Krmilnik grafičnega prikazovalnika

Avtor: Irfan Delalović

Danes smo priče pronicanja elektronike v vse pore našega življenja, posebno pa na področju komunikacij, kjer ima večina naprav vgrajene grafične prikazovalnike (nadalje v članku poimenovan GP). In ravno masovnost GP je bila eden od glavnih razlogov, ki so me spodbudili k izdelavi opisane naprave. Drugi razlog pa je njihova relativno nizka cena, ki jo dosegajo ravno zaradi masovne proizvodnje. Lahko bi rekli, da so tudi glede cene dostopni vsakomur.

Ponuda modelov in proizvajalcev grafičnih modulov je zelo bogata in raznolika, tako da je bila izbira enega samega iz celotne palete kar težka naloga. Izbiro sem si olajšal tako, da sem jo zožal z naslednjimi zahtevami:

- komunikacija displeja z mikrokontrolerjem mora biti paralelna, ker nam omogoča v enoti časa prikazati večje število slik,
- minimalno število barv, ki jih lahko displej prikaže mora biti 65000,
- touch-screen mora biti sestavni del prikazovalnika.

Eden od njih je vsekakor tudi model MI0283QT2, ki ima sve zahtevane karakteristike in še več! Omogoča prikazovanje do 262000 barv (barva je največ 18-bitna), ima ločljivost 320x240 točk, napajalno napetost le 3,2V, vmesnik za povezavo z mikrokontrolerjem pa lahko celo izbiramo (omogoča tudi SPI komunikacijo).

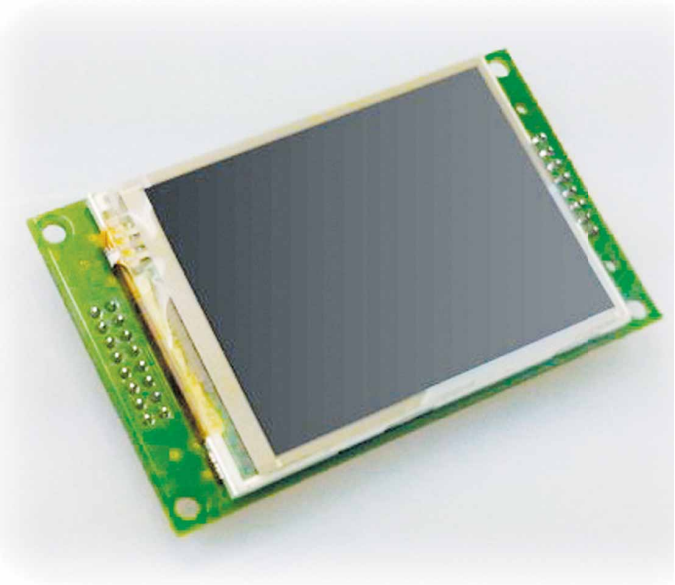
Opisani model je pogosto vgrajen v multimedijske naprave, hitro pa lahko na Internetu o njem najdemo tudi vse potrebne podatke, tako da z opisom ne bom izgubljal besed, razen delov, ki so nepogrešljivi.

Ko izberemo prikazovalnik, nas čaka prav tako pomembna naloga izbire primernega mikrokontrolerja, ki ga bo krmilil in to iz več razlogov. Čeprav bo kasneje opisan zase, naj povem, da je krmilnik grafičnega prikazovalnika samo majhen delček neke zelo zapletene naprave, saj krmilnika ne načrtujemo zato, da bi se s prikazovalnikom igrali. Prav zato mora imeti izbrani mikrokontroler dovolj uporabne periferije in dovolj delovnega pomnilnika tudi za reševanje ostalih zahtevnih nalog. Druga pomembna točka pri izbiri mikrokontrolerja je seveda cena, ki bi morala biti čim nižja in seveda - mikrokontroler mora biti pri izvajanju dovolj hiter in dostopen širokemu krogu uporabnikov.

Zaradi vseh zgoraj navedenih razlogov sem se odločil za ATMEL-ov mikrokontroler ATXMEGA128A1.

Nekatere od njegovih pomembnejših lastnosti so:

- 128 kB flash pomnilnika,
- 8 kB SRAM pomnilnika,
- 4kB EEPROM-a,
- interna 32 MHz in 2 MHz RC oscilatorja,
- 32 kHz RTC oscilator,
- 32-bitni RTC register,
- napajalna napetost od 1.7-3.6V,

