Osnove mikroprocesorske elektronike

Vaja 1: Začetek dela z mikrokrmilniki

Naloge:

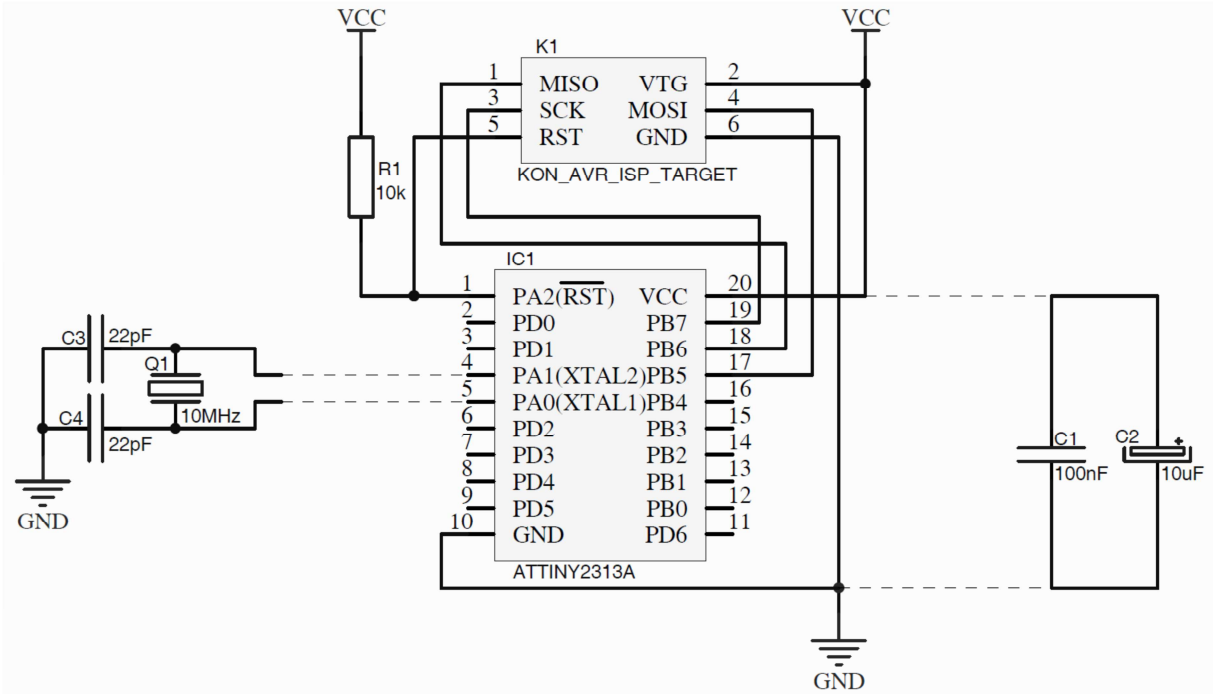
- Na prototipni plošči (protoboard) sestavite minimalno vezje potrebno za delovanje in programiranje mikrokrmilnika ATtiny2313A. Na priključek PD6 mikrokrmilnika priključite (preko ustreznega predupora) LED diodo.
- V Atmel Studio prepišite program, ki prižge LED diodo in ga prevedite. Sprogramirajte mikorokrmilnik in poženite program.
- Program dopolnite z zbirniškimi direktivami, da bo postal čitljiv.
- Program razčlenite v podprogram za inicializacijo in glavno zanko.

Navodila:

Minimalno vezje

Mikrokrmilnik mora imeti za delovanje napajanje, oscilator in priključek $\overline{\text{RESET}}$ priključen na logično 1. Nekateri mikrokrmilniki, vključno z ATtiny2313A, imajo oscilator že vgrajen, zato ni nujno potreben zunanji oscilator. Večina AVR mikrokrmilnikov ima možnost serijskega programiranja ISP (In-circuit Serial Programming), ki deluje preko štirih žic – MISO, MOSI, CLK in RESET.

