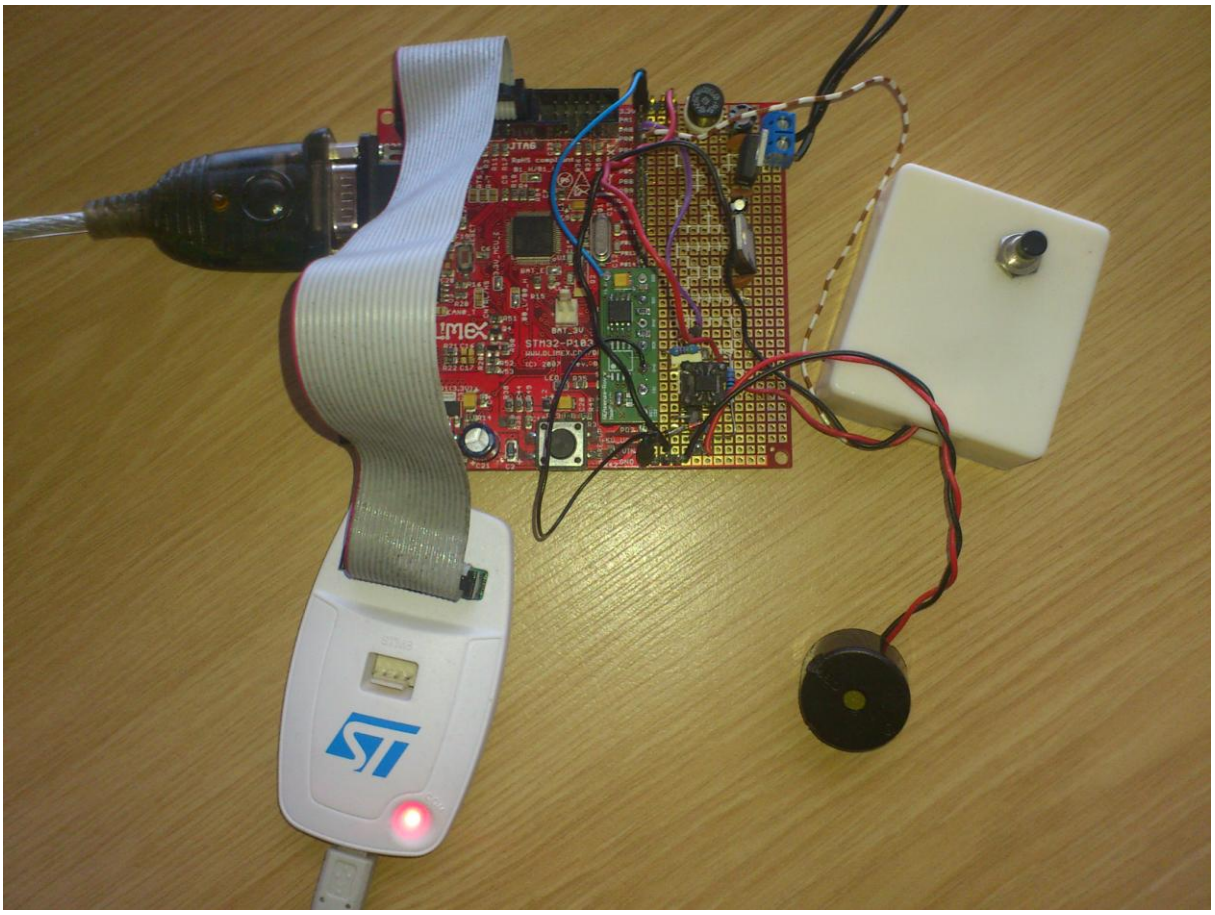
V našem primeru smo uporabili 16-bitni ciklični kod, ki ga podaja standard CRC16-CCITT [3]. Aplikacija pred pošiljanjem vsakega prenosnega paketa izračuna 16 varnostnih bitov, ki se vstavijo neposredno za podatkovni del paketa. Mikrokrmilnik sprejeti paket obdela, če pri prenosu ni prišlo do napake. V nasprotnem primeru paket zavrže.

Pri prenosu pomembnih podatkov preko komunikacijskega kanala je to zelo pomembno. V našem primeru bi na primer zaradi šuma ali kateregakoli drugega dejavnika pri prenosu mikrokrmilnik prejel napačne vrednosti. To bi lahko privedlo do napačnih, posledično katastrofalnih posledic na izhodu visokonapetostnega generatorja. Zato je uporaba cikličnega redundančnega koda pri prenosu podatkov na visokonapetostni generator nujna.

4. Programska rešitev na mikrokrmilniku

Razvijanje programa za visokonapetostni generator smo začeli na razvojni ploščici Olimex [4] (slika 7), ki je vsebovala mikrokrmilnik STM32f103RB. Uporabljali smo razvojno orodje µVision [7] proizvajalca Kiel in programator ST-Link. Celotna koda je napisana v programskem jeziku C.

Simuliranje izhoda visokonapetostnega generatorja smo za potrebe razvoja simulirali s piezo-električnim piskačem. Spreminjanje napetosti na izhodu visokonapetostnega generatorja se je odražalo kot jakost zvoka piskača. Vse druge periferne enote visokonapetostnega generatorja, kot so tipke in komunikacijski moduli, so že bili del razvojne ploščice.



Slika 7 - razvojna ploščica Olimex s programatorjem ST Link

Po uspešnem razvoju in testiranju programa na razvojni ploščici, smo obstoječi mikrokrmilnik Philips LPC2294 v visokonapetostnem generatorju zamenjali z mikrokrmilnikom STM32f103RB, na katerem je potekal celoten razvoj.

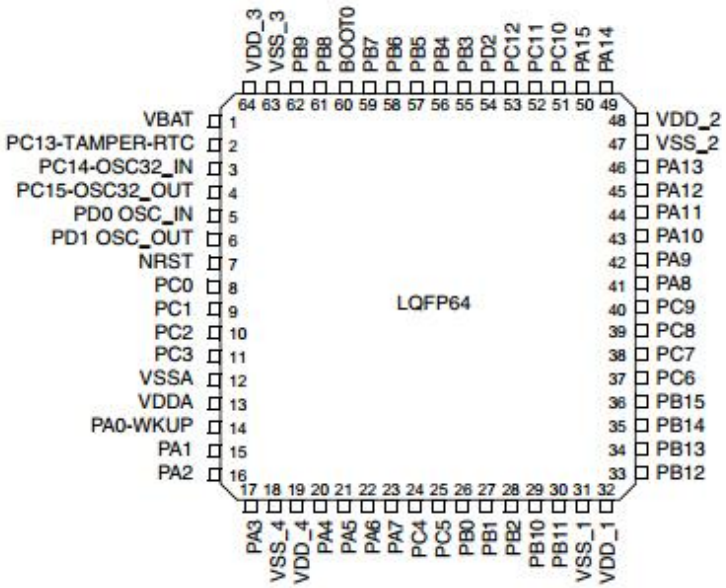
4.1. Mikrokrmilnik STM32f103RB

Najpomembnejša strojna komponenta visokonapetostnega generatorja je mikrokrmilnik STM32f103RB [5] podjetja ST Microelectronics. Pripada srednjemu velikostnemu razredu družine mikrokrmilnikov STM32f103xx, ki imajo 64-nožno podnožje (slika 8). Glavni del mikrokrmilnika je 32-bitno jedro ARM Cortex-M3. Za širok spekter uporabe skrbi veliko število vhodno-izhodnih enot in hiter vgrajeni pomnilnik.

Njegove glavne naloge so sprejemanje podatkov iz računalnika, obdelava prejetih podatkov, postavljanje izhodnih vrednosti visokonapetostnega generatorja in sporočanje povratne informacije nazaj na računalnik.

