# Uporaba mikrokontrolerjev

## Namen vaje

Spoznati osnove delovanja mikrokontrolerjev: principi programiranja, zbirni jezik, osnovni digitalni vhod/izhod, prekinitve. Zapisali bomo nekaj predstavitvenih programov v mikrokontrolerjev spomin in preizkusili njihovo delovanje na primeru Atmelovega ATMEGA328p.

## Mikrokontrolerji

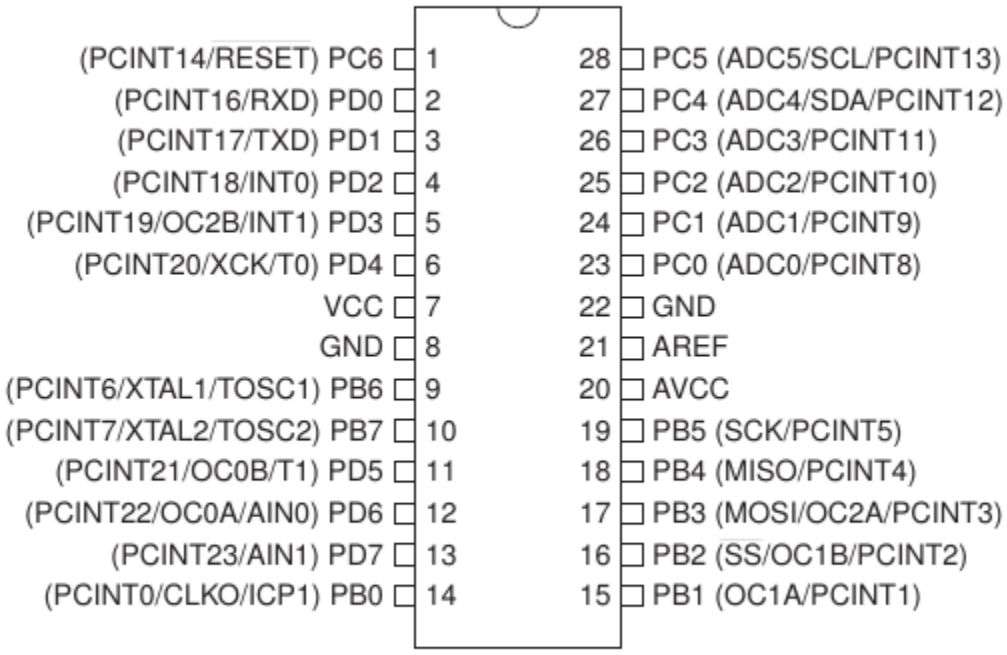
Mikrokontrolerji so vezja, ki opravljajo odločitve v našem električnem vezju. Njihova naloga je, da se ustrezno odzivajo na dane signale, ki so lahko zunanjega izvora ali notranjega. Z njimi opravljamo vsa dela, za katera bi bila osnovna konstrukcija iz enostavnejših komponent prezahtevna ali predraga.

Od klasičnih mikroprocesorjev se razlikujejo v tem, da imajo integriran trajni in delovni spomin. Tipično vsebujejo tudi notranji oscilator, ki lahko daje uro centralni procesni enoti. Poleg naštetega pa lahko vsebujejo tudi časovnik (timer), analogno-digitalni pretvornik (ADC), komunikacijske module (največkrat UART in I2C, lahko pa tudi CAN, USB …), pulzno modulacijske izhode (PWM) … Klasični mikroprocesorji vsega tega ne vsebujejo. Je pa res, da je v zadnjem času s pojavom integriranih sistemov SOC (system on chip), ki so se uveljavili predvsem v manjših pametnih napravah (telefoni, tablični računalniki, nekateri manjši računalniki …), ta meja nekoliko zabrisana. Nič hudega, če vam kakšen od omenjenih pojmov še ni domač; pomembnejše bomo podrobneje spoznali v vaji.

## Strojni jezik

Mikrokontroler deluje tako, da pobira zaporedje bytov iz svojega programskega pomnilnika. Vsak byte pomeni neki ukaz, ki ga nato mikrokontroler izvrši. Delovanje mikrokontrolerjev si bomo ogledali na konkretnem primeru Atmelovega ATMEGA328p, ki sodi v družino AVR mikrokontrolerjev (poleg PIC sodi med popularnejše arhitekture).

AVR arhitektura ima tako kot večina mikrokontrolerjev ločen programski in delovni spomin (takim sistemom pravimo Harvardska arhitektura) za razliko od računalnikov, kjer je ta spomin fizično isti. Program je trajno zapisan v flash pomnilniku (ATMEGA328p ga ima 32kB), medtem ko tekoči program uporablja SRAM pomnilnik (2kB v našem primeru). Čeprav se morda v časih, ko imajo računalniki 8GB delovnega spomina in 1TB SSD diska, zdijo omenjene kapacitete zelo nizke, v veliki večini primerov krepko presegajo zahteve za tipična opravila, za katera so sprogramirani. Tako konstrukcijo seveda narekuje narava mikrokontrolerja, saj ga sprogramiramo samo enkrat.




