KANDÓ KÁLMÁN
VILLAMOSMÉRNÖKI KAR

Sándor Tamás

Programozás II.

ÓE-KVK 2125

Budapest, 2015.

Készítette:

Sándor Tamás, adjunktus, Óbudai Egyetem Kandó Kálmán Villamosmérnöki Kar, Műszertechnikai és Automatizálási Intézet

Bedő Sándor, villamosmérnök (BSC)

Mészáros Dániel, műszaki ügyintéző, Óbudai Egyetem Kandó Kálmán Villamosmérnöki Kar, Műszertechnikai és Automatizálási Intézet

Lektorálta:

dr. Illés Zoltán PhD, egyetemi docens, Eötvös Lóránd Tudományegyetem Informatikai Kar

1 Tartalom

2 Előszó

A Programozás II. tantárgy célja, hogy a korábban megismert programozási alapismereteket már valós, mikrokontrolleres környezetben fejleszteni tudják a hallgatók. A tárgy kereteiben először a mikrokontroller felépítésével ismerkedhetünk meg építve a Digitális technika tárgyakban elsajátított ismeretekre, majd az assembly nyelvű programozás alapjai kerülnek elsajátításra, és végül ismertetésre kerül ugyanannak a mikrokontrollernek a magas szintű programozási nyelven történő kódolása.

A jegyzetben ismertetésre kerül tárgyhoz kapcsolódó laborban (Programozás II. laboratórium) már 2009 óta használt T-bird nevű fejlesztő board, amely megfelelő segítséget nyújt az Atmel cég Atmega128 8 bites mikrokontrollerének megismeréséhez, illetve ehhez a fejlesztő boardhoz kapcsolódó kiegészítő boardon található perifériák megismeréséhez (hétszegmenses kijelző, billentyűzetmátrix, 4x16 karateres LCD kijelző, háromszínű LED, analóg hőmérséklet érzékelő, USB, RS485).

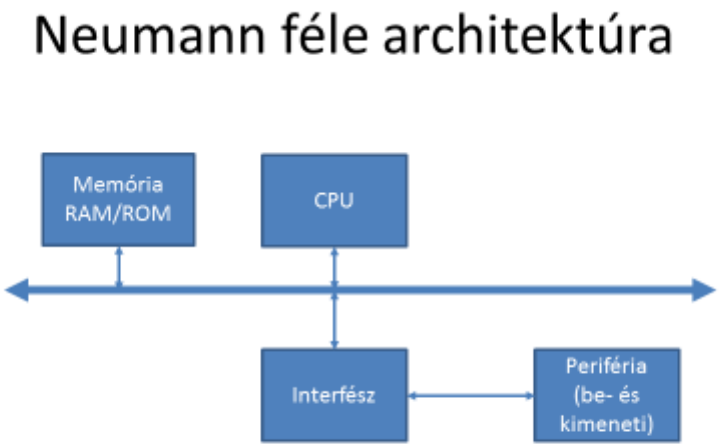
Az eszköz természetesen amellett, hogy a mikrokontroller programozás hatékony megismerését segíti természetesen alkalmas későbbi tárgyak (Beágyazott rendszerek, Projekt I., Projekt II. tárgyak, Információs rendszerek, Beágyazott rendszerek II.) önálló feladatainak megvalósításához, hiszen a önműködő robotoktól, az intelligens kaspón keresztül a beszélő T-bird-ig sokféle alkalmazás készült felhasználásával, illetve a kiegészítő board felhasználásra került 32 bites mikrokontrollerek oktatásában, illetve FPGA-hoz (Nexys4) is illesztésre került.

Budapest, 2015.03.31.

Szerző

3 Számítógép architektúrák

3.1 Neumann architektúra

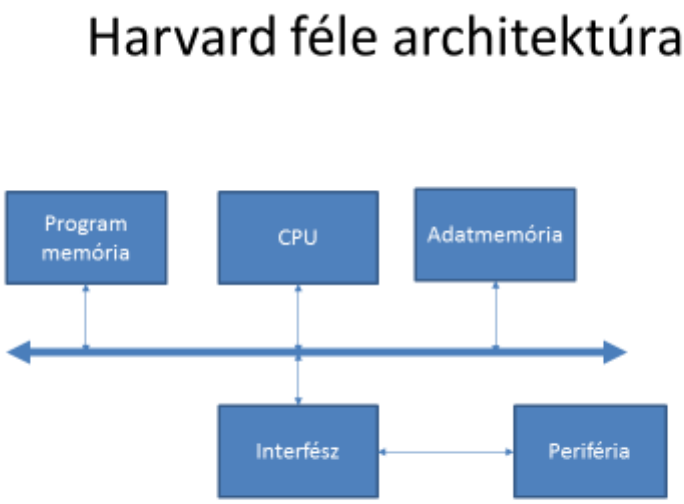


1. ábra Neumann architechtúra

A Neumann architektúra főbb részei a következők:

- Központi vezérlőegység (CPU, Central Processing Unit),
- Memóriák,
  - RAM (Random Access Memory),
  - ROM (Read Only Memory),
  - FLASH,
- Interfészek,
  - Általános felhasználású be- és kimentek (GPIO, General Purpose Input Output),
- Szabványos interfészek (I^2C, TWI, SPI,).

3.2 Harvard architektúra

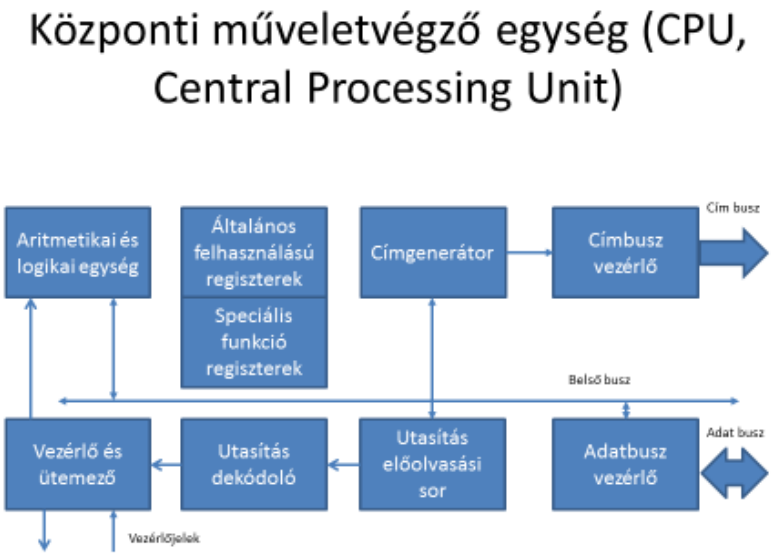


2. ábra Harvard architektúra

A Harvard architektúra főbb részei a következők:

- Központi vezérlőegység (CPU, Central Processing Unit),
- Memóriák,
    - RAM (Random Access Memory),
    - ROM (Read Only Memory),
    - FLASH,
- Interfészek,
    - Általános felhasználású be- és kimentek (GPIO, General Purpose Input Output),
- Szabványos interfészek (I^2C, TWI, SPI,).

3.3 CPU, központi műveletvégző egység

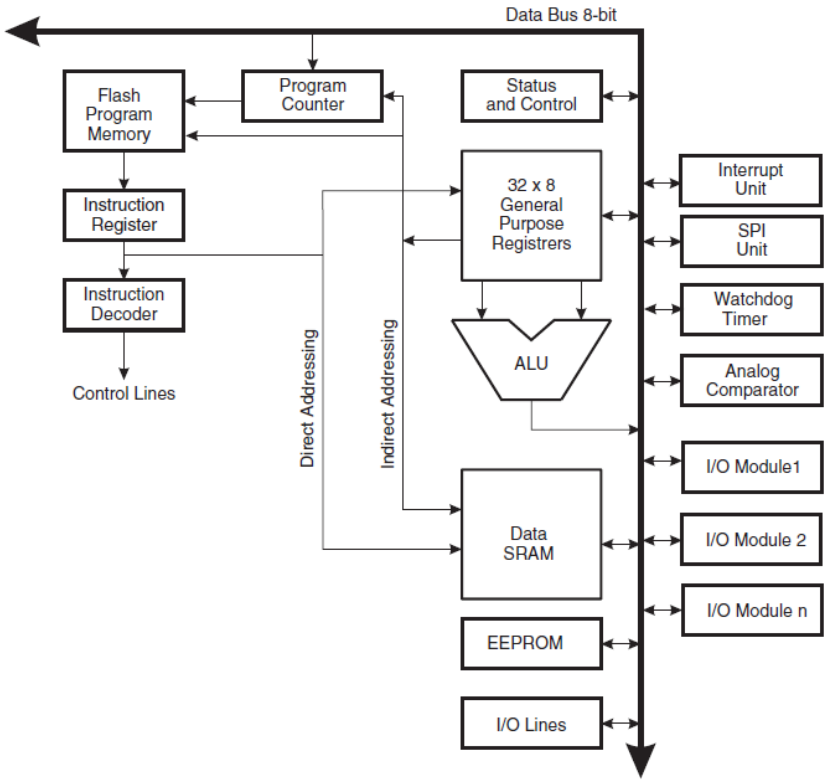


3. ábra Központi műveletvégző egység

A CPU főbb egységei a következők:

- ALU (Aritmetikai és logikai egység)
- Regiszterek
  - Általános felhasználású regiszterek
  - Speciális funkciójú regiszterek
- Utasítás dekódoló
- Vezérlő- és ütemező egység

4 8 bites Harvard architektúra



4. ábra Atmega128 CPU magja

Az Atmega128 mikrokontroller az alábbi főbb egységeket tartalmazza:

- ALU (Arithmetic and Logic Unit), aritmetikai és logikai egység, amelnyek a feladata az operanduszokkal történő műveletvégzés. A műveletvégzés bemeneti forrásainak ismertetése későbbiekben az egyes assambly utasítoknál kerül részletesebben ismertetésre.
- General Purpose Registers, általános felhasználású regiszterek (32 darab, r0-r31), amelyek 8 bites nagyságúak, de r26-r31-as regiszterek regiszterpárként is értelmezhetőek, X, Y és Z regiszterek index

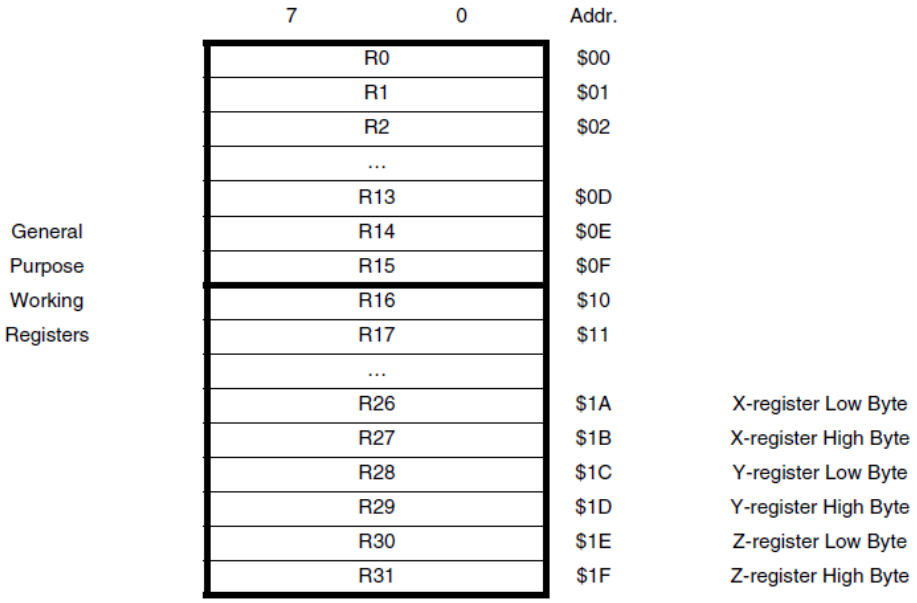
regiszterként is használatosak. Ezek a regiszterek a RAM terület első 32 bájtját adják.

- 4 kbyte DATA SRAM, amelyet 0x100 címtől már a felhasználó is szabadon programozhat.
- 128 kbyte Flash program memória, ahova a felhasználó a programjait feltöltheti.
-

4.1 Regiszterek

A regiszterek (belső tároló területek, amelyeket a mikrokontroller közvetlenül felhasználhat műveletvégzéshez) az SRAM területen találhatók.

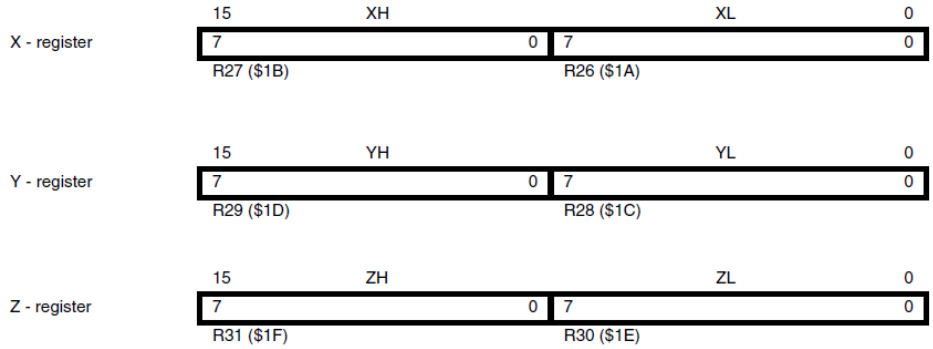
4.1.1 Általános felhasználású regiszterek (r0-r31)



5. ábra Általános felhasználású regiszterek

Általános felhasználású regiszterek:

- 32 darab 8 bites általános felhasználású regiszter, ebből:
    - 26 darab 8 bites regiszter általános
    - 3 db 16 bites indirekt címzés megvalósítására szolgáló indexregiszter:
        - X index regiszter, adatterület címzésére használatos indexregiszter.
        - Y index regiszter, adatterület címzésére használatos indexregiszter.
        - Z index regiszter, kódterület címzésére használatos indexregiszter (Flash program memória címzés).



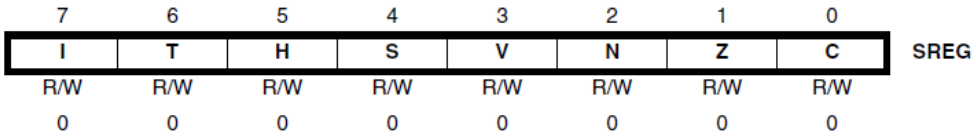
6. ábra Indexregiszterek

4.1.2 Speciális felhasználású regiszterek

- PC (Program Counter), Utasítás mutató (IP)
- SP (Stack Pointer), veremmutató, a verem következő szabad helyére mutat.
- SR (Status Register), a mikrokontroller állapotregisztere, általában a mikrokontroller állapotát jelzi műveletvégzést követően.
- MCU Control Register (IVCE, IVSEL).

4.1.3 Státusz regiszter (Status Register)

A státusz regiszter bitek a mikrokontroller állapotát jelzik műveletvégzést követően. Nézzük az egyes bitek funkcióját!



- I Global Interrupt Enable, általános megszakítás engedélyező bit, ha az értéke 0, akkor tiltott minden maszkolható megszakítás, ha az értéke 1, akkor általánosan engedélyezett a megszakítások használata. Ekkor még egyedileg az engedélyezéseket el kell végezni.
- T Bit Copy Storage (BLD vagy BST forrás vagy cél), a felhasználó által szabadon állítható bit.
- H Half Carry Flag (BCD), a 3. bitről a 4. bitre történő átvitel, amely bitnek a jelentősége BCD számokkal történő műveletvégzés esetén van.
- S Sign Bit ($S = N \oplus V$), előjel bit, amely előjeles számokkal történő műveletvégzés esetén használatos.
- V Overflow, túlcsordulás bit, előjeles számábrázolás esetén műveletvégzés után a számtartomány túllépését jelzi.
- N Negative, műveletvégzés után az eredmény MSB (Most Significant Bit) bitjének másolata.
- Z Zero, műveletvégzés után a bit értéke 1, ha az eredmény 0.
- C Carry, átvitel bit, előjel nélküli számábrázolás esetén a műveletvégzést követően a számtartomány túllépését jelzi.

4.1.4 Utasításmutató (IP, Instruction Pointer) vagy Program számláló (PC, Program Counter)

A következő végrehajtandó utasítás címét tartalmazza. Értékét az utasítás dekóder a beolvasott utasítás dekódolása után állítja. Továbbá állíthatja:

- Szubrutin hívás,
- Megszakítási rutin hívás,
- Elágazó utasítás.

4.1.5 Veremmutató (Stack Pointer)

A verem (STACK) az általános SRAM területen foglalt hely (max mérete a SRAM területe). A veremmutató segítségével lehet megcímezni a beírásnál, illetve a kiolvasásnál az adatokat. Stack Pointer (SP) egy 16 bites regiszter, amely kezelhetünk két datab 8 bites regiszterként is (SPH és SPL). A veremterület a RAM terület végétől kezdődik (RAMEND), és egészen a speciális felhasználású regiszterekig tarthat (0x100), ha a felhasználói programnak nincsen változók számára lefoglalt területe. Verem használatra szubrutin vagy megszakításhíváskor van szükség, így a veremmutató (SP) inicializálása nélkül ezek a programegységek nem is fognak működni, ha assembly programot készítünk.

Példa a verem inicializálására assembly nyelvű programozáskor:

```
ldi     r16,    high RAMEND
out     SPH,    r16
ldi     r16,    low RAMEND
out     SPL,    r16
```

Ekkor a veremmutató (SP) a RAM terület végére fog mutatni.