Mikrokrmilnik ima:

- 32-bitno ARM Cortex-M3 jedro,
- maksimalno frekvenco procesorja 72MHz,
- 128kB pomnilnika Flash in 20kB pomnilnika SRAM,
- dva 12-bitna analogno-digitalna pretvornika s po 16-kanali,
- 7-kanalni krmilnik DMA,
- 37 vhodno-izhodnih vrat, preko katerih se lahko proži do 16 zunanjih prekinitev,
- 2 x 16-bitna časovnika,
- komunikacijske module: 2 x USART, 1 x SPI, 1 x I2C, USB ter CAN,
- modul za izračun ciklično redundančnega koda in
- strojno podprt razhroščevalni način izvajanja.

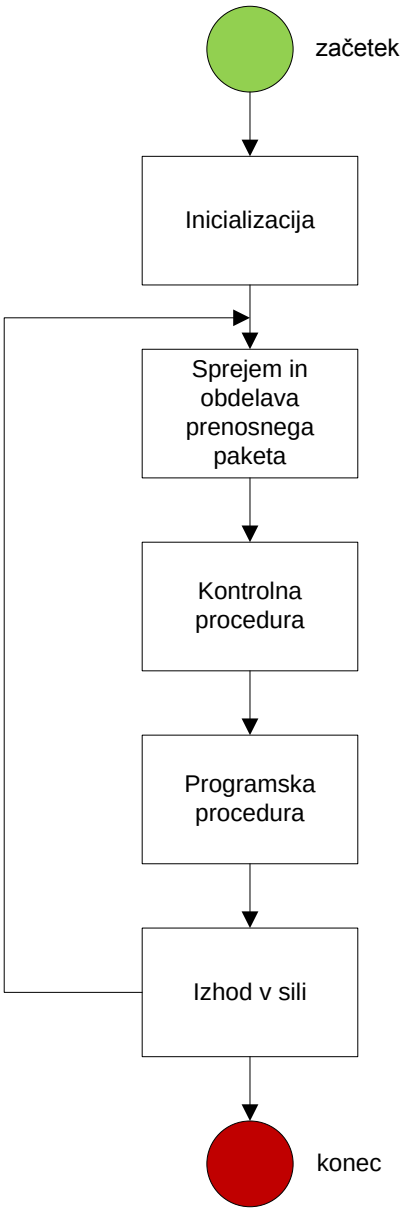


Slika 8 - podnožje mikrokrmilnika STM32f103RB

4.2. Opis delovanja mikrokrmilnika

Program na mikrokrmilniku visokonapetostnega generatorja je v večji meri ostal enak kot na razvojni ploščici. Dodali smo metodo za branje dejanske vrednosti izhoda *vng_o( )* in popravili metodo za postavljanje napetostne funkcije na izhod *hvg_write( )*.

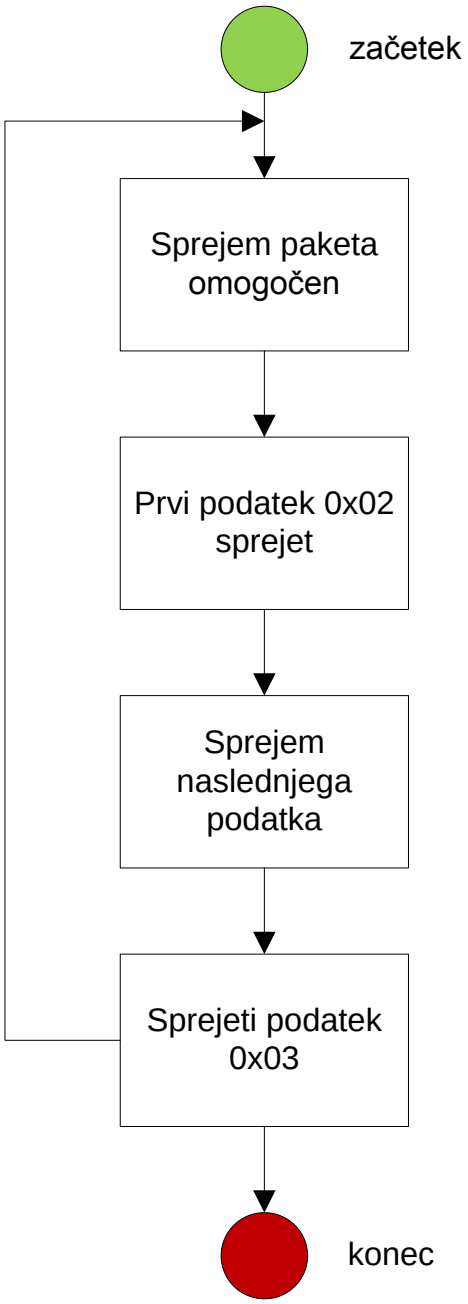
Po zagonu visokonapetostnega generatorja mikrokrmilnik najprej izvede inicializacijo, kar traja nekaj sekund. Nato se program izvaja v neskončni zanki, ki je sestavljena iz štirih modulov (slika 9). Vsak modul se izvede le v primeru izpolnjenega pogoja, kot je sprejem novega paketa ali pritisk na tipko. Po izvedbi modula se pogoj za njegovo izvajanje postavi nazaj v prvotno vrednost in izvajanje zanke se nadaljuje.



Slika 9 - Potek programa na mikrokrmilniku

4.2.1. Sprejem paketa

Ob sprejemu novega podatka preko serijskega vmesnika pride do prekinitve na mikrokrmilniku. V prekinitvenem vektorju se izvede metoda za sprejem podatkov (slika 10). Če so podatki oziroma prenosni paket ustrezne strukture, se ga shrani v tabelo prejetih podatkov. V nasprotnem primeru se podatke zavrže.



Slika 10 - Struktura metode za sprejem podatkov

Metoda za sprejem prenosnega paketa se izvede za vsak prejeti bajt posebej. Znotraj metode naprej preverimo, če je sprejetje novega podatka omogočeno. Nato preverimo, če prvi prejeti podatek ustreza začetku paketa, bajt 0x02. Hkrati v istem pogoju preverimo, da nismo sprejeli še nobenega podatka. Če sta oba pogoja izpolnjena, prvi podatek shranimo v tabelo prejetih podatkov, števec prejetih podatkov povečamo in zabeležimo prejem prvega podatka. Naslednje podatke v vrsti za sprejem vpisujemo v tabelo prejetih podatkov in povečujemo števec, dokler ne pridemo do konca paketa oziroma bajta 0x03, ki označuje konec paketa. Pri tem števec prejetih podatkov in prejem prvega podatka resetiramo. Postavimo tudi pogoj, da so podatki pripravljeni za obdelavo. To je eden od dveh pogojev za izvedbo prvega modula glavne zanke programa, ki podatke ustrezno obdela. Ob naslednji prekinitvi se postopek sprejema prenosnega paketa ponovi.

4.2.2. Obdelava paketa

Modul za obdelavo prenosnega paketa je prvi od štirih modulov v glavni zanki programa. Za njegovo izvedbo morata biti izpolnjena naslednja dva pogoja. Modul kontrolne procedure ne sme biti v izvajanju in podakti morajo biti pripravljeni.

Na začetku modula izvedemo demaskiranje podatkov in izračuno ciklični redundančni kod (CRC) nad prejetimi podatki. Iz tabele prejetih podatkov preberemo CRC, ki smo ga izračunali na računalniku in vstavili v prenosni paket. Če se koda ujemata, je bil sprejem podatkov pravilen in nadaljujemo z izvajanjem modula. V nasprotnem primeru se izvajanje modula za obdelavo podatkov prekine in program se izvaja naprej. Pravilnost prenosnega paketa je zelo pomembna, saj bi lahko ob napačnih podatkih na izhodu generatorja prišlo do nepravih vrednosti, kar pa je v večini primerov nedopustno.

