Az AVR programozás alapjai

Előadja: Both Tamás

Fordító

- C nyelven programozunk
- Ehhez az AVR-GCC fordító áll rendelkezésre
- Ennek használatához a WinAVR-t kell telepíteni
- Teljes értékű C fordító, minden megengedett, ami a C-ben, legyen akármilyen bonyolult nyelvi szerkezet
- Sőt, van amiben többet is tud, pl: bináris számok – a kódba 0b000000 –t írva a 0b utáni részt binárisan értelmezi (ez regiszterek írásnál igen hasznos)

Egy program struktúrája

- 1) Szükséges inlklúdolások
- 2) Minták definiálása
- 3) Változók deklarációja
- 4) Főprogram
- 5) Interrupt függvények

Egy program struktúrája

- 1) Szükséges inlklúdolások
  - #include<avr/io.h>
  - #include<avr/interrupt.h>
  - #include<stdint.h>
- Az első kettő muszáj a lábak és a megszakítások használatához, a harmadik csak érdemes hogy ne kelljen szívni a c-beli számtípusokkal, így átláthatóbb a kód

<stdint.h> tartalma

- uint8_t – 8 bites szám előjel nélkül
- int8_t – 8 bites szám (előjeles)
- uint16_t – 16 bites szám előjel nélkül
- int16_t – 16 bites szám (előjeles)
- uint32_t – 32 bites szám előjel nélkül
- int32_t – 32 bites szám (előjeles)
- uint64_t – 64 bites szám előjel nélkül
- int64_t – 64 bites szám (előjeles)

- Általában elég az uint8_t és uint16_t

Egy program struktúrája

- 2) Minták definiálása (ha szükséges)
  - #define nulla 0b0010000
  - #define egy 0b0001000
  - #define ketto 0b0001001
- Akkor kellhet, ha egy portra pl. kijelzők vannak kötve, amire úgyis számokat írsz ki, és ezek a makrók használatával egyszerűsödik a kód, nem kell a programban mindig a biteket írogatni, csak hogy „nulla”, „egy”, stb

Egy program struktúrája

- 3) Változók deklarációja
  - volatile uint16_t elooszto=0;
  - volatile uint8_t ido[6]={0,0,0,0,1,0};
  - uint8_t szegmens=0;
- Akkor kell a „volatile”, ha a változót főprogramban és interruptban is használjuk, vagy több különböző interruptban, egyébként elhagyható. A változók kezdőértéket is kaphatnak

Egy program struktúrája

- 4) Főprogram:

```
int main(){
        <hardverek konfigurálása>
        <inicializálások>
        while(1){
                <végtelen ciklus>
        }
}
```

Főprogram

- Végtelen ciklus mindig muszáj, ebben végzi feladatait a mikrokontroller, mindig újra lefuttatva az ebben lévő utasítássort, ezt esetlegesen néha megszakítva
- A cikluson kívüli utasítások inicializálásra használhatók, csak egyszer futnak le, pl változók értékének megadása lehet ilyen, vagy tipikusan a belső hardverek beállításának megadása (változtatható később is)

Belső hardverek konfigurálása

- Belső hardverek lehetnek az AVR-ben (típusfüggően változik)
  - IO portok
  - Timer(ek)
  - AD konverter
  - Analóg komparátor
  - Kommunikációs protokollok (UART, SPI, I2C)
  - Stb…

Belső hardverek konfigurálása

- Ezeket be kell állítani, ha használni akarjuk (alapértelmezésben kikapcsoltak), meg kell adjuk a használat módját, paramétereit
- Ezt egy-egy, a hardverhez tartozó regiszterbe érték töltésével tehetjük meg.
- A regiszterek címzése egyszerű, nevüket úgy használhatjuk, mintha változók lennének

  Pl: TCCR2=0b00001101;

Belső hardverek konfigurálása

- A regiszterek neve szabványos, de adatlapból megtudható, ha nem ismered
- Leginkább használtak:
  - SREG – Általános célú regiszter
  - PORTx , DDRx – IO portokhoz
  - TCCRx, OCRx, TIMSK – Számlálókhoz
  - ADCSRA, ADMUX – AD konverterhez
  - UCSRB, UCSRC, UBRRH, UBRRL – UART-hoz

Belső hardverek konfigurálása

- Hogy az épp kellő regisztert mivel kell feltölteni, az adatlap megmondja
- Adatlap a barátod, hiába több 100 oldalas, el kell olvasni és folyamatosan böngészni
- Register Summary címke -> regiszter kiválasztása és odalapozás -> elolvasni a regiszter melyik bitjébe mit kell tölteni az adott működés eléréséhez

Adatlaphasználat okosan

- Nem kell végigrágni az elejétől a végéig
- A használatra kiválasztott hardverhez kapcsolódó részt érdemes elolvasni, ha még nem tetted ezt meg korábban
- Címkék hasznosak ehhez
- A későbbiek során elég a regiszterbe töltendő biteket megnézni
- Egyszer megérted egy hardver működését, az elég, nem fog elfelejtődni, ill. könnyű eleveníteni

Egy program struktúrája

- 4) Interrupt függvények
  - Interrupt = megszakítás, a főprogram ciklusán kívüli kódrészlet végrehajtása
  - AVR programozásban kiemelkedően fontos
  - Célja: kódrészlet lefutásának időzítése, pozícionálása
  - Nem random fut le, amikor odaér hozzá a ciklus, hanem pl. egy bizonyos idő letelte, egy bizonyos feladat befejeződése után

Interrupt függvények

- A kódrészletet, ami meghívódik megszakításkor, egy új függvénybe kell tenni a főprogram után
- A fv neve pl.: ISR(TIMER2_COMP_vect){
- ISR: jelzi, hogy ez egy interrupt fv, nem szabad int vagy void típusú függvénybe tenni
- A zárójelen belüli rész jelzi, hogy ez a függvény melyik interrupt meghívódásakor fusson le (több interrupt is használható egy programban)

Interrupt függvények

- A használandó interrupt nevekhez van egy táblázat, itt böngészhető:
- **http://www.nongnu.org/avr-libc/user-manual/group__avr__interrupts.html**

Interrupt függvények

- Ne féljünk interruptot használni, hadd jöjjön akár másodpercenként több száz is
- Korrekt időzítés csak így lehetséges
- Sok perifériát könnyebb így kezelni, megszakítással jelzi, hogy végzett / figyeljen rá a program

Egyebek

- Egyes feladatokhoz vannak letölthető, már megírt részletek, amik egyszerűsítik a munkánkat. Pl: LCD kezelés
- A kódban törekedni kell a minél kevesebb ismétlésre, a program ésszerűsítésére, mert kifutunk a tárhelyből
- Az átláthatóság érdekében használhatunk függvényeket, bár nem muszáj (void, int, értékátadás, minden mehet ami progon volt)

A továbbiakban…

- Néhány kész program bemutatása
- Ezeken keresztül a perifériák lehetséges funkcióinak megismerése