Verem (Stack) megoldások:

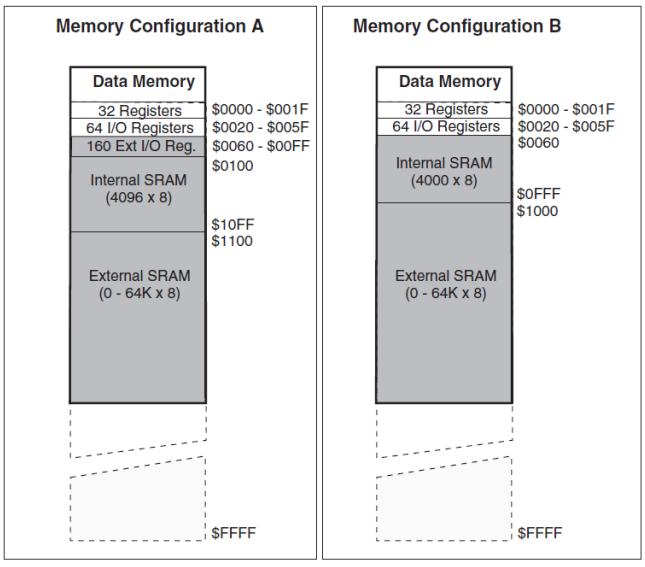
- Szoftveres verem:
  Általános SRAM területen foglalt hely (max mérete a SRAM területe), Atmel AVR
- Hardveres verem:
  Fixen kijelölt belső tároló terület, amely LIFO (Last In First Out) módon működik, PIC

4.2 Memória felépítése

4.2.1 Adatmemória, a RAM terület felépítése

Az adatmemória (SRAM) felépítése a 4.ábrán látható. A memória terület első 32 bájtján az általános hozzáférésű regiszterek (r0-r31) találhatók, majd ezt követi a 64 bájton keresztül az I/O regiszterek, és végül 160 bájton keresztül kiterjesztett I/O terület következik 0xFF címig. A felhasználói programok 0x100-tól kezdődhet, és 4kbyte nagyságú SRAM területet használhatják. Természetesen nem szabad elfelejtkezni a verem területről, amely SRAM terület végétől (RAMEND) építkezik visszafele.

Az egyes assembly utasítások különböző címtartományokban használhatók, nem szabad összekeverni a használatukat, mert a fordító nem fog jelezni szintaktikai hibát, de a program nem azt fogja végrehajtani, amit várnánk tőle.



7. ábra Adatmemória (SRAM) terület felépítése

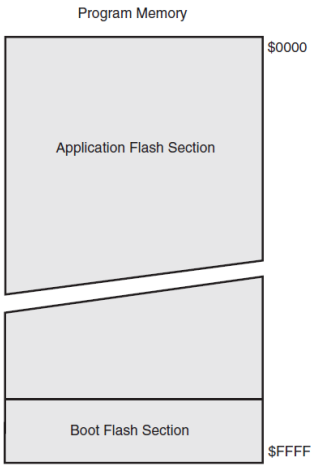
Az alábbiakban összefoglalásra kerül, hogy melyik címtartományban melyik adatmozgató utasítás használható:

- 0x00 – 0x1F közvetlen bitenkénti kezelés (SBI, CBI), vagy adatmozgató utasítás (MOV),
- 0x00 – 0x3F IN vagy OUT utasítással,
- 0x60 – 0xFF ST/STS/STD, LD/LDS/LDD.

4.2.2 Programmemória, a Flash terület felépítése

Az elkészített felhasználói programokat JTAG programozó vagy ISP programozó segítségével a programkód területre lehet letölteni. A Harvard architektúrából adódóan az adat és a programkód terület külön címtartományban található, így a programkód terület is a 0x0000 címtől kezdődik. Ennek a területnek az elején találhatjuk meg a megszakítás vektortáblát, amely a függelékben megadott táblázatban található megszakítást kiszolgáló rutinoknak a szabványos címeit tartalmazza.

Lehetőség van arra is, hogy a mikrokontroller bootloader üzemmódban induljon el, ekkor a Boot Flash Section (a bootloader üzemmódnak megfelelő) programterületen található kód indul el.



8. ábra Programmemória, a Flash terület felépítése

Ez a funkció nagyon jól használható programozó eszköz nélkül történő programfrissítésre, hiszen ha a Boot Flash Section területre olyan kód kerül

letöltésre (természetesen ekkor még alkalmazni kell programozó eszközt), amelyik például soros porton keresztül feltölti az új programot az Application Flash Section-be, akkor ezzel megoldhatjuk azt is, hogy egy távoli eszközre töltsünk le programot (tipikusan például mérésadatgyűjtő eszközök távoli firmware frissítése).

4.3 Portkezelés

4.3.1 I/O portok

Az AVR-ek meglehetősen sok és különféle bemeneti/kimenet portokkal rendelkeznek attól függően, hogy milyen típusú mikrokontrollert választunk. Esetünkben a Programozás II. laboratórium keretein belül az ATmega128 típusú mikrokontrollerrel találkozhatunk.

A laboron használt mikrokontroller 53 darab I/O lábbal rendelkezik, amelyeket 7 portra osztottak szét: PORTA, PORTB, PORTC, PORTD, PORTE, PORTF, PORTG[0-4].

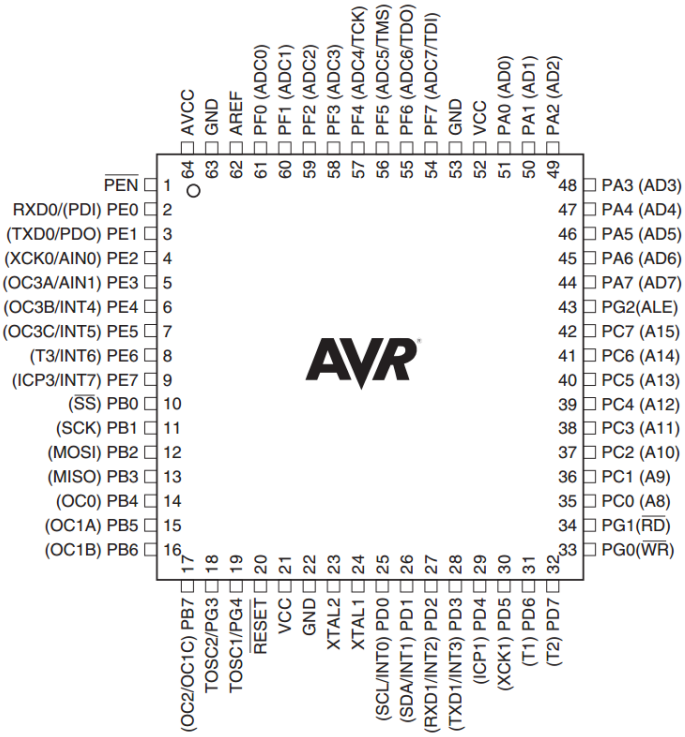
Ezen felül a legfontosabb lábak a következőek:

Láb neve	Megnevezése
VCC	pozitív tápfeszültség
GND	föld
XTAL1-2	külső kvarc bemenet
RESET	reset bemenet
AREF	ADC referencia feszültsége
AVCC	analóg rész pozitív feszültsége
AGND	analóg rész föld

Fontos megjegyezni, hogy az AVR mikrokontroller kimenete portonként maximálisan 125mA-t képes kiadni magából, és a bemeneti feszültség nem haladhatja meg a tápfeszültséget. Ezen értékek figyelembe vétele nélkül a mikrokontroller károsodást szenvedhet, tönkremehet.

4.3.2 I/O lábak funkciói

Az AVR lábainak funkcióit minden esetben megtaláljuk az adatlapban, de esetünkben az ATmega128 mikrokontrollert ebben a részben egy kicsit részletesebben megvizsgáljuk.



1. ábra Atmega 128 lábkiosztása

Mint a képen is látható a lábak mellett zárójelben találhatóak úgynevezett második funkciók. Nézzük is meg ezek jelentéseit:

Megnevezés	Funkció leírása
RXD*x* / TXD*x*	UART*x* fogadó és küldő lábai (bemenet, kimenet)

AIN*x*	ADC analóg bemenetei
INT*x*	külső megszakítás bemenetek
SCL / SDA	TWI busz adatvonalai
OC*x*	8 bites Timer/Counter bemenetek
OC*x*A/B/C	16 bites Timer/Counter bemenetek
TCK, TMS, TDI, TDO	JTAG programozó adatvonalai (ha JTAG programozóval programozzuk a mikrokontrollert, akkor azok a lábak nem használhatóak fel I/O lábként!)
MOSI, MISO, SCK, SS	SPI busz adatvonalai

A táblázatban említett funkciók használatát a későbbi fejezetekben részletesen taglalni fogjuk.

Vizsgáljuk meg, hogy az AVR I/O lábait hogyan tudjuk beállítani:

- bemenetnek / kimenetnek állítás

Először is fontos megemlíteni, hogy ha bármit is be szeretnénk állítani az AVR-en meg kell hívni az “avr/io.h” header fájlt, mert ez a fájl tartalmazza az AVR belső felépítésének megfelelően definiált kifejezéseit, így ennek köszönhetően nem memória címeket kell írni a kódunkba, hanem ezeknek megfelelő hivatkozásokat.