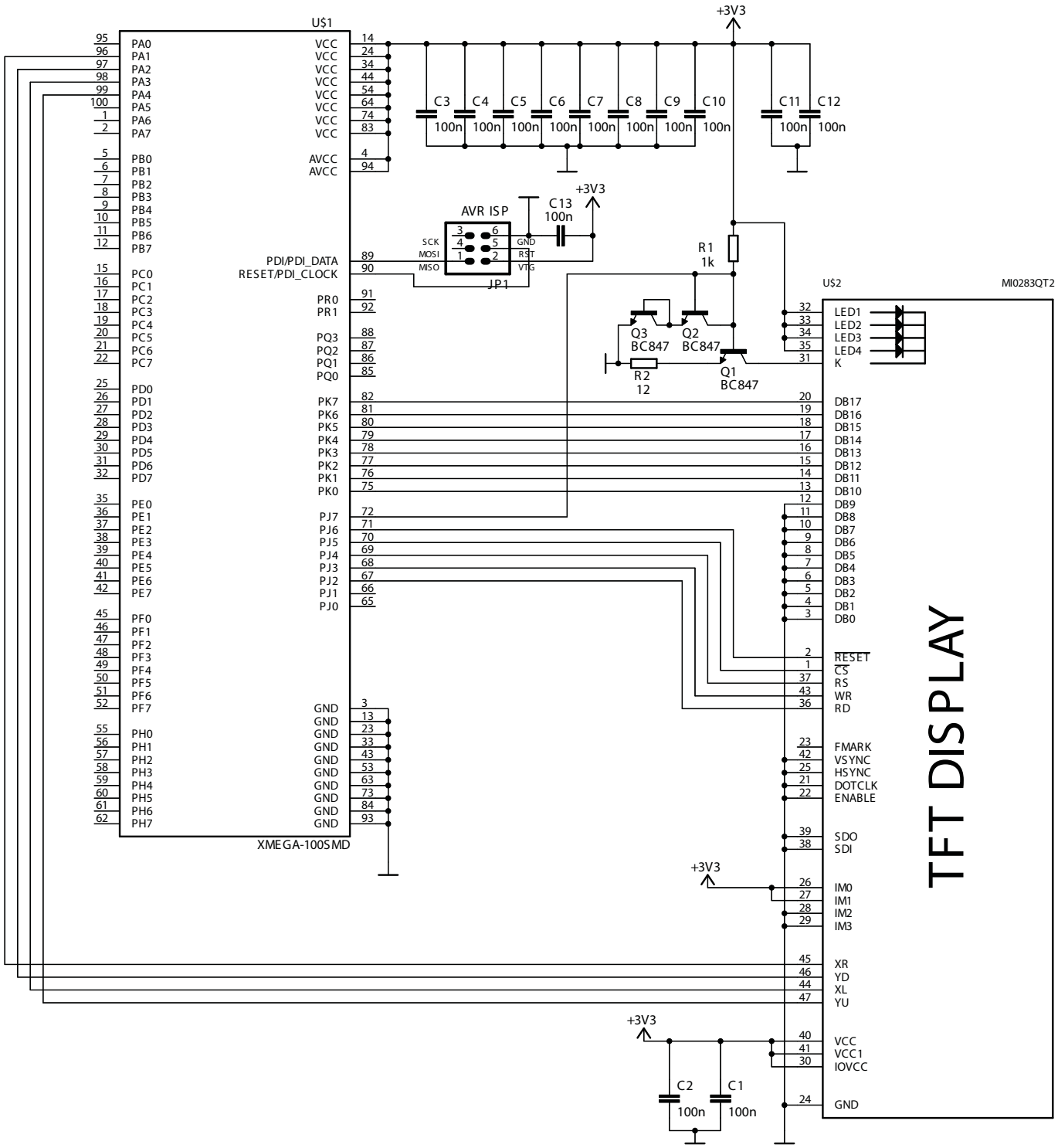


- 8 časovnikov,
- 8 UART serijskih vmesnikov,
- 4 I2C vmesniki,
- 4 SPI vmesniki,
- AES-DES-a kripto vmesnik,
- dva 8-kanalna 12-bitna AD pretvornika z 2M vzorci na sekundo,
- dva 2-kanalna 12-bitna DA pretvornika z 1M vzorci na sekundo.

Čeprav sem omenil le del njegovih karakteristik, lahko vidimo, da gre za zelo močan mikrokontroler, ki ga BASCOM AVR podpira od različice 2.0.5.0 naprej. Najbolj pomembno pri odločitvi pa se mi je zdelo to, da doslej v reviji nisem našel niti enega projekta, kjer bi bil uporabljen ta genialni mikrokontroler, zato bom tokrat izkoristil priložnost in vam ga poskusil čim bolj približati.

Opis električne sheme krmilnika

Gre za kompleksno napravo, katere del je tudi krmilnik grafičnega prikazovalnika (GP), zato je tu prikazan samo tisti del sheme, ki prikazuje povezovanje GP z mikrokontrolerjem, ostali deli pa bodo prikazani po vrsti tako, kot bodo predstavljeni v reviji. Prav tako ni načrta tiskanega vezja, ki bo objavljen v zadnjem delu kot načrt ploščice celotne naprave.



Slika 1: Shema krmilnika

GP je na mikrokontroler priključen prek 8-bitnega podatkovnega vodila in štirimi kontrolnimi signali. Morda vam bo zelo zanimiv način priklopa osvetlitve ozadja GP, ki omogoča konstanten tok skozi LED diode, tako da je osvetljenost GP konstantna, ne glede na spremembo temperature, segrevanje upora R2 in podobno, zato ga bom malo bolj opisal.

Q1 ima vlogo stikala, ki prek upora R1 dobiva napetost na bazo in vklaplja osvetlitev ozadja, razen v primeru, kadar ima pripadajoči priključek na mikrokontrolerju stanje logične "0", oziroma smo prek tega priključka omogočili regulacijo moči osvetlitve in takrat, ko imamo osvetlitev ozadja vključeno, vendar ne uporabljamo regulacije. Regulacija je izvedena s tranzistorjema Q2 in Q3, od katerih je Q2 povezan kot emiterski sledilnik, Q3 pa igra vlogo spremenljivega upora.

Način, kako je GP povezan z mikrokontrolerjem, določamo prek kontrolnih signalov IM0, IM1, IM2 in IM3, ki so v

našem primeru fiksno nastavljeni na 8-bitno podatkovno vodilo, da prihranimo čim več priključkov mikrokontrolerja. Edina pomanjkljivost tega načina je nekaj manjša hitrost prenosa podatkov. Če bi želeli GP povezati kako drugače, lahko več podrobnosti o pomenu teh priključkov preberete v tehnični dokumentaciji prikazovalnika.

Mikrokontrolerja XMEGA128A1 žal ne moremo programirati prek običajnega SPI vmesnika, saj za programiranje uporablja takoimenovani PDI vmesnik, potrebni priključki pa so dostopni prek standardnega 6-pinskega konektora, ki je na shemi tudi prikazan. Od programatorjev, ki ga podpirajo, bom omenil le AVR-ISP in DRAGON od katerih je prvi dostopen tudi neposredno iz BASCOM-a, oba pa sta dostopna iz AVR studija in oba samodejno prepoznata XMEGA mikrokontroler ter nastavita vmesnik. AVR ISP je vseeno veliko bolj prijazen za delo z Bascomom AVR in ga zato toplo priporočam.

Ker smo električni del opisali, bi se sedaj najprej bolj podrobno seznanili s samim GP, z ukazi, ki jih podpira, kot tudi s samo komunikacijo. Pa lepo po vrsti!

Opis grafičnega modula MI0283QT2

Uporabljen grafični modul je namenjen prenosnim multimedijskim napravam, telefonom in podobno. V celoti je izdelan v COG tehnologiji. Krmilniški čip je HX8347D proizvajalca Himax. Skupna poraba je okrog 80mA pri napajalni napetosti 3,2V . Velikost posamezne pike na ekranu je 0,18 x 0,18mm. GP je organiziran v 255 registrov, prek katerih lahko prikazovalnik popolnoma nadziramo.

Opis kontrolnih priključkov

Glede na to, da je vloga kontrolnih priključkov GP enaka za vse paralelne načine delovanja, jih bom na kratko opisal:
Reset. Negativen impulz na tem priključku bo resetiral GP, ob tem pa se bodo registri znotraj GP postavili na začetne vrednosti. Takšen način resetiranja se najpogosteje izvede ob vklopu napajanja, ko se med inicijalizacijo GP resetira programsko. Med resetom morajo biti ostali kontrolni signali na visokem logičnem nivoju.

CS. Z nizkim logičnim nivojem izberemo prikazovalnik, oziroma vklopimo logiko znotraj prikazovalnika. Med komunikacijo oziroma prenosom podatkov mora biti na logični "0", razen med "trdim" resetom (reset z logično "0" na priključku RESET).

WR. Z "Lo" impulzom na tem priključku se v prikazovalnik zapiše vrednost, ki je trenutno na podatkovnem vodilu. Če priključka ne uporabljamo, mora imeti logični "Hi" nivo.

RD. Z "Lo" impulzom na tem priključku se bo na podatkovnem vodilu pojavila vrednost iz registra, ki smo ga pred tem izbrali (naslovili, adresirali) in to vrednost lahko preberemo. Če priključka ne uporabljamo, mora biti priključen na "Hi" logični nivo.

RS. Prek tega priključka označujemo, ali se na podatovnem vodilu nahaja podatek ali ukaz. Logično stanje "Hi" pomeni ukaz, stanje "Lo" pa podatek.

Opis podprogramov za delo z grafičnim prikazovalnikom

Sadaj, ko poznamo pomen kontrolnih signalov, lahko pogledamo, kakšni so posamezni programi, s katerimi na GP pošiljamo ukaze in podatke. Najprej pa nekaj besed o samih podprogramih.