Nadaljujemo z branjem drugega bajta v prenosnem paketu, ki določa enega od dveh načinov delovanja generatorja. Če je prebrana vrednost 0x66, imamo programski način delovanja. V statični pomnilnik generatorja zapišemo število vzorcev napetostne funkcije in napetostno funkcijo. Na računalnik pošljemo obvestilo, da je bila napetostna funkcija uspešno zapisana v pomnilnik generatorja. V drugem primeru je vrednost drugega bajta 0x55, kar označuje kontrolni način delovanja. Pri tem načinu napetostno funkcijo postavimo direktno na izhod generatorja. Iz začetka paketa preberemo frekvenco vzorčenja napetostne funkcije in shranimo. Na računalnik pošljemo obvestilo, ki izklopi gumbe na zaslonski maski aplikacije, zaradi pričetka kontrolne procedure. Zaženemo časovnik in postavimo pogoj, da je kontrolna procedura v teku. Nastavimo števec za branje napetostnih vzorcev na prvi vzorec. Modul za obdelavo paketa se tako ne more izvajati, dokler se kontrolna procedura ne konča.

4.2.3. Kontrolna procedura

Modul kontrolne procedure se izvede, če so izpolnjeni naslednji trije pogoji. Pogoj kontrolne procedure iz prvega modula za obdelavo paketa mora biti izpolnjen. Vrednost časovnika, sproženega v prvem modulu, je večja ali enaka frekvenci vzorčenja. Števec števila vzorcev napetostne funkcije je manjši od števila vseh vzorcev napetostne funkcije.

Vrednost vzorca preberemo in jo postavimo na izhod generatorja ob vstopu v modul. Časovnik postavimo na nič, tako da prične šteti od začetka. Dejansko vrednost napetosti na izhodu generatorja, ki se nam izriše na mreži grafičnega vmesnika, pošljemo na računalnik. Števec vzorcev ustrezno povečamo in preverimo, če smo že prišli do zadnjega vzorca. Če nismo, je kontrolna procedura še vedno v teku in modul se ponovno izvede, ko časovnik doseže vrednost frekvence vzorčenja. Pogoj za izvajanje kontrolne procedure pobrišemo, ko na izhod postavimo zadnjo vrednost vzorca. Ukaz, da se gumbi na grafičnem vmesniku ponovno vklopijo, pošljemo na računalnik. S tem je modul kontrolne procedure zaključen.

4.2.4. Programska procedura

Naloga modula programske procedure je postaviti napetostno funkcijo na izhod generatorja, ki je shranjena v njegovem notranjem pomnilniku. Modul se izvede ob izpolnjenih naslednjih dveh pogojih: kontrolna procedura ni v teku, pritisnili smo zeleni gumb *start* na sprednji strani generatorja (slika 1).

Napetostno funkcijo, število vzorcev napetostne funkcije in frekvenco vzorčenja iz notranjega pomnilnika generatorja preberemo pri prvem vstopu v modul. Nastavimo števec vzorcev in sprožimo časovnik programske procedure. Ukaz, ki izklopi gumbe na grafičnem vmesniku, pošljemo na računalnik. Postavimo pogoj, da smo branje iz notranjega pomnilnika že opravili, tako da se ob naslednjem vstopu v modul ta del kode ne izvede več. Ob enaki ali večji vrednosti časovnika od frekvence vzorčenja iz tabele napetostne funkcije preberemo vrednost naslednjega vzorca in jo postavimo na izhod generatorja. Dejansko vrednost na izhodu generatorja, ki se nam izriše na mreži grafičnega vmesnika, pošljemo na računalnik. Povečamo števec vzorcev in poenostavimo časovnik, tako da začne šteti od začetka. Prejšnji korak ponovimo, ko časovnik ponovno prešteje do vrednosti frekvence vzorčenja, dokler ne pridemo do konca tabele vzorcev. Na koncu ponovno omogočimo pritisk na tipko *start*, omogočimo branje iz notranjega pomnilnika generatorja in na računalnik pošljemo ukaz za vklop gumbov na grafičnem vmesniku. S tem je programska procedura končana.

4.2.5. Izhod v sili

Delovanje visokonapetostnega generatorja lahko v nujnih primerih takoj ustavimo, in sicer s pritiskom na gumb *stop* na sprednji strani generatorja (slika 1). S tem sprožimo modul za izhod v sili. V njem izhod visokonapetostnega generatorja takoj postavimo na nič in onemogočimo vse pogoje za izvajanje preostalih modulov. Obvestilo o prenehanju delovanja pošljemo na računalnik. To se nam izpiše kot pojavno sporočilo na grafičnem vmesniku. Modul izhoda v sili je edini modul, ki se lahko izvede kadarkoli, tudi če je v izvajanju kateri od preostalih modulov.

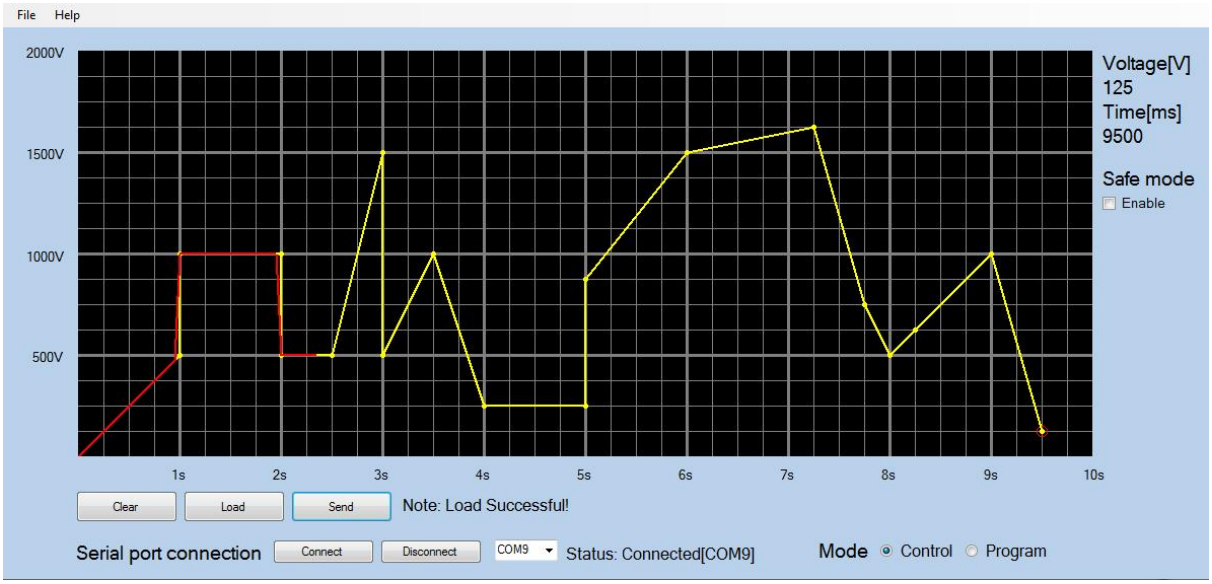
5. Aplikacija

5.1. Opis aplikacije

Aplikacija je namenjena krmiljenju visokonapetostnega generatorja. Po zagonu aplikacije se nam prikaže grafični vmesnik (slika 11), katerega glavni del je risalna mreža na katero narišemo željeno napetostno funkcijo, ki bi jo radi zagotovili na izhodu visokonapetostnega generatorja. Želeno napetostno funkcijo prenesemo na mikrokrmilnik preko komunikacijskih vrat, ki jih izberemo preko grafičnega vmesnika.