Az I/O lábak irányait a “DDRx” regiszterrel tudjuk állítani, ahol az “x” az adott port nevét jelenti. Nézzünk is egy példát a D port beállítására:

```
DDRD |= 0b11110000;
```

A D port (PORTD) felső 4 bitjét kimenetnek, az alsó 4-et pedig bemenetnek állítottuk be. Ebből rögtön kiderül, hogy ha az irány beállító regiszterbe ’1’-et írunk, akkor azt a port bitet kimenetként, ha ’0’-t írunk, akkor pedig bemenetként fogjuk tudni használni.

- kimenet beállításai

A láb logikai jelszintjét a “PORTx” regiszter segítségével lehet beállítani, ahol az “x” itt is a port nevét jelenti. Példa a D port beállítására:

```
PORTD |= 0b11110000;
```

A D port (PORTD) felső 4 lábán logikai ’1’, az alsó 4 lábon pedig logikai ’0’ jelszint fog megjelenni.

- bemenet további beállításai

Ha a láb bemenetnek van állítva, akkor bekapcsolható az AVR belső felhúzó ellenállása, amely olyan esetekben nagyon hasznos, ha egy gombnak hardveresen nem kötöttek be felhúzó ellenállást, és minden lenyomásnál jelentkezik a prell jelenség.

A beállítása a következő:

DDRx	PORTx	I/O	Belső felhúzó ellenállás
0	0	bemenet	nem
0	1	bemenet	igen

- bemenet vizsgálata

Sokszor adódik olyan helyzet, hogy vizsgálni kell AVR felhasználásával egy jel logikai értékét. Ekkor az adott láb bemenetnek állítását követően a "PINx" definiált értékből lehet kiolvasni, ahol az "x" a port nevét jelenti.

T-bird-ön a gombok a G porton kerültek elhelyezésre, így a példa ezeknek a gomboknak a használatára mutat példát:

```
1       #include <avr/io.h>
2
3       int main()
4       {
5            PORT_init();
6            while(1)
7            {
8                 if(PING & 0b00000001)
9                      {
10                          LED_out(0x10);
11                     }
12                else
13                     {
14                          LED_out(0x04);
15                     }
16           }
17      }
18
19      void LED_out(unsigned char szam)
20      {
21           PORTD = szam & 0xF0;
22           PORTB = (szam << 4);
23      }
24      void PORT_init(void)
25      {
26           DDRG |= 0b00000000;  //gombok bemenetek
27           DDRD |= 0b11110000;  //LED-ek kimenetek
28           DDRB |= 0b11110000;  //LED-ek kimenetek
29      }
```

Nézzük ennek az egyszerű programnak a leírását:

1. Általánosan használatos header fájl, az AVR definíciók használatához (pl. DDRB, PORTD, …)

3.-17. main függvény feje, illetve a függvény törzse

5. PORT_init() függvény meghívása, az irányregiszterek beállítása (DDRB, DDRD, DDRG)

6.-16. egy végtelen while ciklusban a PING0-án levő gomb lenyomását figyelve, ennek megfelelően a PORTD4-re, illetve PORTB6-ra kötött LED-ek aktiválódnak.

19.-23. LED_out függvény segítségével egy 0 és 255 közötti számot megjeleníthetünk a T-bird-ön található 8 db LED-en. A T-bird adatlapjából kiderül, hogy a LED0-LED3 a PORTB7-PORTB4-re, illetve a LED7-LED4 a PORTD7-PORTD4-re kerültek elhelyezésre.

24.-29. PORT_init függvény definíciós része, ahol a konkrét irányregiszter beállításokat találhatjuk (emlékeztetőül ’0’: bemeneti irány, ’1’: kimeneti irány).

4.4 Megszakítások

4.4.1 Fogalma, forrásai

- megszakítási esemény
- futó program felfüggesztése
- megszakítás kérés kiértékelése
- Engedélyezett megszakítás esetén:
- Megszakítás Vektor Táblából a megszakítás kiszolgáló rutin címe betöltődik a IP-be (vagy PC-be), a visszatérési cím pedig a verembe
- Megszakítási rutin lefutása után (reti-vel fejeződik be a rutin) IP-be visszatöltődik a veremből a visszatérési cím
  a felfüggesztett program futása folytatódik tovább

4.4.2 Megszakítás vektortábla

Az Atmega128 megszakítás vektor táblája a programkód terület 0x0000 címétől kezdődik, és a mellékletben található táblázatban levő címeken találhatók az egyes kiszolgáló rutinok címe, 2 bájtonként.

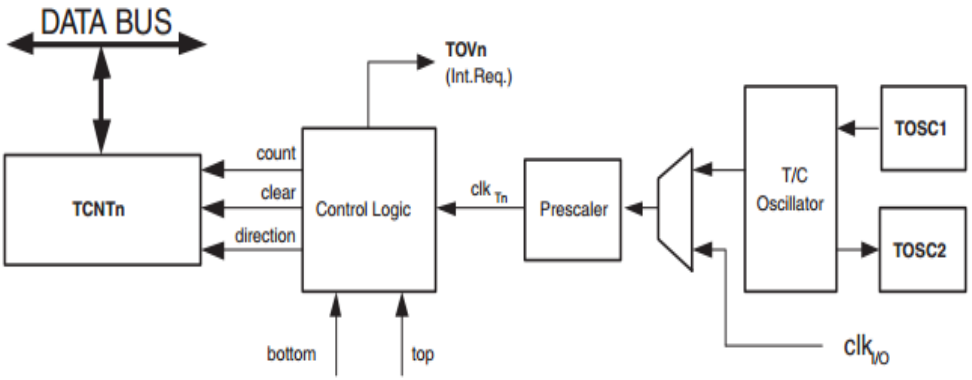
4.4.2.1 Időzítők / Számlálók

4.4.2.2 Timerek az ATmega128 -ban

4.4.2.3 Timer funkciók / beállítások

4.4.2.4 Timer/Interruptok

Minden programnak szüksége van időzítésre, hogy a különböző utasításokat a megfelelő időben hajtsa végre. Erre a célra használjuk a timer-eket (időzítő). A timer az alap órajelét az AVR kvarc bemenetére (XTAL1-2) kötött kvarcról, vagy a belső oszcillátoráról kaphatja. Ez az órajelforrás biztosítja a timer lépési frekvenciát beállító logika bemenetét, amit ez az alábbi képen is látható.



2. ábra Timer blokkvázlata

A Timer léptető logikája egy programozható frekvencia osztó áramkörön keresztül van összeköttetésben az órajel forrással, melyet majd a szoftver felhasználásával állíthatunk, és a Timer aktuális értéke a TCNTn regiszterben található, melynél az "n" a Timer számát jelenti.

A jegyzetben kettő időzítésről lesz szó amelynél a Timer-t használjuk fel:

- pontos időzítés

Az időzített végrehajtást megszakítások felhasználásával végezzük el. Nézzük, hogy mi is az az megszakítás. Interrupt (megszakítás) olyan művelet sorozat, amely a program futását megszakítja, a megszakítás kiszolgáló függvényben lévő utasításokat végrehajtja, majd visszatér a megszakított program futtatásához.

- időzített végrehajtás

A pontos időzítésekre és az időzített végrehajtásokra példa

```
Timer0Init();    //Timer beállítása
while(1)
        {
                if(TCNT > 100)
                {       //ha a timer értéke nagyobb, mint 100
                        LEDon();        //kapcsolja be a LED-eket
                }
                else
                {
                        LEDoff();       //kikapcsolja a ledeket
                }
        }
```

Fontos megemlíteni már a Timer-ek felhasználásánál a megszakítás fogalmát, mivel a Timer-ek is képesek megszakításokat generálni, és majd ezt a képességét felhasználva tudunk időzített végrehajtásokat elvégezni.

ATmega128 időzítői:

- 2 db 8 bites timer (Timer 0, Timer 2)
- 2 db 16 bites timer (Timer 1, Timer 3)

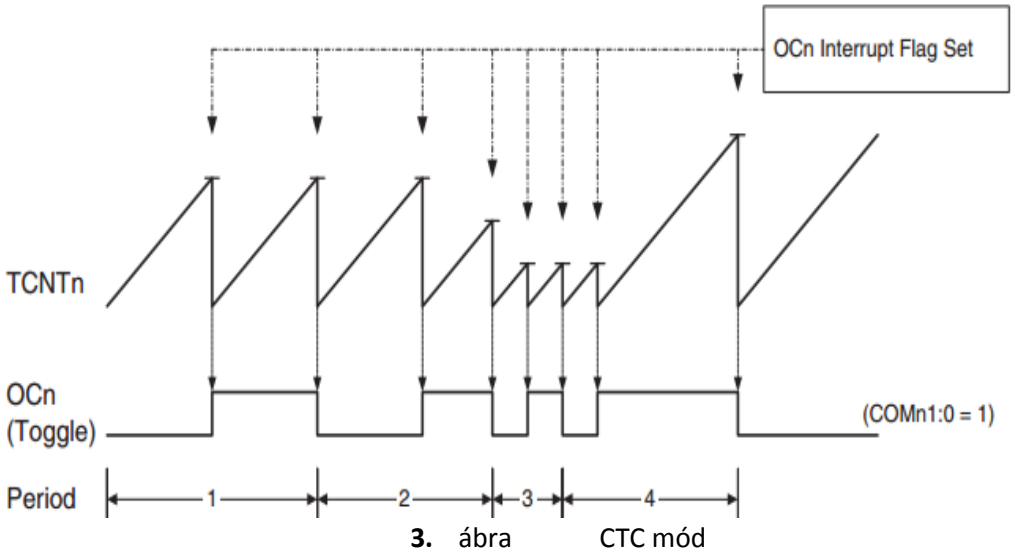
4.4.2.5 Timer0 funkciók

4.4.2.5.1 CTC mód (Clear Timer on Compare Match)

Mint fentebb említésre került, a Timer nem más, mint egy számláló, amelynek ismerjük a lépési frekvenciáját, és ezt kihasználva létre tudunk hozni nagyon pontos időzítéseket.