BASCOM omogoča dva načina uporabe podprogramov in odločil sem se za način z uporabo ključne besede SUB, s čimer sem omogočil, da je programiranje prikazovalnika podobno, kot bi bil sestavni del programskega jezika Bascom. Uporaba GOSUB ukazov bi bila precej manj praktična. Poleg tega so vse funkcije pisane po vzoru programskega razvojnega okolja Visual Basic za osebne računalnike (PC), tako da lahko bralci, ki so že delali z grafiko v Visual Basicu preprosto programirajo tudi grafični prikazovalnik.

Začnimo z osnovnimi podprogrami. Ko smo opisovali kontrolne signale, smo povedali, da lahko GP pošljemo ukaz ali podatek, zato sem podprograme za njuno pošiljanje ločil in jima dal različna prepoznavna imena, njihovo obliko pa lahko vidimo v nadaljevanju, torej podprogram "Pošlji_komando":

AX ELEKTRONIKA
GENERALNI ZASTOPNIK ZA SLOVENIJO
AX ELEKTRONIKA D.O.O.
POT HEROJA TRTNIKA 45,
1000 LJUBLJANA
TEL.: 01 549 14 00
INTERNET: WWW.SVET-EL.SI,
E-MAIL: STIK@SVET-EL.SI
BeeProg+ JE UNIVERZALNI PROGRAMATOR NASLEDNJE GENERACIJE, KI SE MU NADGRAJUJE LE PROGRAMSKA OPREMA IN DODAJA PODPORA NOVIM KOMPONENTAM. POSODOBITVE SO BREZPLAČNE IN SPROTI SLEDIJO DOGAJANJU NA PODROČJU PROGRAMABILNIH ELEKTRONSKIH VEZIJ.
BeeProg+

```
Sub Poslji_komando(byval Komanda As Byte)
Lcd_data = Komanda
      Reset Lcd_wr
      Set Lcd_wr
End Sub
```

in podprogram "Poslji_podatek":

```
Sub Poslji_podatek(byval Podatek As Byte)
      Reset Lcd_rs
      Lcd_data = Podatek
      Reset Lcd_wr
      Set Lcd_wr
      Set Lcd_rs
End Sub
```

Razlika med podatkom in ukazom le v krmilnem signalu RS, ki je vedno (večino časa) postavljen na "Hi". To je tako izvedeno iz preprostega razloga, ker vsako komunikacijo začnemo z ukazom. Na ta način prihranimo en ukaz za uvodno postavljanje priključka na "Hi". Verjetno ste tudi opazili, da bi se za pošiljanje podatkov lahko uporabil tudi podprogram za pošiljanje ukazov, vendar je razlika v porabi programskega pomnilnika z dvema podprogramoma zanemarljiva, ločena pa se tudi hitreje izvajata, kar je v našem primeru še kako pomembno.

Čeprav podprograma za "trdi" reset ne bomo pogosto uporabljali, sem za začetni reset GP napisal poseben podprogram:

```
Sub Lcd_resetovanje()
Set Lcd_cs
      Set Lcd_rd
      Set Lcd_wr
      Reset Lcd_reset
      Waitms 50
      Set Lcd_reset
      Waitms 50
End Sub
```

To je obenem tudi zadnja funkcija najnižjega nivoja. Preden nadaljujem, bi morda še pojasnil zakaj sem pisal program na tak način. Zavedati se moramo, da ne gre za pisanje preprostega programa ampak projekt, ki je zelo podoben pisanju gonilnika za kakšno od računalniških perifernih naprav, ki ga bodo uporabljali tudi programi, pisani v višjih programskih jezikih. Mislim, da smo si enotni, da si nihče ravno ne želi raziskovanja po nižjih programskih nivojih, da bi periferijo sploh lahko uporabljal. Ta način je sicer videti daljši in bolj zapleten, vendar je ravno nasprotno: je hitrejši in bolj praktičen za uporabo, poleg tega je lažje iskanje napak, saj nam ni treba neprestano preverjati, ali je bil poslan ukaz ali podatek, ampak preverimo le to, če so bile poslane prave vrednosti.

Omenil sem že, da si prikazovalnik lahko predstavljamo kot nek pomnilnik, katerega delovanje lahko krmilimo prek pomnilnih celic oziroma registrov. Da bi lahko pisali v posamezni register, sem napisal podprogram, ki to omogoča, istočasno pa je to eden od programov naslednjega, "višjega" nivoja in tudi tu lahko vidimo prednost zgoraj omenjenega koncepta podprogramov.

Torej, čisto preprosto: pokličemo podprogram in mu posredujemo le naslov registra , v katerega želimo zapisati vrednost, ki jo posredujemo v obliki drugega parametra. Opazili boste, da se naši doslej napisani "ukazi" niti ne razlikujejo preveč od originalnih BASCOM-ovih.