Rišemo na mrežo razpona dva tisoč voltov v vertikalni smeri in deset sekund v horizontalni smeri. Risanje funkcije poteka s klikanjem na mrežo, ki izbrano točko linearno poveže s prejšnjo. Tako dobimo množico povezanih točk oziroma graf rumene barve. Izbrati moramo med dvema načinoma delovanja. Programski način funkcijo zapiše v notranji pomnilnik visokonapetostnega generatorja, kontrolni način pa funkcijo postavi direktno na izhod visokonapetostnega generatorja. Ko visokonapetostni generator postavi določeno funkcijo na izhod, aplikacija dobi povratno informacijo o dejanski vrednosti izhoda, ki se nam izrisuje na mreži v obliki grafa rdeče barve. Tako vidimo razliko poteka napetosti izrisane funkcije od dejanske. Aplikacija komunicira z visokonapetostnim generatorjem preko serijskega vmesnika.

5.2. Grafični vmesnik in funkcionalnost

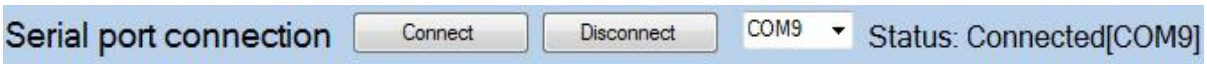


Slika 11 - grafični vmesnik

Slika 12 - risalna mreža z gumbi

Glavni gradnik grafičnega vmesnika je risalna mreža (slika 12), kjer poteka risanje napetostne funkcije. Višina mreže je 400 točk, ki predstavljajo razpon napetosti od nič do 2000 voltov. 400 točk nam tako omogoča natančnost določitve napetosti na pet voltov. Širina mreže je 1000 točk, ki predstavljajo razpon časa od nič do 10 000 milisekund, kar nam omogoča natančnost določitve časa na deset milisekund. Natančnost mreže lahko definiramo, tako da se sprehajamo le po določenih vrednostih. V zgornjem primeru je natančnost koraka 25 točk, kar predstavlja 125 voltov vertikalne in 250 milisekund horizontalne vrednosti.

Na mrežo rišemo s klikanjem na točke želenih vrednosti, ki so med seboj linearno povezane z rumeno črto. Risanje funkcije se prične s prvo točko v koordinatnem izhodišču mreže, pri napetosti in času enakima nič. Točke si morajo slediti po naraščajočem času, drugače izbira točke ni možna. V primeru izbire napačne vrednosti točke ali želje po drugačni funkciji, izbrišemo zadnjo izbrano točko in nadaljujemo od predzadnje. S klikom na gumb *Clear* pobrišemo celotno funkcijo in začnemo od začetka. Ko smo narisali želeno funkcijo, se s pritiskom na gumb *Load* funkcija z uporabo linearne interpolacije in vzorčenja zapiše v ustrezno obliko za pošiljanje na visokonapetostni generator. Funkcijo preko serijskega vmesnika pošljemo na visokonapetostni generator s pritiskom na gumb *Send*. Pred pošiljanjem se funkcijo v ustrezni obliki umesti v prenosni paket.



Slika 13 - spustni meni in gumba za vzpostavitev komunikacije

Komunikacija računalnika in visokonapetostnega generatorja poteka preko serijskega vmesnika. Povezavo vzpostavimo z izbiro ustreznih komunikacijskih vrat iz spustnega menija, na katere je priključen visokonapetostni generator in s klikom na gumb *Connect*. Poleg spustnega menija je status povezave (slika 13). Status se spreminja glede na uspešno vzpostavljeno oziroma neuspešno vzpostavljeno povezavo.

Slika 14 - izbirni meni načina delovanja

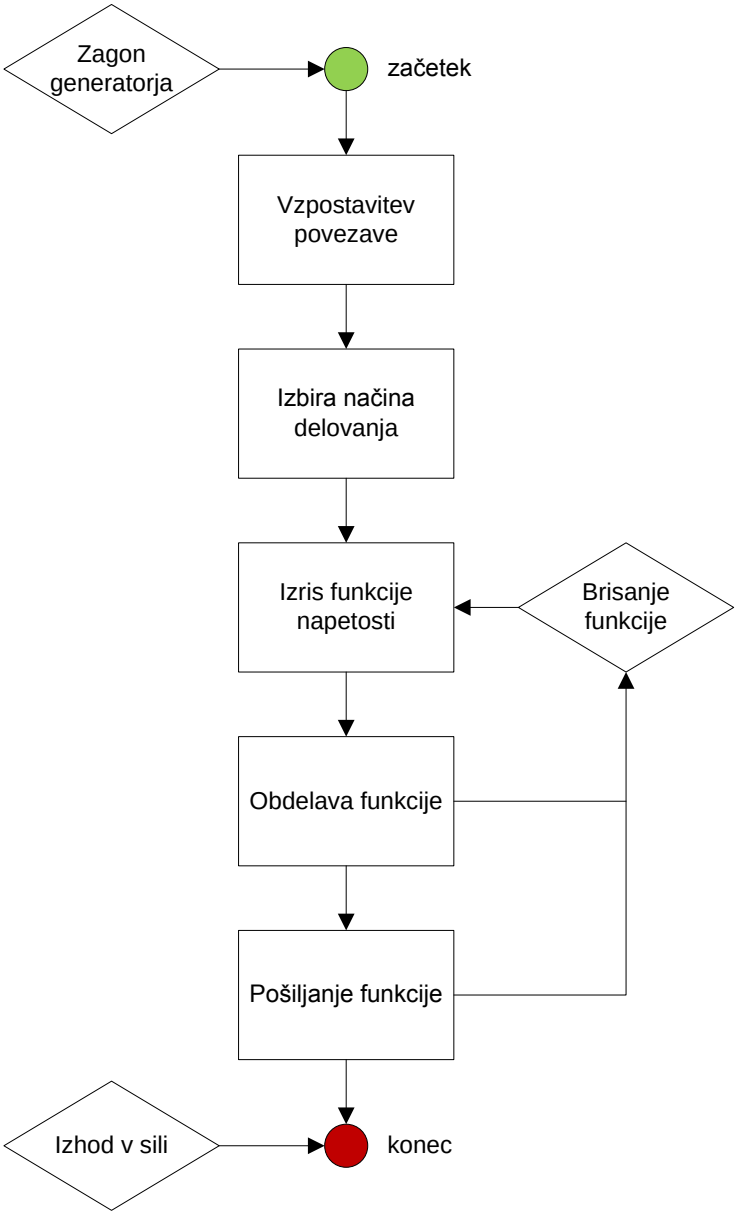
V izbirnem meniju določimo način delovanja (slika 14). Privzeto je nastavljen kontrolni način. V kontrolnem načinu se napetostna funkcija postavi na izhod visokonapetostnega generatorja, aplikacija pa dobiva povratno informacijo o dejanski vrednosti na izhodu in izrisuje dejanski graf poteka napetosti rdeče barve. Tako vidimo odstopanje dejanske funkcije od željene. Pri programskem načinu se napetostno funkcijo shrani v notranji spomin visokonapetostnega generatorja, ki se preko ustrezne tipke sproži in postavi na izhod. V tem primeru se nam na mrežo izriše le dejanski graf poteka napetosti.

Slika 15 - prikaz položaja trenutne točke in avtomatsko zaključevanje

V desnem zgornjem kotu okna imamo prikaz vrednosti napetosti in časa v trenutni točki (slika 15). Če s kazalcem zapustimo risalno mrežo, se obe vrednosti postavita na nič. Pod prikazom trenutnih vrednosti lahko izberemo varni način delovanja, ki nam omogoča avtomatsko zaključevanje funkcije po pritisku na gumb *Load*. Avtomatsko zaključevanje pomeni, da se funkcija zaključi z ničelno vrednostjo napetosti, kadar zadnja točka nima ničelne vrednosti napetosti. Če ne izberemo varnega načina, isto funkcionalnost dosežemo z desnim klikom.