A mód fontossága, mint a nevéből is kiderül, törli a Timer tartalmát egy megadott érték elérésekor, melyet a szoftverből szabadon tudunk módosítani.

Egy kép a reprezentálásához az adatlapból:



3. ábra CTC mód

Látható, hogy az időzítő értéke folyamatosan nő az idő teltével, amíg el nem éri a komparálási értéket, melyet az "OCn" regiszterben állíthatunk be. Ekkor az értéke 0 lesz, és kezdi elölről a számolást.

4.4.2.5.2 Timer 0 beállítása

A Timer0 egy 8 bites időzítő, ami az jelenti, hogy az időzítés maximális értéke 255 lehet. Fontos megjegyezni, hogy az időzítő csak akkor kezdi el a számolást, ha annak nullánál nagyobb értékét állítottuk be (pl.: egy órajel osztást).

Példa 1: Készítsünk egy programot, mely 32ms-os periódussal növeli a LED-eken megjelenített számok értékét!

```
1
2   #include <avr/io.h>
3
4   void PORT_init(void);
5   void Timer0_init(void);
6   void LED_out(unsigned char szam);
7
8   int main()
9   {
10        unsigned char szamolo = 0;
11        PORT_init();
12        Timer0_init();
13
14        while(1)
15        {
16              if(TCNT >= 250) //32 ms
17              {
18                    szamolo++;
19                    LED_out(szamolo);
20              }
21        }
22  }
```

```
23  void PORT_init()
24  {
25       DDRB = 0xF0;
26       DDRD = 0xF0;
27  }
28  void Timer0_init()
29  {
30       TCCR0 = (1<<CS02)   //órajelosztás [1024]
31                 | (1<<CS01)
32                 | (1<<CS00)
33                 | (1<<WGM01)  //CTC mód
34                 | (1<<COM01); //komparáláskor törölje
35                                //a Timert
36            OCR0 = 250;    //Timer maximális értéke
37  }
38  void LED_out(unsigned char szam)
39  {
40       PORTD = szam & 0xF0;
41       PORTB = (szam << 4);
42  }
```

Magyarázat:

Az első a Timer0Init() függvény (28.-37. sorok), amelyben a Timer0-t inicializáljuk.

Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Az adatlapban is megtalálható a TCCR0 regiszter belső felépítése, amely pontosan leírja, hogy az egyes bitek beállításának hatását.

Vizsgáljuk meg pontosabban azokat a biteket, amelyek fontosak a laboratóriumi gyakorlat elsajátításához:

- WGM00, WGM01 - ezekkel a bitekkel tudjuk beállítani, hogy az időzítő milyen üzemmódban szeretnénk üzemeltetni

Table 52. Waveform Generation Mode Bit Description

Mode	WGM01[1] (CTC0)	WGM00[1] (PWM0)	Timer/Counter Mode of Operation	TOP	Update of OCR0 at	TOV0 Flag Set on
0	0	0	Normal	0xFF	Immediate	MAX
1	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	1	0	CTC	OCR0	Immediate	MAX
3	1	1	Fast PWM	0xFF	BOTTOM	MAX

Mint az adatlapban is látható, a WGM01 bit beállításával állíthatjuk be a Timer0-t CTC módba.

```
TCCR0 = (1<<WGM01);
```

- COM00, COM01 - Különböző módok további beállításai érhetőek el vele. Esetünkben a CTC mód beállításai.

Table 53. Compare Output Mode, non-PWM Mode

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Toggle OC0 on compare match
1	0	Clear OC0 on compare match
1	1	Set OC0 on compare match

A COM01 beállításával érhetjük el azt, hogy a Timer értéke nullázódjon a komparálási szint elérésekor, vagyis amikor eléri az OCR0 értékét.

```
TCCR0 = (1<<COM01);
```

- CS01, CS02, CS03 - A Timer belső frekvencia osztóját tudjuk állítani velük, vagyis a Timer lépési frekvenciáját

Table 56. Clock Select Bit Description

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	clk_{T0S}/(No prescaling)
0	1	0	clk_{T0S}/8 (From prescaler)
0	1	1	clk_{T0S}/32 (From prescaler)
1	0	0	clk_{T0S}/64 (From prescaler)
1	0	1	clk_{T0S}/128 (From prescaler)
1	1	0	clk_{T0S}/256 (From prescaler)
1	1	1	clk_{T0S}/1024 (From prescaler)

A CS0n bitek beállításával jelen esetben egy 1024-es osztást állítottunk be a Timer0-nak, ami a következőt jelenti:

Tegyük fel, hogy az AVR 8MHz-ről működik. Akkor az 1024-es osztással 7812,5Hz-et kapunk, tehát a Timer lépési frekvenciáját kapjuk meg. Ezt időre átszámolva:

$$T=\frac{1}{f} = \frac{1}{7812{,}5Hz} = 0{,}000128\ s\ =\ 128\mu s$$

vagyis a timer 1 lépést 128µs alatt tesz meg. Ami 255 lépésre átszámolva vagyis a túlcsordulásához szükséges idő: 32,64 ms.

Jelen esetben nekünk csak 32ms -ra van szükségünk, ezért be kell állítani egy komparálási szintet az OCR0 regiszterben, ami 250. Ugyanis 250*128µs az 32ms-al egyenlő. Ezen felül a CTC mód használata szükséges, mivel gondoljunk csak bele, hogy ha nem nulláznánk a timert a komparálási szinten, akkor az első komparálás igaz jó lenne, de az utána lévőekhez hozzáadódna még 5 * 128µs ami az az idő ameddig eléri a maximális értékét ami 255.

Példa 2: Készítsünk egy programot, amely 800 ms -os periódussal meghív egy függvényt amely növeli a LED-eken megjelenített szám értékét.

Komplexebb időzítési feladatok elvégzéséhez már interruptokat használunk. A Timer-ek interrupt meghívásának 2 módja van AVR esetén:

- Owerflow interrupt
- Compare Match interrupt

Overflow interrupt (Túlcsordulás interrupt)

Akkor hívódik meg, ha a Timer elérte a maximális értékét és túlcsordul. 8 bites timer esetén 255, 16 bites timer esetén 65535.

Compare match interrupt

Komparálási szint elérésekor hívódik meg az interrupt. A komparálási szintet az előzőekben bemutatott OCRn regiszter értékével lehet állítani.

A második fontos dolog, egy ilyen feladat megtervezésénél a timer kiválasztása. Ugyan is ha 800ms-os végrehajtást szeretnénk létrehozni ahhoz már a 8 bites timer nem elég. Nézzük is meg egy egyszerű számítással levezetve:

- 8 bites timer (8MHz-es órajel)

  A timer maximális előosztója 1024, és a maximális értéke 255.

  $$\frac{1}{8000000/1024} * 255 = 0{,}03264\ s \Rightarrow 32{,}64\ ms$$

  Tehát látható, hogy egy 8 bites Timer felhasználásával maximum 32,64ms-os interrupt meghívást tudunk létrehozni.

- 16 bites timer (8MHz-es órajel)

  A timer maximális előosztója itt is 1024 és a maximális értéke 65535.

$$\frac{1}{8000000/1024} * 65535 \ = 8{,}38\ s$$

Látható, hogy ezzel a timerrel már több mint 8 másodperces interrupt meghívásokat is létre tudunk hozni.