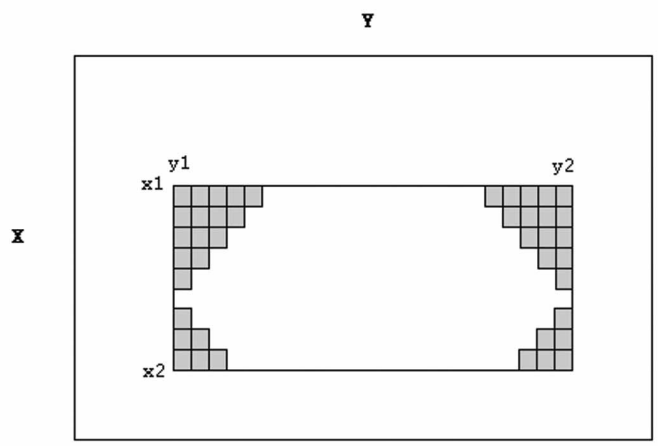
Po priključitvi na napajanje je prikazovalnik pripravljen za delo in vse kar mu je v tem trenutku kristalno jasno je način komunikacije, ki je določen fizičnim povezovanjem priključkov IM0-IM3. Da bi sploh lahko karkoli smiselnega (in očem vidnega) z njim počeli, ga je potrebno pripraviti oziroma inicializirati. Program za inicializacijo je napisan kot poseben program brez vseh parametrov in ko se izvede, je prikazovalnik pripravljen za uporabo. S tem je uporaba GP še bolj prijazna končnim uporabnikom. Na prikazovalniku bo vidna naključno obarvana mreža točk, ki je v bistvu vsebina pomnilnika, ki se ob resetu ne izbriše.

```
Sub Lcd_inicijalizacija()
Call Lcd_resetovanje
Upisi_u_registar &HEA , &H00
Upisi_u_registar &HEB , &H20
Upisi_u_registar &HEC , &H0C
Upisi_u_registar &HED , &H08
Upisi_u_registar &HE8 , &H40
Upisi_u_registar &HE9 , &H38
Upisi_u_registar &HF1 , &H01
Upisi_u_registar &HF2 , &H10
Upisi_u_registar &H27 , &HA3
Upisi_u_registar &H1B , &H1B
Upisi_u_registar &H1A , &H01
Upisi_u_registar &H24 , &H2F
Upisi_u_registar &H25 , &H57
Upisi_u_registar &H23 , &H8D
Upisi_u_registar &H18 , &H36
Upisi_u_registar &H19 , &H01
Upisi_u_registar &H01 , &H00
Upisi_u_registar &H1F , &H88
Waitms 5
Upisi_u_registar &H1F , &H80
Waitms 5
Upisi_u_registar &H1F , &H90
Waitms 5
Upisi_u_registar &H1F , &HD0
Waitms 5
Upisi_u_registar &H17 , &H05
Upisi_u_registar &H36 , &H00
Upisi_u_registar &H28 , &H38
Waitms 40
Upisi_u_registar &H28 , &H3C
Upisi_u_registar &H16 , &H18
End Sub

Sub Lcd_inicijalizacija()
Call Lcd_resetovanje
```

```
Upisi_u_registar &HEA , &H00
Upisi_u_registar &HEB , &H20
Upisi_u_registar &HEC , &H0C
Upisi_u_registar &HED , &H08
Upisi_u_registar &HE8 , &H40
Upisi_u_registar &HE9 , &H38
Upisi_u_registar &HF1 , &H01
Upisi_u_registar &HF2 , &H10
Upisi_u_registar &H27 , &HA3
Upisi_u_registar &H1B , &H1B
Upisi_u_registar &H1A , &H01
Upisi_u_registar &H24 , &H2F
Upisi_u_registar &H25 , &H57
Upisi_u_registar &H23 , &H8D
Upisi_u_registar &H18 , &H36
Upisi_u_registar &H19 , &H01
Upisi_u_registar &H01 , &H00
Upisi_u_registar &H1F , &H88
Waitms 5
Upisi_u_registar &H1F , &H80
Waitms 5
Upisi_u_registar &H1F , &H90
Waitms 5
Upisi_u_registar &H1F , &HD0
Waitms 5
Upisi_u_registar &H17 , &H05
Upisi_u_registar &H36 , &H00
Upisi_u_registar &H28 , &H38
Waitms 40
Upisi_u_registar &H28 , &H3C
Upisi_u_registar &H16 , &H18
End Sub
```

O podprogramu ni kaj povedati, morda le to, da so vrednosti, ki jih vpisujemo v register priporočene vrednosti proizvajalca in jih lahko najdete v tehnični dokumentaciji. Med temi vrednostmi je tudi nastavitev osvetlitve ozadja in lahko trdim, da je to obenem tudi najbolj kritična nastavitev, ki je potrebna. Lahko se namreč zgodi, da boste s kakšnimi vrednostmi pretiravali in se na prikazovalniku ne bo prikazalo nič. Takrat je najboljša rešitev, da pokličete ta program v tej prvotni obliki.



Slika 2: "Delovni" prostor prikazovalnika in orjentacija osi

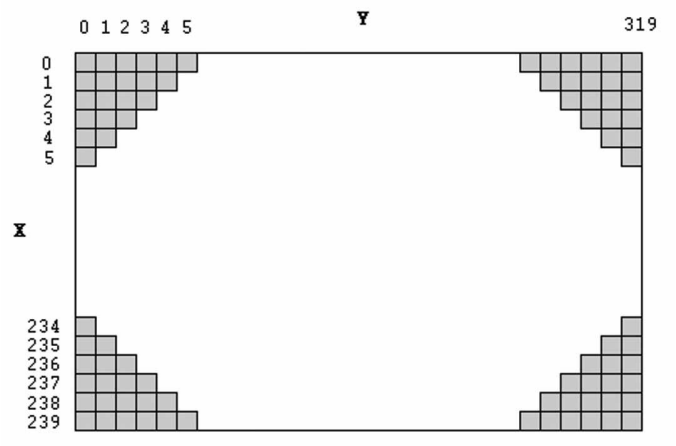
V redu, prikazovalnik je zdaj pripravljen, da nanj začnemo pisati ali risati. Ker je risanje lažje od pisanja (prikazovalnik je grafični) se moramo najprej seznaniti z njegovim "delovnim" prostorom.

Začetek ekrana je v zgornjem levem kotu, kar je malce nenavadno, saj smo vsi nekako navajeni, da je izhodišče koordinatnega sistema v spodnjem levem kotu, vendar se bomo na to že navadili. Zamenjana sta tudi položaja X in Y osi.

Smo edino podjetje v Sloveniji, kjer pod eno streho dobite svoj izdelek na ključ
vse cene so brez DDV-ja
24 URNI PROTOTIPI
PCB 49€
enoslojno ali dvoslojno tiskano vezje, do velikosti 1,5 dm2, dobava v treh tednih
MULTILAYER 99€
štirislojno tiskano vezje, do velikosti 1,5 dm2, dobava v treh tednih
SESTAVA PCB
Nudimo kompletno storitev izdelave tiskanega vezja (ES, DS, MULTI), opremljanje plošč s komponentami, izdelava mask, filmov, šablon ter prodaja laminata in elektronskega materiala.
25 letna tradicija zagotavlja najvišjo kakovost
www.luznar.com / info@luznar.com
Tel.: 04 28 18 800
Tiskana vezja
LUZNAR d.o.o.

Položaj posamezne točke je določen z vsebino registra z naslovom &h22. Vsaka točka je dostopna prek 16-bitne vrednosti, ki predstavlja vrednost barve v kateri bo točka prikazana. Točke se vedno štejejo od 0, oziroma do točke ne moremo direktno dostopati prek tega registra ampak na drugačen način, ki ga bomo obravnavali kasneje. Dobra stran prikazovalnika je, da ima vgrajen samodejno povečevanje naslova točke, tako da nas vsak vpis premakne na lokacijo naslednje točke. To je seveda veliko hitreje, kot če bi posamezne lokacije morali naslavljati sami, kar je še posebno opazno pri risanju slike prek celotnega ekrana. Vidimo je da je relativno preprosto prikazati sliko, dovolj je nasloviti "video" register in potem narediti 76800 vpisov, kar je precej veliko število in zahteva precej časa. Seveda pa bi včasih želeli narisati le del neke slike. K sreči so o tem razmišljali tudi načrtovalci prikazovalnika in so ob tem predvideli tudi možnost delnega vpisa in še več! Enostavno določimo začetno in končno točko na X osi, začetno in končno točko na Y osi in nato vpisujemo posamezne točke na popolnoma enak način, kot bi risali celotno sliko, prikazovalnik pa bo sam poskrbel za prehod v naslednjo vrstico. Uporabno, kajneda? Poglejmo torej kakšen je princip "okenskega" pisanja,

Na osnovi tega je napisan tudi priročen podprogram, v katerega kot parametre posredujemo "dimenzije okna", potem pa se vsi nadaljni vpisi nanašajo le na to območje.