Minimalno vezje za delovanje in programiranje mikrokrmilnika ATtiny2313A je prikazano na sliki 1. S črtkanimi črtami sta dodana blokirna kondenzatorja, ki blokirata motnje v napajanju in s tem povečata zanesljivost delovanja procesorja ter kristalni oscilator, ki je potreben le, če z notranjim oscilatorjem nismo zadovoljni.



Slika 1: Minimalno vezje za delovanje in programiranje ATtiny2313A z dodanima blokirnima kondenzatorjema (C1, C2) in kristalnim oscilatorjem (Q1, C3, C4).

Prvi program

V Atmel Studiu ustvarite nov projekt, vanj prepišite spodnji program in ga prevedite:

```
sbi   0x11,6
sbi   0x12,6
rjmp  PC-0
```

Program naložite v simulator, odprite si okna »Processor View« in »I/O View«, izvajajte program po korakih in pri vsakem koraku opazujte, kako se spremenijo stanja registrov.

Ko ste zadovoljni z delovanjem programa v simulatorju, sprogramirajte mikrokrmilnik.

Zbirnik

Prejšnji program, ki prižge, eno LED diodo zgleda v dvojiškem zapisu strojne kode tako:

```
1001101010001110
1001101010010110
1100111111111111
```

Oziroma če to pretvorimo v šestnajstiški zapis za malo lažjo berljivost tako:

```
9A8E
9A96
CFFF
```

Vsak ukaz je sestavljen iz nekaj bitov, ki povedo za kateri ukaz gre in iz preostalih bitov, ki so parametri za ta ukaz. Včasih je bilo treba za vsak ukaz vedeti kako je sestavljen in ga v celoti vnesti v spomin. Ker to duhomorno delo sedaj enostavno izvede vsak računalnik, je ta del prepuščen računalniku. Program, ki besedne ukaze prevaja v strojno kodo se imenuje zbirnik. Sedaj je treba točno sestavo ukazov vedeti le redkokdaj, vendar če te informacije vseeno potrebujemo, so na voljo v pomoči Atmel Studia.

Mikrokrmilniki imajo poleg splošnih spominskih lokacij tudi takšne, ki vplivajo na delovanje mikrokrmilnika. Takšen primer so registri vhodno/izhodnih portov. Vsi spominski naslovi so po vrsti oštevilčeni od 0 naprej. Če hočemo spremeniti delovanje vhodno/izhodnega porta, moramo v ustrezen register napisati neko vrednost. Torej moramo vedeti številko tega registra. Npr.:

```
sbi   0x12,6
```

Številka 0x12 (oz. desetiško 18) je številka registra, s katerim lahko nastavimo vrednost na portu D. Ker je število registrov ogromno, bi bilo zelo težko programirati, če bi morali ves čas gledati v tabele, na katerem naslovu je posamezen register. Zato so zbirniki razširjeni z dodatnimi zbirniškimi ukazi (ang. assembler directive), ki jih ne izvaja procesor, temveč le povejo zbirniku, kako naj prevaja program.

En najpomembnejših zbirniških ukazov je .equ, ki določeni vrednosti da ime:

```
.equ      PORTD = 0x12
```

Sedaj lahko namesto številke 0x12 v programu uporabljamo ime PORTD, kar zelo olajša pisanje programa in izboljša njegovo čitljivost. Registrov je veliko, zato je pisanje .equ stavkov za cel procesor zelo dolgotrajno. Ker takšen seznam potrebuje vsak programer, jih proizvajalci mikrokrmilnikov pripravijo vnaprej. Atmel Studio samodejno vključi datoteko z definicijami registrov za izbrani procesor v projekt. Če uporabljamo starejšo različico Atmel Studia ali kateri drug zbirnik, pa lahko datoteko z definicijami registrov vključimo v svoj program z ukazom .include:

```
.include "tn2313Adef.inc"
```

Seveda je za vsak mikrokrmilnik pripravljena druga datoteka definicij registrov. V Atmel Studiu 6 se nahajajo v »C:\Program Files\Atmel\Atmel Studio 6.0\extensions\Atmel\AVRAssembler\2.1.51.64\avrassembler\include\«.