Következzen a feladat megoldása Timer 1 felhasználásával:

```
#include <avr/io.h>
#include <avr/interrupt.h>

void LED_out(unsigned char szam) {
     PORTD = szam & 0xF0;
     PORTB = (szam << 4);
}

void Timer1Init() {
     TCCR1A = (1<<WGM10) //CTC mód
          | (1<<COM1A1); //Töröl komparálási szinten
     TCCR1B = (1<<WGM12) //CTC mód
          | (1<<CS10)    //előosztás [1024]
          | (1<<CS12);
     TCCR1C = 0;
     OCR1A = 6250;       //1 lépés: 128□s -> 800ms
     TIMSK |= OCIE1A;    //Komparálási interrupt
engedélyezése

     sei();              //Globális interruptok
engedélyezése
}

unsigned char szamlalo = 0;

int main() {
     DDRD = 0xF0;   //LEDek beállítása
     DDRB = 0xF0;

     Timer1Init();  //Timer beállítása

     while(1) {
          LED_out(szamlalo);  //szamlalo erteke a
LED-eken
     }
}

ISR(TIMER1_COMPA_vect) {      //Timer1 komparálási
interrupt
```

```
        szamlalo++;
    }
```

Magyarázat:
Először is pár szó a 16 bites Timerről. Alapjában véve nagyon hasonlít a 8 bites Timerre, annyi különbséggel, hogy több funkcióval rendelkezik és a beállításait több regiszterre bontották szét.

Az “avr/io.h” header fájl mellett most már használnunk kell az “avr/interrupt.h” header fájlt is, ami az AVR mikrokontroller esetében az interrupt(megszakítások) kezelését tartalmazza.

Bit	7	6	5	4	3	2	1	0	
	COM1A1	COM1A0	COM1B1	COM1B0	COM1C1	COM1C0	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

A TCCR1A regiszter felépítése a következő:

- COM1x1, COM1x0 : Mint ahogy a 8 bites esetén a timer különböző funkcióit állíthatjuk be velük
- WGM10, WGM11 : A timer futási módjait állíthatjuk be vele, mint például az előzőekben ismertetett CTC módot.

Table 58. Compare Output Mode, non-PWM

COMnA1/COMnB1/ COMnC1	COMnA0/COMnB0/ COMnC0	Description
0	0	Normal port operation, OCnA/OCnB/OCnC disconnected.
0	1	Toggle OCnA/OCnB/OCnC on compare match.
1	0	Clear OCnA/OCnB/OCnC on compare match (set output to low level).
1	1	Set OCnA/OCnB/OCnC on compare match (set output to high level).

Esetünkben a kiszínezett sor a jelentős, mely beállításához a COM1A1-es regisztert kell beállítani a TCCR1A-ban. Ezzel azt érjük el, hogy ha a timer elérte a megadott komparálási szintet, akkor kinullázza magát.

Bit	7	6	5	4	3	2	1	0	
	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

A TCCR1B regiszter felépítése a következő:

- WGM12, WGM13 : Ezekkel és a TCCR1A regiszterben lévő bitekkel tudjuk beállítani a timert különböző funkciókba.
- CS10,CS11,CS12 : Ezekkel a bitekkel tudjuk beállítani a timer előosztását, melye a timer0 beállításánál részletesebben elmagyarázásra került.

Table 61. Waveform Generation Mode Bit Description

Mode	WGMn3	WGMn2 (CTCn)	WGMn1 (PWMn1)	WGMn0 (PWMn0)	Timer/Counter Mode of Operation[1]	TOP	Update of OCRnx at	TOVn Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCRnA	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICRn	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCRnA	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICRn	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCRnA	TOP	BOTTOM
12	1	1	0	0	CTC	ICRn	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICRn	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCRnA	BOTTOM	TOP

Mint a táblázatban látható, a timer CTC módjának a beállításához a WGM11-t kell beállítani, hogy a komparálási szintje az OCR1A legyen.

Table 62. Clock Select Bit Description

CSn2	CSn1	CSn0	Description
0	0	0	No clock source. (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/1 (No prescaling
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on Tn pin. Clock on falling edge
1	1	1	External clock source on Tn pin. Clock on rising edge

A timer előosztását 1024-re szeretnénk beállítani, ezért a CS10 és a CS12 bitet kell beállítani a TCCR1B regiszterben.
Nézzünk is rá egy példát:

A mikrokontrollerre kötött órajelforrás az 8MHz.
Tehát az F_CPU = 8000000
Az előosztás: 1024

A Timer lépési ideje:

$$\frac{8\,000\,000\ Hz}{1024} = 7812{,}5\ Hz$$

$$\frac{1}{7812{,}5\ Hz} = 0{,}000128\ s = 128\mu s$$

Tehát $128\mu s$ szükséges ahhoz hogy a timer lépjen egyet.
Ahhoz, hogy a feladatban megfogalmazott 800 ms-os időzítést létre tudjuk hozni ki kell számolni az OCR1A értékét, amely a komparálási szint.

$$\frac{800\ ms}{128\ \mu s} = 6250$$

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Most már csak azt kell beállítani, hogy ha a timer értéke elérte a komparálási szintet, akkor generáljon egy interruptot. Ezt a TIMSK regiszterben állíthatjuk be.

Számunkra most csak az OCIE1A és a TOIE1 bit magyarázata a fontos.

Általában a Timerek kettő fajta interruptot tudnak generálni:

- Overflow interrupt: amikor a timer elérte a maximális értékét és úm. túlcsordul
- Compare match interrupt: amikor beállítunk egy komparálási értéket, és amikor ezt eléri a timer, akkor generál egy interruptot

Fontos, hogy az inerruptok megnevezéseit az “iom128.h” fileban találhatjuk, melyet az AVR Studio egy fordítás után automatikusan legenerál a “Dependencies” mappába.

Egyszerre tud Overflow és Compare match interruptot is generálni.

4.4.2.6 PWM

4.4.2.7 Soros portkezelés

Az RS-232 interfészt a CCITT nemzetközi távközlési bizottság határozta meg a soros adatátviteli interfész elektromos jellemzőit és feladatait. Ezt az adatátvitelt először számítógépek közötti kommunikációra használták, de a későbbiekben a mikrokontrollerek elterjedésének köszönhetően már minden ilyen nagyobb chip

tartalmaz legalább egy soros interfészt, melyet UART[1] vagy USART[2] –nak neveznek. A soros kommunikáció során az adatcsomagok egy start bittel kezdődnek, amely a csomag kezdetét jelzik, majd ezt követi a 7-9 -ig terjedő adatbit, amelyet a mikrokontrollereknél külön be tudunk állítani, majd következik egy paritás bit, és az üzenet egy stop bit zárja le. Az adatátviteli sebességét baudrate –ben határozzuk meg, amely a másodpercenként átvitt bitek számát jelenti. Ezen kívül lehetőségünk nyílik beállítani a stopbitek számát, melynek akkor van jelentősége, hogy ha az eszközök feldolgozási teljesítménye nem elegendő, akkor van még egy órajel ideje feldolgozni az adatot. A mikrokontrollerekben az UART általában duplán bufferelt, amely azt jelenti, hogy a fogadott adatot rögtön áttölti egy másik bufferbe és majd onnan történik az adat feldolgozása.

4.4.2.8 SPI

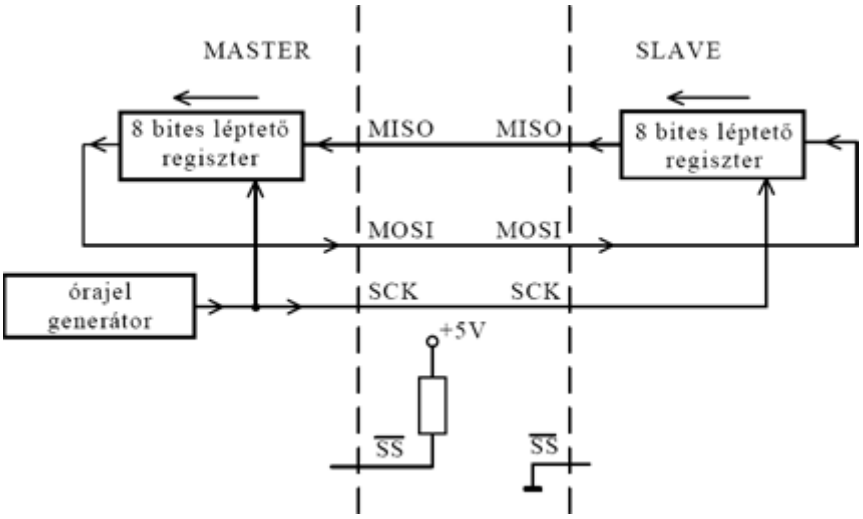
Az SPI buszt a Motorolla fejlesztette ki, amely egy full duplex protokoll. Mint ahogy a nevéből is adódik ez egy soros szinkron interfész. Az SPI kommunikáció is a master- slave kapcsolatot alakítja ki, annyi különbséggel az I^2C buszhoz képest, hogy itt a címzés nem az adat buszon történik, hanem minden slave eszköz rendelkezik egy ún. „[3]CS” lábbal, amely állításával tudja megcímezni az eszközt a master. A megcímzett eszköz felől ezt a lábat „[4]SS”- nek jelölik a gyártók. Minden mikrokontroller rendelkezik hardveresen chip select lábakkal, de ezeket szoftveresen a felhasználó is tudja állítani. Az SPI kommunikációnál a szerepek hardveresen kötöttek, így a master – slave szerep felcserélése nem lehetséges (egy mikrokontroller és egy külső periféria között). Viszont ha kettő

[1] Universal asynchronous Receiver/transmitter
[2] (Universal Synchronous Asynchronous Receiver/Transmitter
[3] CS - Chip Select
[4] SS - Slave Select

darab mikrokontroller kommunikál egymással, akkor ezek egy újrakonfigurálással könnyedén megoldhatók. Magához a kommunikációhoz 4 darab vezetékre van szükség:

- MOSI (Master Output Slave Input)
- MISO (Master Input Slave Output)
- SCK (Serial Clock)
- CS (Chip Select)

A kommunikáció szinkronizálása az SCK vezetéken történő órajelre történik. És a MOSI és a MISO lábakon a master és a slave eszköz úm. adatot cserél bitenként (mindig a master generálja az órajelet). Ezt a kommunikációt úgy is elképzelhetjük, mint kettő darab shift regisztert, amelyek az órajelre bitenként adatot cserélnek. Az 13.ábra a MOSI és a MISO kapcsolatát, és a busz bekötését szemlélteti.