5.3. Blokovna shema delovanja aplikacije

Na shemi so prikazane vse ključne funkcionalnosti aplikacije in potek delovanja. Prvi trije koraki so med seboj neodvisni, izvajajo se v poljubnem vrstnem redu. Obdelava funkcije mora slediti izrisu funkcije. V zadnjem koraku željeno funkcijo pošljemo na visokonapetostni generator in končamo, ali pa pričnemo z izrisom nove funkcije. V primeru zunanje zahteve po takojšnji prekinitvi delovanja se aplikacija lahko konča v katerem koli koraku. Zaradi neodvistnosti prvih treh korakov je možnih več različnih potekov delovanja. Prikazan je najobičajnejši (slika 16).



Slika 16 - običajen potek delovanja aplikacije

5.4. Realizacija v kodi

Aplikacijo smo napisali v programskem jeziku C# [6], v razvijalnem okolju Visual Studio 2010 [8]. V naslednjih podpoglavjih je predstavljena realizacija pomembnejših delov in funkcionalnosti aplikacije.

5.4.1. Natančnost risalne mreže

Risalna mreža je eden najpomembnejših gradnikov aplikacije, saj na njej poteka načrtovanje napetostne funkcije. Natančnost mreže določamo posebej v horizontalni in posebej v vertikalni smeri. Izbira točk je tako možna do izbrane natančnosti. V primeru izbire natančnosti petih točk, se po mreži premikamo le po točkah večkratnika pet. To smo dosegli z uporabo enačb:

$$px = \left(\frac{eX}{snapX}\right) * snapX \quad , \quad py = \left(\frac{eY}{snapY}\right) * snapY \quad , \tag{1}$$

kjer je:

- px izbrana točka v horizontalni smeri,
- py izbrana točka v vertikalni smeri,
- eX točka kazalca v horizontalni smeri,
- eY točka kazalca v vertikalni smeri,
- $snapX$ natančnost mreže v horizontalni smeri in
- $snapY$ natančnost mreže v vertikalni smeri.

5.4.2. Funkcionalnost risalne mreže

Z levim klikom miške sprožimo dogodek, ki na risalno mrežo izriše izbrano točko. To se zgodi le v primeru, če je horizontalna vrednost oziroma čas zadnje točke večji ali enak od predzadnjega. S tem onemogočimo risanje nazaj v času. Med prejšnjo in trenutno izbrano točko se izriše linearna črta, ki povezuje obe točki. Tako povezane črte tvorijo graf napetosti. Izbrano točko se shrani v polje vseh do sedaj izbranih točk.

Z desnim klikom miške sprožimo dogodek, ki funkcijo avtomatsko zaključi z vrednostjo nič. V primeru da je vrednost napetosti v zadnji točki že nič, se ne zgodi nič.

V primeru napačno izbrane točke ali željo po drugačni napetostni funkciji, jo z dvoklikom na zadnjo točko grafa odstranimo. Zadnja točka se odstrani iz polja vseh do sedaj izbranih točk. Tako nadaljujemo izris od predzadnje, ali postopek ponovimo. V želji po čisto drugačni funkciji, je hitreje s klikom na gumb *Clear* funkcijo pobrisati in z risanjem začeti od začetka.

Risalna mreža se osveži oziroma preriše vsakič, ko pride do spremembe izgleda mreže. To se zgodi ob sprožitvi dogodkov kot so klik miške, premik miške na risalni mreži, vstop miške na risalno mrežo, brisanje funkcije in tako dalje. Ker imamo v polju točk shranjene vse vrednosti do sedaj izbranih točk, jih na mrežo ponovno narišemo z metodama *DrawEllipse* in *FillEllipse*. Povezave med točkami se narišejo z uporabo metode *DrawVoltage*, ki vse izbrane točke linearno poveže z rumeno črto. Tako ob osvežitvi mreže dobimo nazaj trenutni graf napetosti.

5.4.3. Vzorčenje napetostne funkcije

Preden funkcijo pošljemo na visokonapetostni generator, jo moramo zapisati v ustrezno obliko, tabelo vzorcev. Vzorec je vrednost napetosti v določenem času. Po pritisku na gumb *Load* pokličemo metodo za vzorčenje, ki sprejme dva argumenta. Prvi je vzorčna frekvenca s katero bomo vzorčili funkcijo, drugi pa polje vseh izbranih točk, ki nam predstavljajo našo funkcijo. Na začetku metoda izračuna število vzorcev in ustvari novo tabelo dolžine enake številu vzorcev, za vpisovanje le teh. Vzorčenje funkcije se prične na intervalu med prvima dvema izbranima točkama, na katerem izračunamo smerni koeficient in odmik od izhodišča. V zanki z uporabo enačbe premice med dvema točkama $f(x) = k * x + n$, kjer je k smerni koeficient in n odmik od izhodišča, izračunamo vrednost napetosti v vmesni točki in jo dodamo v tabelo vzorcev.

Izračun vzorcev sledi na intervalu med prvo in drugo točko. Postopek ponavljamo vse do končnega intervala med predzadnjo in zadnjo točko, ko dobimo tabelo vrednosti vseh vzorcev.

5.4.4. Sestava prenosnega paketa in pošiljanje

Po izrisu in vzorčenju željene napetostne funkcije, logično sledi njeno pošiljanje na mikrokrmilnik visokonapetostnega generatorja. Pred tem jo moramo ustrezno umestiti v prenosni paket. S klikom na gumb *Send* se sproži dogodek, ki sestavi prenosni paket in opravi pošiljanje. Pred pričetkom sestavljanja prenosnega paketa preverimo vse pogoje, da se sestavljanje lahko prične. Ti pogoji so uspešno vzpostavljena povezava, izbran način delovanja in izveden postopek vzorčenja funkcije.

Prenosni paket je sestavljen iz polja bajtov. Sestavljanje pričnemo z razčlenitvijo vzorcev na dva dela, zgornjega in spodnjega. Vzorci imajo lahko vrednost do 2000 voltov, kar je

prevelika vrednost za zapis v en bajt. Tako ustvarimo novo tabelo, dvakrat daljšo od tabele vzorcev, in zapišemo zgornji in spodnji del vzorca posebej, kar predstavlja podatkovni del prenosnega paketa. Zatem pokličemo metodo *CRC_calc*, ki izračuna vrednost varnostnih bitov. Ker je rezultat 16 bitna koda, jo moramo tako kot prej vzorce razčleniti na dva dela. Nato podatkovnemu delu na začetek dodamo bajt načina delovanja, na koncu pa varnostne bite. Sledi maskiranje do sedaj sestavljenega paketa. Na začetku in koncu maskirnega paketa dodamo bajt, ki predstavlja enolično določen začetek oziroma konec paketa. S tem je sestava prenosnega paketa zaključena.

Na koncu sestavljeni prenosni paket preko serijskega vmesnika pošljemo na visokonapetostni generator.

5.4.5. Poslušanje serijskih vrat

Ob zagonu aplikacije se zažene metoda *serialPort1_DataReceived( )*, ki ves čas delovanja spremlja, kdaj se na serijskem vmesniku pojavi nov podatek. Ko se pojavi, ga preberemo in dodamo v polje do sedaj že prebranih neobdelanih podatkov. Zatem pokličemo metodo, ki prebrane podatke ustrezno obdela in po obdelavi izbriše iz polja že prebranih podatkov.

Visokonapetostni generator preko povratne vezave aplikaciji sporoča naslednje:

- dejanska vrednost napetosti na izhodu visokonapetostnega generatorja,
- pisanje v notranji pomnilnik visokonapetostnega generatorja je bilo uspešno,
- izhod v sili.