Slika 3: "Okno" prikazovalnika in princip naslavljanja

```
Sub Lcd_region ( x_start As Byte ,_
Y_start As Word , X_kraj As Byte ,_
Y_kraj As Word)
Local Bajt As Byte
Upisi_u_registar &H02 , 0
Upisi_u_registar &H03 , X_start
Upisi_u_registar &H04 , 0
Upisi_u_registar &H05 , X_kraj
Bajt = High(y_start)
Upisi_u_registar &H06 , Bajt
Bajt = Low(y_start)
Upisi_u_registar &H07 , Bajt
Bajt = High(y_kraj)
Upisi_u_registar &H08 , Bajt
Bajt = Low(y_kraj)
Upisi_u_registar &H09 , Bajt
End Sub
```

Še nekaj pomembnega moramo omeniti v zvezi z "oknom". Prikazovalnik namreč na osnovi vrednosti, ki so zapisane v registrih &h02-&h09 naslavlja video pomnilnik tako, da gre vedno od y1 do y2, potem poveča vrednost X in tako do konca, dokler ne doseže x2. Tega načina delovanja ne moremo spremeniti, vendar lahko za določene potrebe zamenjamo položaj X in Y osi, kar bomo pojasnili pri obdelavi prikaza .bmp datotek.

Skoraj vsi prikazovalniki v zbirki ukazov podpirajo neko obliko ukaza za brisanje ekrana, predvsem monokromatski, naš pa takega ukaza nima. Zakaj ne?

Glavni razlog je vsekakor dejstvo, da pri monokromatskem prikazovalniku ekran brišemo tako, da le ugasnemo vsako posamezno piko. Obstaja torej samo "on/off" funkcija na nivoju pike. Pri grafičnih prikazovalnikih pa ni tako, saj ima vsaka pika lastnost barve, ki ni le običajna "on/off" funkcija. Kako potem izbrisati grafični prikazovalnik?

Brisanje ekrana se pri GP izvede z obarvanjem pike z barvo ozadja, oziroma pike celotnega ekrana obarvamo s pikami enake barve. Vse, kar je pri tem treba storiti je, da naslovimo video pomnilnik in vanj od začetka do konca vpišemo vrednost neke barve. Pred tem pa naj pojasnim še način, kako vpišemo barve. Za vsako piko na ekranu je rezervirana 16-bitna spominska lokacija, kamor se vpiše podatek za rdečo, zeleno in modro bravo, z mešanjem vseh treh pa nastane pika z neko želeno barvo.

Da sem dobil barvo iz osnovnih barv, sem napisal funkcijo, ki je popolnoma enaka RGB funkciji v Visual Basic-u, uporabljena pa je predvsem zato, ker je program s tem bolj čitljiv in razumljiv za kasnejše analize.

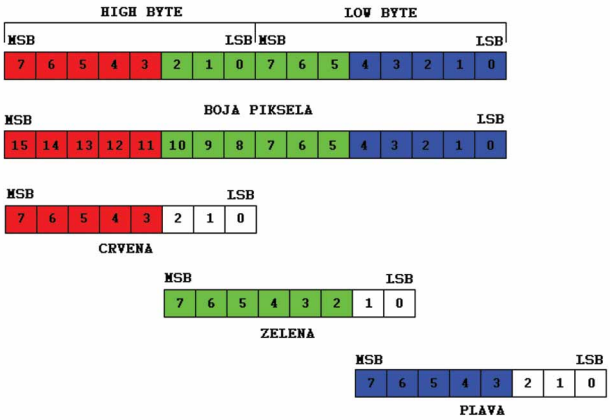
```
Function Rgb(byval Crvena As Byte ,_
Byval Zelena As Byte ,_
Byval Plava As Byte), As Word
Local Bojaz As Byte
Shift Crvena , Left , 3
Plava = Plava And &B00011111
Bojaz = Zelena And &B00111111
Shift Bojaz , Right , 3
Crvena = Crvena Or Bojaz
Bojaz = Zelena
Shift Bojaz , Left , 5
Bojaz = Bojaz And &B11100000
Plava = Plava Or Bojaz
Rgb = Makeint(plava , Crvena)
End Function
```

Funkcija je preprosta za uporabo, za rdečo bravo bi na primer napisali:

```
Dim boja as word
Boja = rgb(30,0,0)
```

Nekatere barve so predstavljene v obliki konstante, to so CMYK barve, ki se v nekaterih aplikacijah zelo pogosto uporabljajo in sem jih predstavil kot primer preproste priprave barve.

```
Const Black = &H0000
Const Blue = &H001F
Const Red = &HF800
Const Green = &H07E0
```



Slika 4: Vpisovanje barve pik na ekranu

```
Const Cyan = &H07FF
Const Magenta = &HF81F
Const Yellow = &HFFE0
Const White = &HFFFF
```

Tako, bervo že znamo določiti, kako pa bi prikazovalnik zbrisali? Da bo program bolj preprost, je je napisan za brisanje celega ekrana, tako da se razlikuje od podprogramov, ki jih bomo obravnavali kasneje, želimo pa tudi ohraniti "kompatibilnost" z ostalimi GP, ki jih podpira BASCOM, glede na to, da večina uporabnikov običajno po inicijalizaciji pobriše tudi ekran.

```
Sub Lcd_clear(boja As Word)
Local Boja_h As Byte
Local Boja_l As Byte
Lcd_region 0 , 0 , 239 , 319
Boja_h = High(boja)
Boja_l = Low(boja)
Reset Lcd_cs
Posalji_podatak &H22
For Pom1 = 0 To 319
For Pom2 = 0 To 239
Posalji_komandu Boja_l
Posalji_komandu Boja_h
Next Pom2
Next Pom1
Set Lcd_cs
End Sub
```

Sledi podprogram za risanje ene točke v izbrani barvi. Ime in vhodni parametri so enaki kot v drugih programih – gre seveda za ukaz "pset".

```
Sub Lcd_pset(tacka_x As Byte ,_
Tacka_y As Word , Tacka_boja As Word)
Lcd_region Tacka_x , Tacka_y , 239 , 319
Reset Lcd_cs
Posalji_podatak &H22
Boja_h = High(tacka_boja)
Posalji_komandu Boja_h
Boja_h = Low(tacka_boja)
Posalji_komandu Boja_h
Set Lcd_cs
End Sub
```

Tu je podobnost s "pset" funkcijo v Visual Basicu še bolj očitna. Če pazljivo pogledamo, bomo videli, da so ukazi sestavljeni iz dveh delov, prvi je "Lcd", ki opredeljuje ukaz kot ukaz, namenjen GP, drugi del ukaza pa je "pset", ki je podobna enakemu ukazu v Visual Basic-u in že samo ime pove, kakšna naj bi bila njena funkcija. X ni Y sta koordinati katerekoli veljavne točke znotraj ekrana. Pravzaprav je ta ukaz morda še najbolj slikovit primer, kako nam lahko pride prav možnost pisanja znotraj določenega "okna", saj bi morali v nasprotnem primeru najprej izvesti potrebno število praznih vpisov, da bi prišli do koordinate želene točke, to pa pomeni porabo časa poleg tega pa še računanje položaja. Ta način je bolj eleganten in kar je še najbolj pomembno – veliko bolj preprost.