1. ábra –SPI kommunikáció összeköttetés

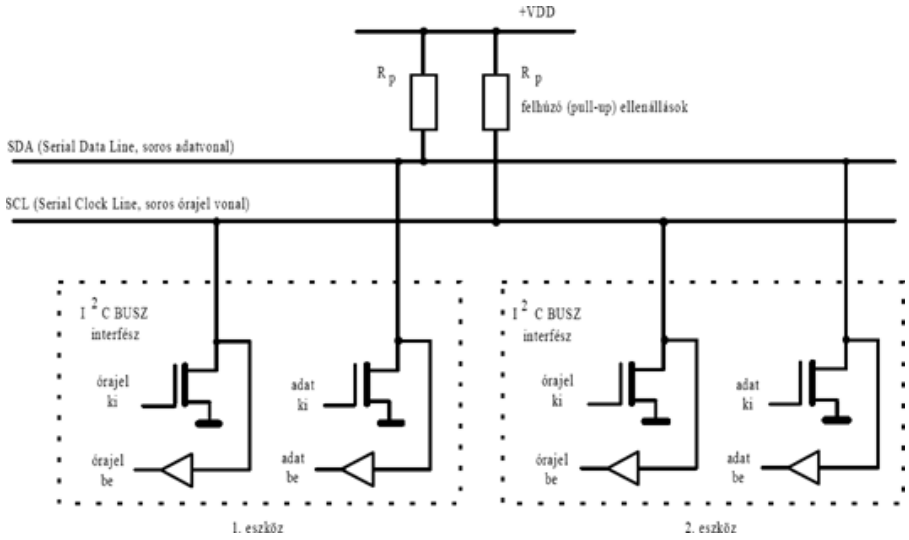
Ha a mikrokontrollerrel szeretnénk egy SPI kommunikációt kialakítani, akkor lehetőségünk adódik, beállítani az órajel fázisát és polaritását, melynek köszönhetően különféle megoldású soros elemek is összekapcsolhatók az SPI

rendszerrel. Ezen kívül több beállítási mód is létezik, hogy az adattranszfer az órajel mely „részénél” történjen meg. Lehetőségünk van fel- és lefutó élre is beállítani, ún. alap és fordított polaritásnál is.

Az SPI buszt általában 2MHz- ig használják, de pl.: a Xicor cég X2565 soros adatkezelésű EEPROM IC-je, amely SPI jelleggel kezelhető, akár az 5MHz es működési frekvenciát is elérheti.

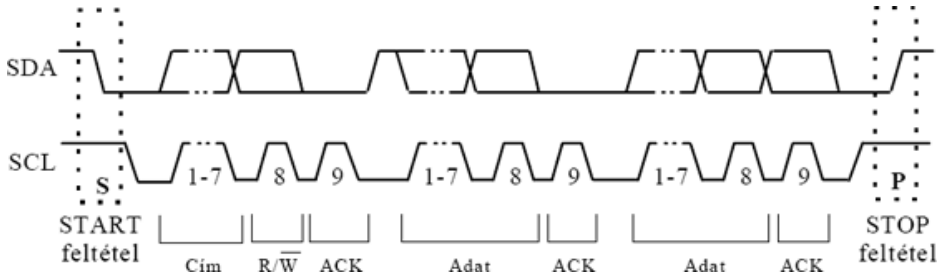
4.4.2.9 TWI

Története: Az 1980- as évek elején a Philips fejlesztette ki abból a célból, hogy a Televíziójukban a központi egység minél kevesebb vezetéken tudjon kommunikálni a perifériákkal. Az I^2C egy soros szinkron adatátviteli rendszer. A kommunikáció két vezetéken történik: SDA (Serial Data Line) , SCL (Serial Clock Line). Az SDA vonalon történik az adatátvitel az SCL vonal pedig szolgáltatja az adatátvitel ütemezéséhez szükséges órajelet, a start és a stop bitet. A buszra csatlakoztatott eszközök a két vezetékre nyitott draines ill. nyitott kollektoros kimenettel csatlakoznak (11.ábra), ezért fontos, hogy az SDA és az SCL vezetéket is általánosan 2k2 – 4k7 –es ellenállásokkal pozitív tápra húzzuk. Az ellenállások értéke annak függvénye, hogy milyen sebességen kommunikálnak az eszközök és hogy milyen a kapacitása a vezetékeknek. Fontos megjegyezni, hogy a buszon lévő eszközök külön címekkel rendelkeznek.



2. ábra – Open kollektoros csatlakozások

A kommunikáció folyamata: Az adat küldést mindig a master kezdeményezi úgy, hogy amíg az órajel magas állapotában nullára változtatja az SDA- n lévő értéket. Ez jelenti a start bit- et. Majd ezt követi az eszköz címe, amely általában 7 bit, mivel a 8.bit a R/W bit, amely azt jelzi, hogy a master olvasni vagy írni akar a slave eszközről. Ha ez megtörtént, akkor a megcímzett slave eszköz visszaküld egy ACK bit- et mellyel visszaigazolja a vételt. Ezután megkezdődik az adatbitek átküldése, fogadása melyeket mindig egy ACK bit zár le az adatellenőrzés miatt. Majd ha befejeződött az adatátvitel, akkor a master egy magas órajel állapotban magas állapotba állítja az SDA vonalon lévő bitet, ezzel jelezvén az adatátvitel végét. A 12.ábra ezt szemlélteti.



3. ábra –I²C kommunikáció folyamata

Tehát összefoglalva: az I²C adatátvitel két vezetéken történik (SDA, SCL), minden eszköz saját címmel rendelkezik, multi master- es, a busz időzítését mindig a master végzi, a maximális működési frekvenciája: -eredetileg 100kb/sec, –fast módban 3.2Mb/sec, –ultra fast módban 5Mb/sec

4.5 JTAG

5 T-bird3

5.1 Hardveres felépítése

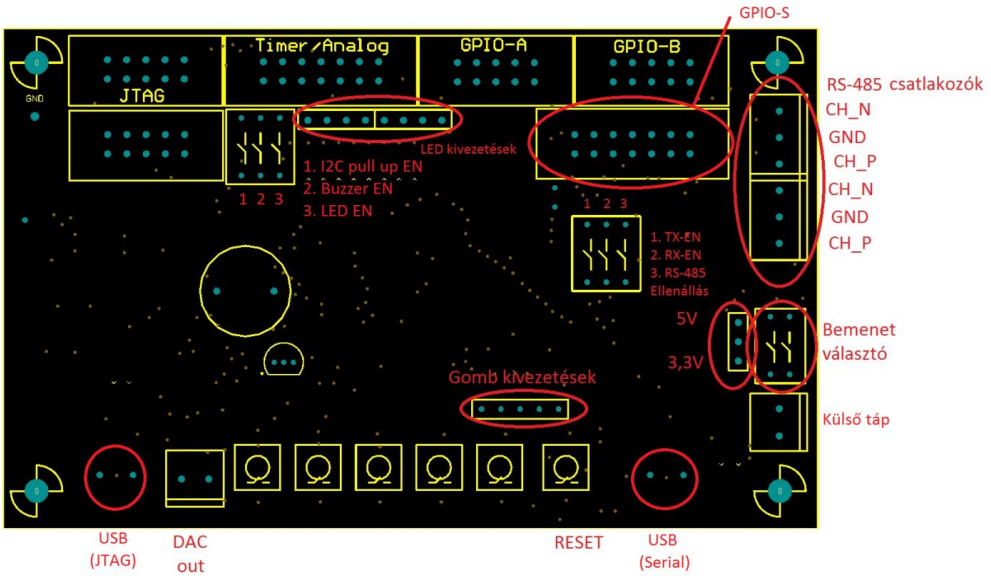
Főbb tulajdonságok

- ATMEL AVR – Atmega128 típusú mikrokontroller
- USB – soros átalakító
- valós idejű óra (RTC – Real Time Clock, PCF8563)
- Piezzo buzzer
- Analóg hőmérséklet szenzor (LM35)
- RS-485 interfész (SN75176, belső védelemmel)
- USB és külső tápellátás lehetőség
- 5 db nyomógomb
- 8 db LED
- Integrált JTAG debugger (JTAG ICE)
- védőbiztosíték
- Digitál – Analóg Átalakító

A T-bird 3 fejlesztőpanel az előző verziók teljes értékű helyettesítése. A fejlesztő panel csatlakozói és lábkiosztása megeggyezik az előző verziók kiosztásaival, ezért a kiegészítő panelok csatlakoztatása nem okoz gondot. A JTAG ICE programozónak köszönhetően a panel a vásárlás után nem igényel semmi plusz alkatrészt.