Vsako od teh sporočil visokonapetostnega generatorja je zapisano v polju štirih bajtov, kjer sta prvi in zadnji vedno enolično določena na enak način kot pri prenosnem paketu. Srednja bajta nam povesta, za katero od treh možnosti gre. Pri pisanju v notranji pomnilnik visokonapetostnega generatorja in izhodu v sili sta srednja bajta enolično določena. Pri vseh ostalih vrednostih srednjih bajtov gre za povratno informacijo izhoda visokonapetostnega generatorja.

V primeru, da imata oba srednja bajta vrednost 0xFF, visokonapetostni generator aplikaciji sporoči, da je bilo pisanje v notranji pomnilnik uspešno. Na grafičnem vmesniku se nam prikaže pojavno sporočilo o uspešnosti zapisa.

Pri izhodu v sili, visokonapetostni generator pošlje srednja bajta z vrednostjo 0xF0. Na grafičnem vmesniku se nam prikaže pojavno okno s sporočilom, o ročni zaustavitvi visokonapetostnega generatorja.

Po prejemu kakšnih drugih vrednosti na srednjih dveh bajtih, visokonapetostni generator sporoča aplikaciji vrednost izhoda. Napetostni vzorci dejanske vrednosti izhoda se zapisujejo v polje prejetih točk. Ob vsaki na novo shranjeni prejeti točki se risalna mreža osveži, tako se na risalni mreži sproti izrisuje graf poteka napetosti na izhodu visokonapetostnega generatorja. Vrednost izhoda, zapisana v srednjih bajtih ne more biti nikoli enaka vrednosti srednjih bajtov povratne informacije pri zapisu v notranji pomnilnik ali izhodu v sili. Tako ne more priti do dvoumnosti, kaj aplikaciji sporoča visokonapetostni generator.

6. Zaključek

V diplomski nalogi smo napisali program za krmiljenje visokonapetostnega generatorja. Začeli smo s programiranjem aplikacije. Najprej smo sprogramirali grafični vmesnik z risalno mrežo, gumbi in ostalimi gradniki. Uporabljali smo programsko okolje Visual Studio 2010, ki omogoča hitro in preprosto gradnjo grafičnih vmesnikov. Glavni gradnik aplikacije je risalna mreža, za katero smo spisali največ programske kode. Večji problem smo imeli z implementacijo logike mreže, ki smo ji določili natančnost izbire možnih točk [9]. Drugih težav pri razvoju aplikacije ni bilo.

Nato smo pričeli z načrtovanjem protokola za komunikacijo med aplikacijo in mikrokrmilnikom v visokonapetostnem generatorju. Zasnovali smo ga v obliki pošiljanja prenosnih paketov, z enolično določenim začetkom in koncem paketa. Nad podatkovnimi biti paketa smo izračunali ciklični redundančni kod in ga dodali v paket za podatkovnim delom. Vsak paket smo pred prenosom maskirali, tako da ni moglo priti do dvoumnosti, kaj je začetek in konec paketa. S tem smo zagotovili prepoznavo prenosnega paketa na sprejemni strani in možnost preverjanja pravilnega prenosa zaradi izračuna ciklično redundančnega koda. Komunikacijski protokol smo si zamislili in izdelali pravilno, saj pri nadaljnem razvoju ni prišlo do popravkov ali sprememb protokola.

Razvoj programa za mikrokrmilnik smo pričeli na razvojni ploščici Olimex, saj visokonapetostni generator dalj časa ni bil na voljo. Za potrebe simulacije smo Olimexu dodali zunanji pomnilnik Flash za shranjevanje napetostne funkcije, dodaten gumb za primer izhoda v sili in piezo-električni piskač. S piskačem smo simulirali izhod visokonapetostnega generatorja, ki se je odražal kot jakost zvoka piskača. Nato smo začeli razvijati program za mikrokrmilnik. Na začetku smo spoznali programsko okolje µVision, ki je namenjen programiranju mikrokrmilnikov. Najprej smo preverili, če komunikacija med aplikacijo in mikrokrmilnikom deluje pravilno, pošiljanje in sprejemanje pravih vrednosti. Komunikacijski protokol je deloval odlično, tako je bilo potrebno sprejete podatke na mikrokrmilniku le še ustrezno interpretirati in obdelati.

Ko je bil visokonapetostni generator na voljo, smo program za mikrokrmilnik na razvojni plošči uporabili tudi na mikrokrmilniku visokonapetostnega generatorja. Pred tem ga je bilo treba malo popraviti. Po opravljenih testih smo se prepričali, da program za krmiljenje deluje kot mora.

Že med samim razvojem, predvsem pa pri testiranju, smo opazili nekatere možnosti izboljšave. Te se nanašajo predvsem na aplikacijo. Sedaj na risalni mreži izbiramo točke in med njimi se izrisujejo linearne črte, ki v celoti sestavljajo napetostno funkcijo. Smiselno bi bilo dodati funkcionalnost, ki bi med izbranimi točkami izrisala eksponentno ali logaritemsko črto. Na grafični vmesnik bi lahko implementirali vnosno polje za določitev natančnosti risalne mreže in vrednost frekvence vzorčenja funkcije. Sedaj moramo ti dve vrednosti spremeniti v kodi sami. Risalni panel bi lahko poljubno povečali oziroma pomanjšali.

7. Priloge

7.1. Dodatek A

7.1.1. Sprejem paketa

```
void Receive_State (unsigned char c)
{
com1.start = (char)STX;
com1.end   = (char)ETX;
if (com1.sRX == TRUE)
{
if ((c == com1.start) && (com1.sFirst == FALSE))
{
com1.sFirst = TRUE;
com1.stRX  = 0;
com1.RXbuf[com1.stRX++] = c;
}
else if (com1.sFirst == TRUE)
{
com1.RXbuf[com1.stRX] = c;
if (c == ETX)
{
com1.sRX = FALSE;
com1.sFirst = FALSE;
com1.Data_Ready = TRUE;
}
if (com1.stRX < N) com1.stRX++;
else
{
com1.stRX = 0;
com1.sRX = TRUE;
com1.sFirst = FALSE;
}
}
}
} //void Receive_State
```

7.1.2. Obdelava paketa

```
if (com1.Data_Ready == TRUE && !control_proc)
{
/* Escaping decoding */
counter = 0;
crc_check = 0;
for(i = 0; i < com1.stRX; i++)
{
if(com1.RXbuf[i] == 0x7d)
```

```
{
com1.RXbuf_esc[counter] = (com1.RXbuf[i+1]^0x20);
i++;
counter++;
}
else
{
com1.RXbuf_esc[counter] = com1.RXbuf[i];
counter++;
}
}
for(di = 2; di < counter-3; di++)
{
crc_check = CRC_16bCITT(com1.RXbuf_esc[di], crc_check);
}
if (crc_PC == crc_check)
{
//program mode
if(com1.RXbuf_esc[1] == 0x66)
{
com1.RXbuf_esc_data[0] = counter-4; //data_length + fs + data
for(i = 1; i < counter-4; i++)
{
com1.RXbuf_esc_data[i] = com1.RXbuf_esc[i+1];
}
//com1.RXbuf_esc_data[i] = 0xFF;
/* Zapis na FLASH pomnilnik  */
while(flash_busy());
flash_write(1, com1.RXbuf_esc_data, 528); //flash_read(1, com1.RXbuf_flash);
while(flash_busy());
uc_ack(0xFF, 0xFF);
}
//control mode
if(com1.RXbuf_esc[1] == 0x55)
{
fs_h = com1.RXbuf_esc[2];
fs_l = com1.RXbuf_esc[3];
fs_value = ((fs_h << 8) & 0xFF00) + fs_l;
uc_ack(0xEE, 0xEE);
counter_start_control = 1;
control_proc = 1;
i=4;
}
} //ifCRC
com1_reset();
} //com1.Data_Ready == TRUE && !control_proc
```