Kaj pa, če bi želeli na ekranu narisati črto? Ena izmed rešitev je, da znotraj for-next zanke rišemo pike na želenih koordinatah, vendar s stališča hitrosti to ni ravno najboljša rešitev. Visual Basic ima za risanje pravokotnika in črte isti ukaz, razlika je le v parametrih, s katerimi določamo, ali gre za črto ali pravokotnik, pri pravokotniku pa določimo še to, ali je prazen ali "poln". V želji, da bi bili klici ukazov čim manj zapleteni, sem se odločil za različne klice za prazen in "poln" pravokotnik. Obema je skupno, da "znata" risati črte, zato za risanje črte nisem pisal posebnega programa.

Na ta način lahko narišemo le črte na eni izmed osi, res pa je, da je debelina te črte lahko različna. V kakšni od prihodnjih različic bom to verjetno še nadgradil, za zdaj pa je risanje črt omejeno z orientacijo le v eni izmed obeh osi.

Z neko barvo izpolnjen pravokotnik narišemo z naslednjim podprogramom:

```
Sub Lcd_fillbox(byval X0 As Byte ,_
Byval Y0 As Word , Byval X1 As Byte ,_
Byval Y1 As Word , Byval Boja As Word)
Lcd_region X0 , Y0 , X1 , Y1
Boja_h = High(boja)
Boja_l = Low(boja)
Reset Lcd_cs
Posalji_podatak &H22
For Pom1 = X0 To X1
For Pom2 = Y0 To Y1
```

```
Posalji_komandu Boja_h
Posalji_komandu Boja_l
Next Pom2
Next Pom1
Set Lcd_cs
End Sub
```

Rutina za prazen pravokotnik je, lahko bi rekli neupravičeno, znatno daljša od prejšnje. Zakaj je to potrebno, če bi lahko isto nalogo opravili tako, da bi najprej izrisali pravokotnik v želeni barvi, znotraj njega pa še drugi pravokotnik, ki bi bil za debelino črte manjši od prejšnjega?

Želel sem, da je mogoče prazen pravokotnik narisati okrog že narisane grafike na ekranu, ne da jo pri tem izbrisali, kar v omemnjenem primeru ne bi bilo mogoče. Tako napisan program je res daljši, zato pa imamo na razpolago več različnih možnosti uporabe.

```
Sub Lcd_box(byval X_start As Byte ,_
Byval Y_start As Word ,_
Byval X_end As Byte , Byval
Y_end As Word ,_
Byval Sirina As Byte ,_
Byval Boja_linije As Word)
      Boja_h = High(boja_linije)
      Boja_l = Low(boja_linije)
      Tempy = Y_start + Sirina
      Lcd_region X_start , Y_start ,_
      X_end , Tempy
      Reset Lcd_cs
            Posalji_podatak &H22
            For Pom1 = X_start To X_end
               For Pom2 = Y_start To Tempy
               Posalji_komandu Boja_h
               Posalji_komandu Boja_l
                  Next Pom2
            Next Pom1
      Set Lcd_cs
      Tempx = X_end - Sirina
      Lcd_region Tempx , Y_start ,_
      X_end , Y_end
      Reset Lcd_cs
            Posalji_podatak &H22
            For Pom1 = Tempx To X_end
               For Pom2=Y_start To Y_end
                Posalji_komandu Boja_h
                   Posalji_komandu Boja_l
               Next Pom2
            Next Pom1
      Set Lcd_cs
      Tempy = Y_end - Sirina
      Lcd_region X_start , Tempy , X_end ,_
      Y_end
      Reset Lcd_cs
            Posalji_podatak &H22
            For Pom1 = X_start To X_end
              For Pom2 = Tempy To Y_end
                    Posalji_komandu Boja_h
                    Posalji_komandu Boja_l
                  Next Pom2
              Next Pom1
      Set Lcd_cs
      Tempx = X_start + Sirina
      Lcd_region X_start, Y_start, Tempx ,_
      Y_end
      Reset Lcd_cs
            Posalji_podatak &H22
            For Pom1 = X_start To Tempx
              For Pom2 = Y_start To Y_end
                Posalji_komandu Boja_h
                Posalji_komandu Boja_l
              Next Pom2
            Next Pom1
      Set Lcd_cs
End Sub
```

Nekateri boste hitro opazili, da gre za štirikratni klic zgornjega podprograma z nekaj malega računanja. Prav to je tudi bil namen takšnega pisanja podprogramov – vidimo, da jih je sedaj zelo preprosto kombinirati, ne da bi pri tem morali poznati njihovo zgradbo in delovanje na najnižjem nivoju. Cena te elegance in ugodja je počasnejše izvajanje kode v nekaterih delih programa, ki pa ga kot uporabnik sploh ne opazimo

Zdaj že imamo nekakšno paleto ukazov za ustvarjanje lastne grafike, recimo tipk ali stikal, ki bi bili enakovredni tistim v Windows-ih, manjka pa nam pa pisanje nekega besedila na ekran. To je bila naloga, za katero sem porabil največ časa, vendar ne toliko za samo programiranje, kot za izbiro najboljše končne rešitve.

Čeprav je pisanje na ekran videti bolj enostavno od risanja, je v resnici čisto drugače! Glavni razlog za to je, da GD ne pozna besedila kot zaporedja črk, ampak kot sliko. Res je, da so tudi vsi alfanumerični prikazovalniki narejeni kot grafični, vendar imajo za vsak znak že vgrajeno matrico točk z določeno ASCII kodo, zato je za prikaz posameznega znaka dovolj poslati njegovo kodo. Pri čisto grafičnem prikazovalniku displeja tega seveda ni, zato moramo za prikaz znakov poskrbeti sami. Del rešitve se nam zasvetlika že takoj ko rečemo, da je vsaka črka pravzaprav slika.

Dovolj je torej, da naredimo tabelo znakov, jo pretvorimo v pike in tabelo pik za vse velikosti znakov shranimo v spominskem prostoru mikrokontrolerja. Veliko dela za navidez čisto preprosto opravilo! Vprašamo se, če je to mogoče narediti tudi kako drugače in kako bi nam pri tem lahko pomagal BASCOM.

Za delo s fonti na grafičnih prikazovalnikih ima BASCOM zelo uporabno orodje Font Editor, ki ga lahko najdemo v menuju Tools/Plugins/Font Editor. Uporabljamo ga za oblikovanje fontov, ki jih bomo uporabljali za izpis na GP. Ustvarjeni fonti se potem čisto preprosto dodajo s pseudoinstrukcijo $include na koncu programa. Ker sem imel že od prej izdelanih nekaj fontov, se mi je zdelo pametno, da bi jih kar uporabil, vendar sem moral pred tem najprej raziskati, kako Bascom-ove fonte uporabiti.