Poleg imen spremenljivk včasih potrebujemo tudi imena za določene naslove v programskem spominu. To potrebujemo, če želimo da program med izvajanjem tja skoči. Ime programskega naslova se imenuje labela (ang. label):

```
    rjmp  SKOCI_SEM
    …
SKOCI_SEM:
```

Pri pisanju zbirniških programov potrebujemo še zbirniške ukaze `.cseg`, `.dseg` in `.eseg`. `.cseg` pomeni, da bo sledil segment programske kode (code segment), `.dseg`, da bo sledil segment podatkovnega spomina (data segment) in `.eseg`, da bo sledil segment podatkov za v EEPROM (EEPROM segment). Če ne podamo nobenega od teh ukazov zbirnik privzame da vse kar napišemo spada v `.cseg`.

V kombinaciji z ukazi .cseg, .dseg in .eseg lahko uporabimo ukaz .org, s katerim lahko nastavimo začetek trenutnega segmenta.

Znotraj segmenta .dseg lahko uporabimo zbirniški ukaz .byte, s katerim rezerviramo en ali več byte-ov spomina za neko spremenljivko. Npr.

```
.dseg
mojaspremenljivka: .byte 1
moja_tabela:       .byte 10
```

Razlago za ostale zbirniške ukaze lahko najdete v pomoči Atmel Studia (»Help/View Help«, v kazalu na levi izberete »Library Home« in nato »AVR Assembler User Guide«).

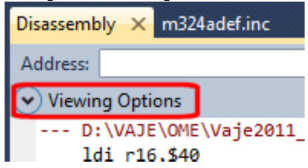
S pomočjo opisanih zbirniških ukazov lahko isti program napišemo še enkrat v precej bolj berljivi obliki:

```
.include "tn2313Adef.inc"

    sbi   DDRD,6   //pin 6 registra DDRD nastavi na 1
    sbi   PORTD,6  //pin 6 registra PORTD nastavi na 1
KONEC:
    rjmp  KONEC
```

Ko je program preveden in naložen v simulator si lahko pogledamo strojno kodo programa s pomočjo razzbirnika (ang. Disassembler). Razzbirnik vključite z »Debug/Windows/Disassembly« ali kombinacijo tipk Alt+8.

Vključite strojno kodo – kliknite »Viewing options«:



In nato še »Show code bytes«.

Isto (a manj podrobno) lahko vidite, če vključite pogled na spomin procesorja (Alt+6).

Podprogrami in inicializacija

Pri reševanju problemov je problem smiselno razbiti na več manjših problemov in reševati vsakega posebej. Prav tako je smiselno programe razbiti na več podprogramov, ki jih potem kličemo iz glavne zanke. Prvi podprogram, ki ga ponavadi pokličemo še pred začetkom glavne zanke je inicializacija, ki nastavi vso periferno opremo mikrokrmilnika v takšno stanje, kot ga potrebuje naša aplikacija.

V našem programčku spada pod inicializacijo nastavitev smeri pina PD6:

```
sbi   DDRD,6
```

V glavni zanki pa lahko LED diodo prižgemo:

```
sbi   PORTD,6
```

Program za prižiganje LED diode, ko ga razbijemo na podprograme zgleda tako:

```
.include "tn2313Adef.inc"

    rcall   Init
MAIN:
    rcall   PrizgiLED
    rjmp  MAIN

Init:
    sbi   DDRD,6
    ret

PrizgiLED:
    sbi   PORTD,6
    ret
```

Prepišite ga v Atmel Studio, prevedite in naložite v simulator ter opazujte kako se izvaja po korakih. Pri tem opazujte registre »Stack Pointer«, PORTD, DDRD in PIND ter konec podatkovnega spomina.