7.1.3. Kontrolna procedura

```
//Control_procedure
if((control_proc) && (sys_counter_control >= fs_value) && (i < counter-3))
{
out_h = com1.RXbuf_esc[i];
out_l = com1.RXbuf_esc[i+1];
out_value = ((out_h << 8) & 0xFF00) + out_l;
sys_counter_control  = 0;
hvg_write(out_value);
uc_ack( (((unsigned short)vng_o) >> 8) & 0x00ff, ((unsigned short) vng_o) & 0x00ff);
i = i + 2;

if(i >= counter - 3)
{
control_proc = 0;
uc_ack(0xDD, 0xDD); //buttons ON
}
}
```

7.1.4. Programska procedura

```
//Program procedure
if(START_btn && !control_proc)
{
if(flash_r)
{
while(flash_busy());
flash_read(1, com1.RXbuf_flash); //flash_read(1, com1.RXbuf_flash);
while(flash_busy());
flash_length = com1.RXbuf_flash[0];
flash_counter = 3;
fs_h = com1.RXbuf_flash[1];
fs_l = com1.RXbuf_flash[2];
fs_value = ((fs_h << 8) & 0xFF00) + fs_l;
uc_ack(0xEE, 0xEE);
flash_r = FALSE;
}
if(sys_counter >= fs_value) // <= frekvenca sempliranja
{
sys_counter = 0;
out_h = com1.RXbuf_flash[flash_counter];
out_l = com1.RXbuf_flash[flash_counter+1];
out_value = ((out_h << 8) & 0xFF00) + out_l;
hvg_write(out_value);
uc_ack( (((unsigned short)vng_o) >> 8) & 0x00ff, ((unsigned short) vng_o) & 0x00ff);
flash_counter = flash_counter + 2;
}
if(flash_counter >= flash_length)
{
TIPKA = FALSE;
flash_counter = 3;
flash_r = TRUE;
```

```
uc_ack(0xDD, 0xDD);
}
} //while(START_btn && !control_proc)
```

7.1.5. Izhod v sili

```
if(STOP_btn)
{
uc_ack(0xF0, 0xF0);
START_btn = 0;
STOP_btn = 0;
control_proc = 0;
hvg_write(0);
}
```

7.2. Dodatek B

7.2.1. Natančnost risalne mreže

```
int _snapX = 5;
int _snapY = 5;
protected MouseEventArgs MouseSnap(MouseEventArgs e)
{
int px, py;
if(_snap)
{
px = (int)(((float)e.X / _snapX) +0.5f ) * _snapX;
py = (int)(((float)e.Y / _snapY) +0.5f) * _snapY;
}
else
{
px=e.X;
py=e.Y;
}
MouseEventArgs t=new MouseEventArgs(e.Button,e.Clicks,px,py,e.Delta);
return t;
}
```

7.2.2. Funkcionalnost risalne mreže

```
//dodajanje točke na risalno mrežo in avotmatsko zaključevanje
private void gridSnap1_MouseClick(object sender, MouseEventArgs e)
{
if (!function_finish)
{
if (e.Button == System.Windows.Forms.MouseButtons.Left)
{
if (pos.X > temp.X)
{
grid.DrawEllipse(Pens.Red, pos.X - 2, pos.Y - 2, 4, 4);
grid.DrawLine(pen_yellow, temp.X, temp.Y, pos.X, pos.Y);
temp.X = pos.X;
temp.Y = pos.Y;
points.Add(pos);
vertical = true;
}
else if (pos.X == temp.X && pos.Y != temp.Y && vertical)
{
grid.DrawEllipse(Pens.Red, pos.X - 2, pos.Y - 2, 4, 4);
grid.DrawLine(pen_yellow, temp.X, temp.Y, pos.X, pos.Y);
temp.X = pos.X;
temp.Y = pos.Y;
points.Add(pos);
vertical = false;
}
this.gridSnap1.Invalidate();
}
if (e.Button == System.Windows.Forms.MouseButtons.Right && temp.Y != 400)
{
points.Add(new Point(temp.X + gridSnap1.SnapX, 400));
```

```
temp.X = temp.X + gridSnap1.SnapX;
temp.Y = 400;
function_finish = true;
}
} //!function_finish
} //gridSnap1_Mouse_click

//odstranjevanje zadnje točke grafa na risalni mreži
private void gridSnap1_MouseDoubleClick(object sender, MouseEventArgs e)
{
if (temp.X == pos.X && temp.Y == pos.Y) //pozicija enaka zadnji vneseni tocki
{
erased = (Point)points[points.Count - 1];
points.RemoveAt(points.Count - 1);
function_finish = false;
if (points.Count > 1)
{
last = (Point)points[points.Count - 1];
before_last = (Point)points[points.Count - 2];
temp.X = last.X;
temp.Y = last.Y;
if (last.X == before_last.X)
vertical = false;
else
vertical = true;
}
else if (points.Count == 1)
{
last = (Point)points[points.Count - 1];
temp.X = last.X;
temp.Y = last.Y;

if (temp.X != 0)
vertical = true;
else
vertical = false;
}
else
{   //izhodisce, no point in buffer
temp.X = 0;
temp.Y = 400;
vertical = true;
}
}
} //Mouse_doubleClick

//osveževanje risalne mreže
private void gridSnap1_Paint(object sender, PaintEventArgs e)
{
foreach (Point p in points)
{
e.Graphics.DrawEllipse(Pens.Yellow, p.X - 2, p.Y - 2, 4, 4);
e.Graphics.FillEllipse(Brushes.Yellow, p.X - 2, p.Y - 2, 4, 4);
}
drawVoltage(points, pen_yellow, e);
drawVoltage(points_control, pen_green, e);
if (!mouse_enter)
{
pos.X = 0;
pos.Y = 400;
}
//izris kazalca
```

```
if (mouse_enter)
{
e.Graphics.DrawEllipse(Pens.Red, pos.X - 5, pos.Y - 5, 10, 10);
}
if (!function_finish)
{
//izris trenutne crte
if (pos.X > temp.X && mouse_enter)
{
e.Graphics.DrawEllipse(Pens.Yellow, pos.X - 2, pos.Y - 2, 4, 4);
e.Graphics.FillEllipse(Brushes.Yellow, pos.X - 2, pos.Y - 2, 4, 4);
e.Graphics.DrawLine(pen_yellow, temp.X, temp.Y, pos.X, pos.Y);
}
if (pos.X == temp.X && vertical && mouse_enter)
{
e.Graphics.DrawEllipse(Pens.Yellow, pos.X - 2, pos.Y - 2, 4, 4);
e.Graphics.FillEllipse(Brushes.Yellow, pos.X - 2, pos.Y - 2, 4, 4);
e.Graphics.DrawLine(pen_yellow, temp.X, temp.Y, pos.X, pos.Y);
}
}
} //gridSnap1_Paint
private void drawVoltage(ArrayList points, Pen svincnik, PaintEventArgs e)
{
Point[] tocke = new Point[points.Count];
int st = 0;

foreach (Point i in points)
{
tocke[st] = i;
st++;
}
for (int a = 0; a < st; a++)
{
if (a == 0)
e.Graphics.DrawLine(svincnik, 0, 400, tocke[a].X, tocke[a].Y);
else
e.Graphics.DrawLine(svincnik,tocke[a-1].X,tocke[a-1].Y,tocke[a].X,tocke[a].Y);
}
} //drawVoltage
```