Struktura je dokaj enostavna: vsakemu znaku pripada matrica točk, do katere lahko dostopamo prek ASCII kode tega znaka. To je idealno, saj ne potrebujemo kakšnih nadnaravnih sposobnosti da bi ugotovili, kje v tabeli se znak nahaja, ampak le njegovo ASCII vrednost. Druga dobra stvar pri tem pa je, da je delo s tabelami v BASCOM-u več kot dobro podprto, zato je bilo pisanje programa res enostavno.

```
Sub Lcd_text(byval S As String ,_
Byval Xoffset As Byte, Byval Yoffset As Word ,_
Byval Fontset As Byte, Byval
Forecolor As Word ,_
Byval Backcolor As Word)
Local Tempstring As String
* 1, Temp As Word
Local A As Word , Pixels As Byte ,_
Count As Byte, Carcount As
Byte, Lus As Byte
Local Row As Byte , Byteseach As Byte ,_
Blocksize As Byte , Dummy As Byte
Local Colums As Byte, Columcount As Byte,_
Rowcount As Byte , Stringsize As Byte
Local Pixelcount As Byte

Stringsize = Len(s) - 1
For Carcount = 0 To Stringsize
If Fontset = 1 Then Restore Font8x8
If Fontset = 2 Then Restore Font16x16
If Fontset = 3 Then Restore Font6x8
If Fontset = 4 Then Restore Font5x5
If Fontset = 5 Then Restore Font12x16
If Fontset = 6 Then Restore Font20x24
Temp = Carcount + 1
Tempstring = Mid(s , Temp , 1)
Read Row
Read Byteseach
Read Blocksize
Read Dummy
Temp = Asc(tempstring) - 32
For Lus = 1 To Temp
For Count = 1 To Blocksize
Read Pixels
Next Count
Next Lus
Colums = Blocksize / Row
Row = Row * 8
Row = Row - 1
Colums = Colums - 1
For Rowcount = 0 To Row Step 8
A = Rowcount + Yoffset
For Columcount = 0 To Colums
Read Pixels
Xpos = Columcount
Temp = Carcount * Byteseach
Xpos = Xpos + Temp
Xpos = Xpos + Xoffset
For Pixelcount = 0 To 7
Ypos = A + Pixelcount
Pixelword = Pixels.0
If Pixelword = 1 Then
Pixelword = Forecolor
Else
Pixelword = Backcolor
End If
Lcd_pset Xpos , Ypos , Pixelword
Shift Pixels , Right
Next Pixelcount
Next Columcount
Next Rowcount
Next Carcount
End Sub
```

Podprogram je zelo podoben Bascom-ovemu ukazu LCD DAT vendar je nadgrajen z možnostmi ki jih ponuja grafični prikazovalnik, to pa je posredovanje parametrov barve znaka in barve ozadja ter seveda velikosti fonta, s katerim pišemo. Pri tem nismo omejeni na pisanje "po vrsticah" kot z Bascom-ovim ukazom, ampak ga lahko pozicioniramo na nivoju posamezne pike. Besedilo, ki ga želimo izpisati, se podprogramu posreduje kot besedilo, vse številčne vrednosti pa moramo pred tem tudi pretvoriti v znake, drugače bi bili lahko rezultati popolnoma različini od pričakovanih.

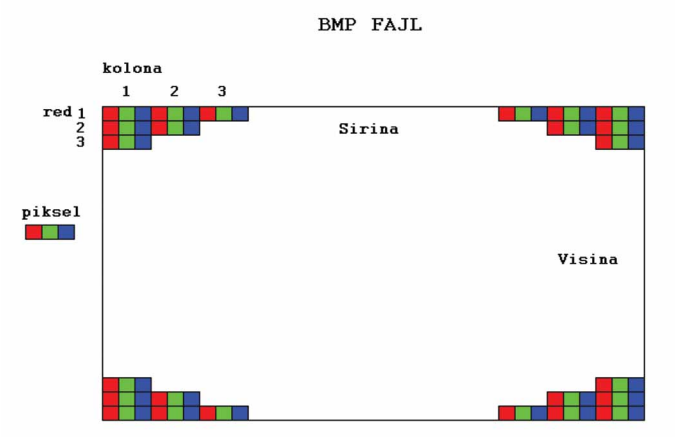
Od grafičnih funkcij ne potrebujemo ničesar več, razen morda nečesa, kar bom opisal v nadaljevanju. Recimo da želimo imeti vmesnik, ki bi bil takšen kot je običajen pri današnjih telefonih kar pomeni, da za njihovo shranjevanje

AX ELEKTRONIKA
GENERALNI ZASTOPNIK ZA SLOVENIJO

AX ELEKTRONIKA D.O.O.
POT HEROJA TRTNIKA 45,
1000 LJUBLJANA
TEL.: 01 549 14 00
INTERNET: WWW.SVET-EL.SI,
E-MAIL: STIK@SVET-EL.SI

SMARTPROG2 JE VSESTRANSKI 40-PINSKI PROGRAMATOR Z ISP IZBIRO IN Z USB POVEZAVO NA PC, KVALITETEN HARDVER JE NADPOLNJEN Z VRHUNSKIM PROGRAMOM, KI KOT STANDARDE PODPIRA VSE MS WINDOWS OPERACIJSKE SISTEME (OD WIN.95 DO WIN7). PROGRAMIRA VEČ KOT 24100 RAZLIČNIH MIKROKONTROLERJEV.

SmartProg2
universal 40-pindrive programmer



Slika 5: BMP datoteka

potrebujemo dokaj velik spominski prostor. Res je sicer, da je mogoče veliko doseči že s shranjevanjem slik v spominski prostor mikrokontrolerja, vendar je vmesnik lažje narisati v enem izmed široko dostopnih grafičnih orodij in ga kot datoteko enostavno prenesti recimo na SD kartico v enem od nestisnjenih formatov kot je recimo .bmp. Zato zdaj poglejmo kakšna je struktura takšne datoteke.

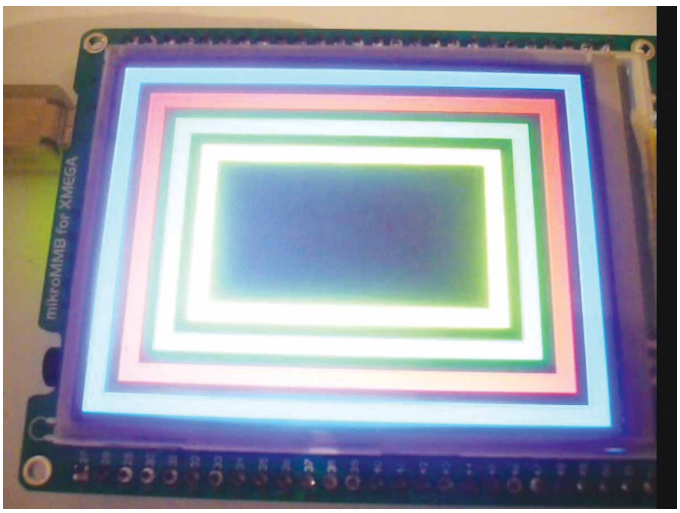
BMP datoteko sestavljata dva dela, glava in podatkovni del. V glavi so zapisani podatki o sami datoteki, v podatkovnem delu pa so zapisane vrijednosti barve posameznih pik v 24-bitnem formatu, pri čemer je po en 8-bitni byte namenjen rdeči, zeleni in modri barvi in sicer v obratnem vrstnem redu, kot se prikazujejo. Podatki se torej zapisujejo tako, da najprej zapišemo vrednost modre, zelene in rdeče barve za zadnjo piko na desni strani zadnje vrstice, potem spet trije byti za eno piko pred njo in tako naprej vse do začetka zadnje vratice, potem pa naprej na konec predzadnje vrstice. To se ponavlja vse do skrajno leve pike prve vrstice. Očitno je, da bo imel podatkovni del 3 x več bytov kot je skupno število vseh pik neke BMP slike.