*Drawing 1: Postavitev signalov na mikrokontrolerju ATMEGA328p*

Poglejmo si na primer zaporedje bytov 0x01E005B8 (predpona 0x je v več programskih jezikih oznaka za šestnajstiški zapis števila). Takšno zaporedje bytov bo postavilo vse PB izhode (glej sliko 1) na nizek logični nivo razen izhoda PB0, ki ga bo postavilo na visok logični nivo.

Pisanje programov z zaporedjem številk je zelo zahtevno opravilo. Poleg tega je človeku neberljivo in razvozlavanje zaporedja zahteva precejšen napor. Zato si pomagamo z zbirnikom (assembler). Zbirnik (tudi zbirni jezik) je zbirka mnemonik za vsak ukaz, ki ga procesor razume. Tako se npr. sledeči program

```
LDI R0,1
OUT 5,R0
```

prevede na isto zaporedje bytov (zaporedje generira računalniški program iz izvorne kode) kot prej 0x01E005B8. Kljub temu je zgornjo kodo lažje razumeti. Ukaz LDI je kratica za *Load Immediate*. To pomeni, da bomo dani register R0 napolnili z vrednostjo 1. Registre lahko razumemo kot delovne spremenljivke procesorja. Drugi ukaz OUT pa pomeni, da bomo na dano lokacijo 5 (Atmel tudi te lokacije imenuje registri, zato lahko prihaja do občasne zmede) zapisali vrednost registra R0. Da bi polno razumeli ta ukaz, moramo vedeti, kaj se nahaja na tej lokaciji. V našem primeru se tam nahaja register PORTB, ki definira, kakšne vrednosti bomo imeli na izhodih od PB0 do PB7.

Kdor je vajen programiranja, se bo morda vprašal, zakaj nismo direktno na lokacijo 5

vpisali vrednosti 1 z ukazom

```
OUT 5,1 .
```

Izkaže se, da takega ukaza ni v naboru. Vidimo torej, da smo za razliko od klasičnih programov pri zbirniku omejeni na nabor argumentov, ki jih lahko podamo posameznemu ukazu.
Sicer pa je tudi programiranje v zbirniku precej nehvaležno opravilo (ki pa je včasih nujno potrebno), zato si raje pomagamo s kakšnim višjim programskim jezikom. Za mikrokontrolerje največkrat uporabimo jezik C (navadni, ne kakšne njegove izpeljanke kot sta C++ in C#). Tako bi se isto zaporedje v C zapisalo

```
PORTB=1; .
```

## Model semaforja

Za začetek bomo naš mikrokontroler povezali tako, da bo deloval kot semafor. Rdečo luč bomo povezali na PB0, rumeno na PB1 in zeleno na PB2. Stanje na izhodu definiramo z registrom PORTB. Register DDRB pa nam določa, ali je določena povezava vhod ali izhod. V mikorkontroler bomo zapisali sledeči program:

```
/* PORTB, DDRB in ostali so definirani tu */
#include <avr/io.h>

/* potrebujemo za _delay_ms */
#include <util/delay.h>

int main() {
        /* n bo štel sekunde od za?etka cikla */
        int n;
        /* PB2, PB1 in PB0 bodo uporabljeni kot izhod, ostali PB kot vhod */
        DDRB=(1<<PB2)|(1<<PB1)|(1<<PB0);
        PORTB=0;
        /* Poganjamo v nedogled */
        while (1) {
                /* cikel naj traja 2 minuti */
                for (n=0; n<120; n++) {
                        /* Na za?etku prižgemo rde?o lu?*/
                        /*(in izklopimo rumeno iz prejšnjega cikla) */
                        if (n==0) {
                                PORTB&=~(1<<PB1);
                                PORTB|=1<<PB0;
                        }
```

```
                        /* po eni minuti še rumeno */
                        if (n==60) PORTB|=1<<PB1;
                        /* po dveh sekundah izklopimo rde?o in rumeno in */
                        /* vklopimo zeleno */
                        if (n==62) {
                                PORTB&=~((1<<PB1)|(1<<PB0));
                                PORTB|=1<<PB2;
                        }
                        /* zadnjih 5s vklopimo rumeno */
                        if (n==115) {
                                PORTB&=~(1<<PB2);
                                PORTB|=1<<PB1;
                        }
                        /* po?akamo 1s */
                        _delay_ms(1000);
                }
        }
}
```

## Tipka za interventna vozila

Naslednji program bo služil kot prikaz branja vhoda. Na PB3 bomo priklopili tipko za interventna vozila, ki bo na semaforju nemudoma prižgala rdečo luč in omogočila prehod interventnim vozilom. Nekako intuitivno bi bilo pričakovati, da bomo stanje na PB3 prebrali z registra PORTB. Vendar so se Atmelovi inženirji odločili drugače. Vrednosti vhodov beremo na registru PINB, PORTB pa služi za morebitno vezavo vhodov z visokim nivojem preko upornika (pull-up resistor). Dodali bomo sledečo vrstico na začetek zanke

```
        /* ?e je pritisnjena tipka, za?ni cikel na novo (z rde?o) */
        if (!(PINB&(1<<PB3))) n=0;
```

poleg tega pa bomo PORTB inicializirali na

```
        PORTB=1<<PB3;
```

Preveri delovanje semaforja, poišči napako in jo odpravi. Opiši spremembo.

Koliko časa sveti zelena luč? Skrajšaj ta čas na 30s. Opiši spremembo.