7.2.3. Vzorčenje napetostne funkcije

```
public int[] sampling(ArrayList points, int sample_f)
{
float k, n, x1, x2, y1, y2;
Point last = (Point)points[points.Count - 1];
int num_samples = (last.X*10) / sample_f;
int[] sample_values = new int[num_samples+1];
sample_values[0] = sample_f;
int stev = 1;
Point[] tocke = new Point[points.Count + 1];
tocke[0].X = 0;
tocke[0].Y = 400;
int st = 1;
foreach (Point i in points)
{
tocke[st] = i;
st++;
```

```
}
for (int i = 0; i < tocke.Length - 1; i++)
{
x1 = tocke[i].X*10;
x2 = tocke[i + 1].X * 10;
y1 = (400 - tocke[i].Y)*5;
y2 = (400 - tocke[i + 1].Y)*5;
k = ((y2 - y1) / (x2 - x1));
n = y2 - x2 * k;
if (x2 == x1)
sample_values[stev - 1] = (int)y2;
for (int j = (int)x1 + sample_f; j < x2 + sample_f; j += sample_f)
{
sample_values[stev] = (int)((j * k) + n);
stev++;
}
}
return sample_values;
} //sampling
```

7.2.4. Sestava prenosnega paketa in pošiljanje

```
private void button3_Click(object sender, EventArgs e) //send
{
if (buttons_on)
{
if (sp_open)
{
if (!radioButton1.Checked && !radioButton2.Checked)
{
label20.Text = "Warning: Choose mode!";
}
else
{
if (load_true)
{
ushort crc_code;
byte mode; //0x55 - Control, 0x66 - Program
byte crc_high;
byte crc_low;
byte[] package_data = new byte[send_buf.Length * 2];
for (int i = 0; i < send_buf.Length * 2; i += 2)
{
package_data[i] = (byte)((send_buf[i / 2] >> 8) & 0x00FF);
package_data[i + 1] = (byte)(send_buf[i / 2] & 0x00FF);
}
if (radioButton1.Checked) //Control mode
{
mode = 0x55;
cons = sampling_freq;
if (points_control_empty)
points_control_empty = false;
if (!points_control_empty)
{
points_control.Clear();
gridSnap1.Invalidate();
}
}
else //Program mode
```

```
{
mode = 0x66;
}//CRC calculation
crc_code = methods.CalcCrc(package_data);
crc_high = (byte)((crc_code >> 8) & 0x00FF);
crc_low = (byte)(crc_code & 0x00FF);
//data package assembly
byte[] test_package = new byte[package_data.Length + 3];
test_package[0] = mode;
for (int i = 0; i < package_data.Length; i++)
test_package[i + 1] = package_data[i];
test_package[package_data.Length + 1] = crc_high;
test_package[package_data.Length + 2] = crc_low;
//escaping
test_package = methods.escaping(test_package);
//adding stx & etx
byte[] send_package = new byte[test_package.Length + 2];
send_package[0] = stx;
for (int i = 0; i < test_package.Length; i++)
send_package[i + 1] = test_package[i];
send_package[send_package.Length - 1] = etx;
serialPort1.Write(send_package, 0, send_package.Length);
}
else
{
label20.Text = "Warning: No function to send!";
}
} //Choose mode
} //sp_open
else
{
label20.Text = "Warning: No connection with COM port!";
}
} //if(buttons_off)
} //send
```

7.2.5. Izračun ciklične redundance

```
//CRC calculating
public ushort UpdateCrc(ref ushort crc, byte b)
{
crc ^= b;
crc = (ushort)((crc >> 8) | (crc << 8));
crc ^= (ushort)((crc & 0xFF00) << 4);
crc ^= (ushort)((crc >> 8) >> 4);
crc ^= (ushort)((crc & 0xFF00) >> 5);
return crc;
}
public ushort CalcCrc(byte[] data)
{
ushort crc = 0x0000;
for (int i = 0; i < data.Length; i++)
crc = UpdateCrc(ref crc, data[i]);
return crc;
} //CRC calculating
```

7.2.6. Maskiranje podatkov

```
public byte[] escaping(byte[] data)
{
int counter = 0;
for (int i = 0; i < data.Length; i++)
{
if (data[i] == 0x02 || data[i] == 0x03 || data[i] == 0x7d)
counter++;
}
byte[] esc_data = new byte[data.Length + counter];
byte mark = 0x7d;
counter = 0;
for (int j = 0; j < data.Length; j++)
{
switch (data[j])
{
case 0x02:
esc_data[counter] = mark;
counter++;
esc_data[counter] = data[j];
esc_data[counter] ^= 0x20;
counter++;
break;
case 0x03:
esc_data[counter] = mark;
counter++;
esc_data[counter] = data[j];
esc_data[counter] ^= 0x20;
counter++;
break;
case 0x7d:
esc_data[counter] = mark;
counter++;
esc_data[counter] = data[j];
esc_data[counter] ^= 0x20;
counter++;
break;
default:
esc_data[counter] = data[j];
counter++;
break;
}
}
return esc_data;
} //escaping data
```

7.2.7. Poslušanje serijskih vrat

```
private void serialPort1_DataReceived(object sender, SerialDataReceivedEventArgs e)
{
int bytes = serialPort1.BytesToRead;
byte[] buff = new byte[bytes];
serialPort1.Read(buff, 0, bytes);
for (int i = 0; i < buff.Length; i++)
       serial_buff.Add(buff[i]);
hvg_message();
```

```
}
private void hvg_message()
{
int msg_start = -1;
int msg_end = -1;
bool startP = true;
if (serial_buff.Count < 4)
return;
for (int i = 0; i < serial_buff.Count; i++)
{
if (serial_buff[i] == stx && startP)
{
msg_start = i;
startP = false;
}
if (serial_buff[i] == etx)
{
msg_end = i;
if (msg_end - msg_start == 3)
break;
}
}
if (msg_start == -1 || msg_end == -1)
return;
if (msg_end < msg_start)
{
serial_buff.RemoveRange(0, msg_start);
return;
}
List<byte> msg = new List<byte>();
msg = serial_buff.GetRange(msg_start + 1, msg_end - msg_start - 1);
serial_buff.RemoveRange(0, msg_end + 1);
if (msg.Count == 2)
{
if (msg[0] == 0xFF && msg[1] == 0xFF)
{
MessageBox.Show("Writting in Flash successful!");
}
else if (msg[0] == 0xF0 && msg[1] == 0xF0)
{
MessageBox.Show("EMERGENCY EXIT!");
E_Close();
}
else
{
if (msg[0] == 0xEE && msg[1] == 0xEE)
{
cons = sampling_freq;
points_control.Clear();
buttons_on = false;
}
else if (msg[0] == 0xDD && msg[1] == 0xDD)
{
buttons_on = true;
}
else
{
byte value_h = msg[0];
byte value_l = msg[1];
int value = (int)(((value_h << 8) & 0x0000FF00) + value_l);

Point c = new Point(cons / 10, 400 - (int)(value / 5));
```

```
points_control.Add(c);
//drawVoltage(points_control, pen_green);
gridSnap1.Invalidate();
cons += sampling_freq;
}
}
} //msg.Count == 2
} //hvg_message
```

7.3. Kazalo slik

8. Viri

[1] HVG-2000 user manual_v2.pdf

[2] http://www.analog.com/static/imported-files/data_sheets/AD421.pdf

[3] http://srecord.sourceforge.net/crc16-ccitt.html

[4] Olimex STM-P103 Development board users manuel,Olimex Ltd., 2008

[5] STM32F103 Reference Manuel, STMicroelectronics, 2009

[6] Head First C#, 2E: A Learner's Guide to Real-World Programming with Visual C# and .NET, Andrew Stellman and Jennifer Greene, 2010

[7] http://www.keil.com/uvision/

[8] http://www.microsoft.com/visualstudio/eng/products/visual-studio-2010-express

[9] http://stackoverflow.com/questions/1892474/c-sharp-create-snap-to-grid-functionality