Kakšni podatki pa se potem nahajajo v glavi datoteke? Tu so zapisani podatki o tipu datoteke, širini in višini slike in še nekateri drugi, vendar se bomo mi osredotočili predvsem na naslednje:

- 11,12,13,14 vsebujejo podatek o številu bytov, od koder se začne data polje. Podatek je tipa long, prvi LSB, zadnji MSB bajt,
- 19, 20, 21, 22 vsebujejo podatek o dolžini slike v pikah. Podatek je tipa long, prvi LSB, zadnji MSB bajt,
- 23, 24, 25, 26 vsebujejo podatek o širini slike v pikah. Podatek je tipa long, prvi LSB, zadnji MSB bajt.

Format je dokaj preprost razen razen ene podrobnosti, namreč da je slika v bistvu zapisana inverzno glede na X in glede na Y os, razen tega pa še podatek o barvi, kjer sta rdeča in modra zamenja mesti. Da bi bila stvar malce lažja, sem napisal podprogram, ki sliko z SD kartice prikaže na ekranu. Vse kar je treba storiti je to, da ob klicu podprograma kot parameter navedemo ime datoteke in koordinato točke, kjer naj bi se slika izrisala. V BASCOM-u še ni podobnega ukaza, zato je z njim uporabna vrednost krmilnika prikazovalnika dvignjena na znatno večji nivo. Oglejmo si program, ki nam to omogoča:

```
Sub Lcd_showpicture_bmp(byval Filename$ As String ,_
Byval Xoffset As Byte , Byval Yoffset As Word)
Local Ff As Byte
Ff = Freefile()
Open Filename$ For Binary As #ff
Get #ff , Pixelword , &H0B
Incr Pixelword
Get #ff , Pom_b , &H13
Decr Pom_b
Get #ff , Pom_w , &H17
Decr Pom_w
Lcd_mode 4
Pom_b = Xoffset + Pom_b
Pom_w = Yoffset + Pom_w
Lcd_region Xoffset , Yoffset , Pom_b ,_
Pom_w
Reset Lcd_cs
Posalji_podatak &H22
Do
Get #ff , Boja1 , Pixelword
Incr Pixelword
Get #ff , Boja2 , Pixelword
Incr Pixelword
Get #ff , Boja3 , Pixelword
Incr Pixelword
Shift Boja1 , Right , 3
Shift Boja2 , Right , 2
Shift Boja3 , Right , 3
Prevpixel = Rgb(boja3 , Boja2 , Boja1)
Boja_h = High(prevpixel)
Posalji_komandu Boja_h
Boja_h = Low(prevpixel)
Posalji_komandu Boja_h
Loop Until Eof(#ff) <> 0
Close #ff
Set Lcd_cs
Lcd_mode 0
End Sub
```



Dokaj preprosto kajneda? Verjetno ste opazili, da smo uporabili nov ukaz krmilnika, namreč Lcd_mode. Ta ukaz določa, kakšna je orientacija posameznih osi na ekranu. Opis tega ukaza sem prihranil za konec, ker šele tu dobi svojo pravo vrednost in ga morda spremlja tudi najlepši primer uporabe. Poglejmo kakšne načine delovanja omogoča ta ukaz:

- običajni način delovanja. To je privzeti način delovanja in se tudi najpogosteje uporablja,
- zrcaljenje glede na y os, kadar želimo sliko zrcaliti glede na y os,
- zrcaljenje glede na x os, kadar želimo sliko zrcaliti glede na x os,
- zrcaljenje glede na obe osi,
- zamenjava osi. X postane Y, Y postane X.
- enako kot zgoraj, poleg tega je še zrcaljenje glede na X os,
- enako kot zgoraj, poleg tega je še zrcaljenje glede na Y os,
- zamenjana sta položaja X in Y razen tega sta obe osi še zrcaljeni.

Podprogram, ki nam to omogoča je zelo preprost:

```
Sub Lcd_mode(byval Lcdmod As Byte)
If Lcdmod = 1 Then
Upisi_u_registar &H16 , &H98
Elseif Lcdmod = 2 Then
Upisi_u_registar &H16 , &H58
Elseif Lcdmod = 3 Then
Upisi_u_registar &H16 , &HD8
Elseif Lcdmod = 4 Then
Upisi_u_registar &H16 , &H38
Elseif Lcdmod = 5 Then
Upisi_u_registar &H16 , &H78
Elseif Lcdmod = 6 Then
Upisi_u_registar &H16 , &HB8
Elseif Lcdmod = 7 Then
Upisi_u_registar &H16 , &HF8
Else
Upisi_u_registar &H16 , &H18
End If
End Sub
```

To je obenem tudi zadnji ukaz, ki ga krmilnik trenutno pozna. Glede na trenutne potrebe jih je čisto dovolj, lahko pa se zgodi, da bom čez nekaj časa dodal še ukaza "Circle" in "Line", vendar o tem kaj več takrat, ko bosta ukaza resnično dodana v nabor ukazov, ki jih krmilnik podpira. V naslednjem prispevku bomo napisali nekaj programov, do takrat pa uživajte v slikah prikazovalnika in si morda celo ogledajte film prikazovalnika med delovanjem. Na koncu sem pripravil še pregleden spisek vseh ukazov, ki jih krmilnik podpira, glej tabelo 1.

www.svet-el.si

Lcd_upisi_u_registar register,vrednost	Vpiše določeno vrednost v določen register GP
Lcd_posalji_komandu ukaz	Pošlje določen ukaz na GP
Lcd_posalji_podatak podatek	Pošlje določen podatek na GP
Lcd_inicijalizacija	Pripravi GP za uporabo
Lcd_resetovanje	Resetira GP
Lcd_mode mod_rada	Postavi GP v enega od načinov delovanja
Lcd_clear boja_pozadine	Briše ekran z določeno barvo
Lcd_region xstart,ystart,xkraj,ykraj	Določi območje na ekranu za vpis pik
Lcd_fillbox xstart,ystart,xkraj,ykraj,boja	Črta ali poln pravokotnik na ekranu v izbrani barvi
Lcd_box xstart,ystart,xkraj,ykraj,debljina,boja	Na ekranu nariše okvir izbrane debeline in barve
Lcd_pset x,y,boja	Nariše piko na ekranu v izbrani barvi
Lcd_text tekst,x,y,broj_fonta,boja_teksta,boja_pozadine	Na ekranu izpiše besedilo
Lcd_showpicture_bmp ime_slike,x,y	Na ekranu prikaže sliko z zunanje SD kartice
Rgb crvena,zelena,plava	Generira 16-bitno barvo iz osnovnih barv

Tabela 1