Csatlakozók ismertetése

A T-bird 3 fejlesztői panelre minden külső eszközt hagyományos szalagkábel csatlakozón keresztül tud rácsatlakoztatni.

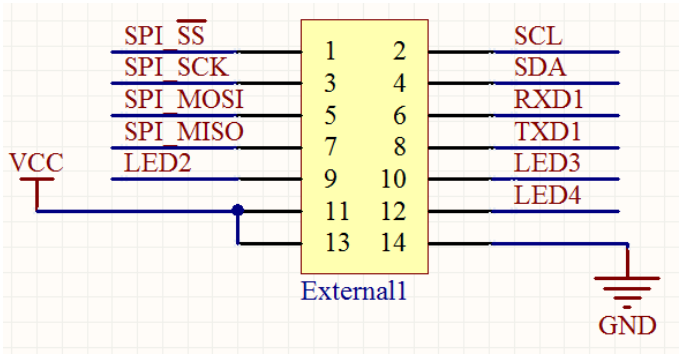


Külső táp: A fejlesztőpanel külső tápellátására szolgáló csatlakozó. A bemeneti feszültség maximális értéke +5V lehet.
Bemenet választó: A DIP-SWITCH kapcsoló segítségével választhatjuk ki, hogy mely bemeneti feszültségforrással szeretnénk megtáplálni a fejlesztőpanelt.

1. kapcsoló: külső csatlakozó
2. kapcsoló: USB csatlakozó

Egyes modellek esetében ezenfelül a csatlakozó mellett található jumperrel választhatjuk ki, hogy a panelt milyen feszültségszintről szeretnénk üzemeltetni (3.3V vagy 5V).

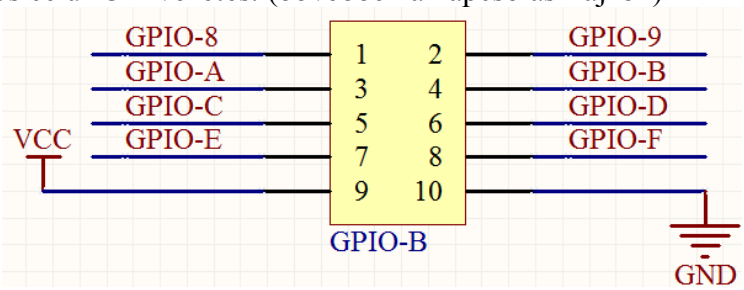
RS-485 csatlakozók: A csatlakozók duplikálása megkönnyíti a fejlesztőpanel buszba való beépítését (továbbvezetés).
RS-485 DIP SWITCH: Ezen a kapcsolón keresztül tudjuk letiltani az RS-485 meghajtó IC-t, és állítani a buszlezáró ellenállást. Fontos, hogy ha 3,3V-ról működteti a panelt, akkor ezek a kapcsolók be legyenek kapcsolva!
GPIO-S: Általános célú IO kivezetés. (bővebben a kapcsolási rajzon)



SPI_SS - PB0
SPI_SCK - PB1
SPI_MOSI - PB2
SPI_MISO - PB3
LED2 - PB6

SCL - PD0
SDA - PD1
RXD1 - PD2
TXD1 - PD3
LED3 - PB7
LED4 - PD4

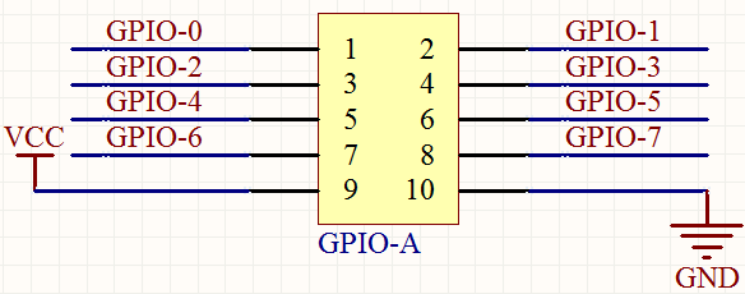
GPIO-B: Általános célú IO kivezetés. (bővebben a kapcsolási rajzon)



GPIO-8 - PC0
GPIO-9 - PC1
GPIO-A - PC2
GPIO-B - PC3

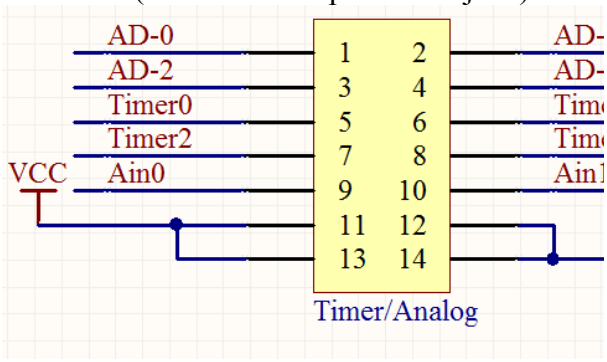
GPIO-C - PC4
GPIO-D - PC5
GPIO-E - PC6
GPIO-F - PC7

GPIO-A: Általános célú IO kivezetés. (bővebben a kapcsolási rajzon)



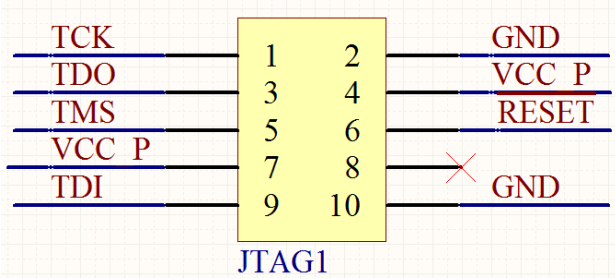
GPIO-0	- PA0	GPIO-4	- PA4
GPIO-1	- PA1	GPIO-5	- PA5
GPIO-2	- PA2	GPIO-6	- PA6
GPIO-3	- PA3	GPIO-7	- PA7

Timer/Analog: Általános célú IO kivezetés. (bővebben a kapcsolási rajzon)



AD-0 - PF0
AD-2 - PF2
Timer0 - PE4
Timer2- PE6
Ain0 - PE2
AD-1 - PF1
AD-3 - PF3
Timer1- PE5
Timer3- PE7
Ain1 - PE3

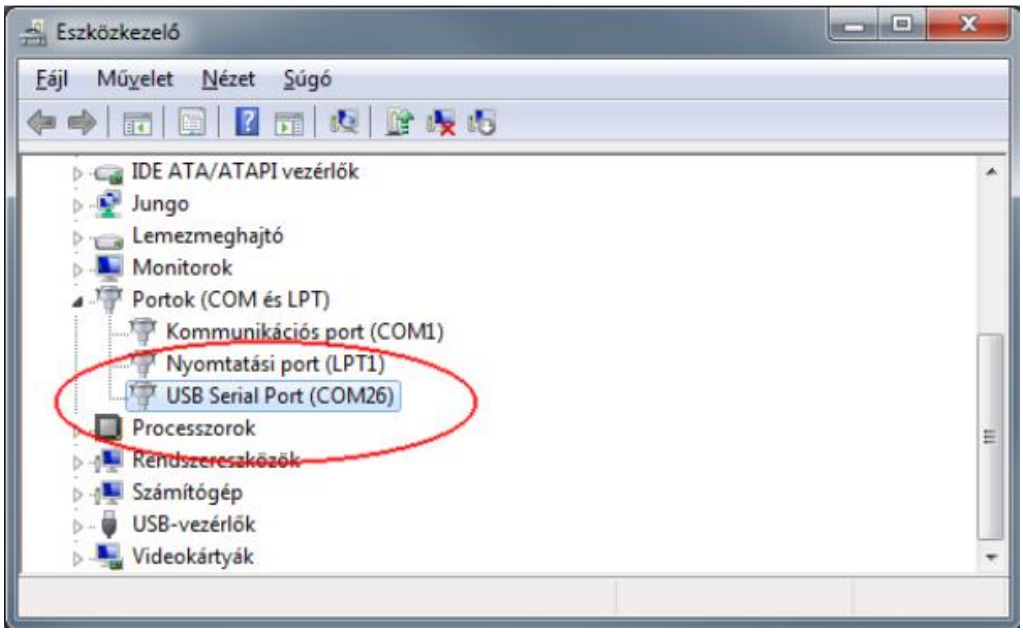
JTAG csatlakozó: Az integrált JTAG debugger csatlakozó felülete.



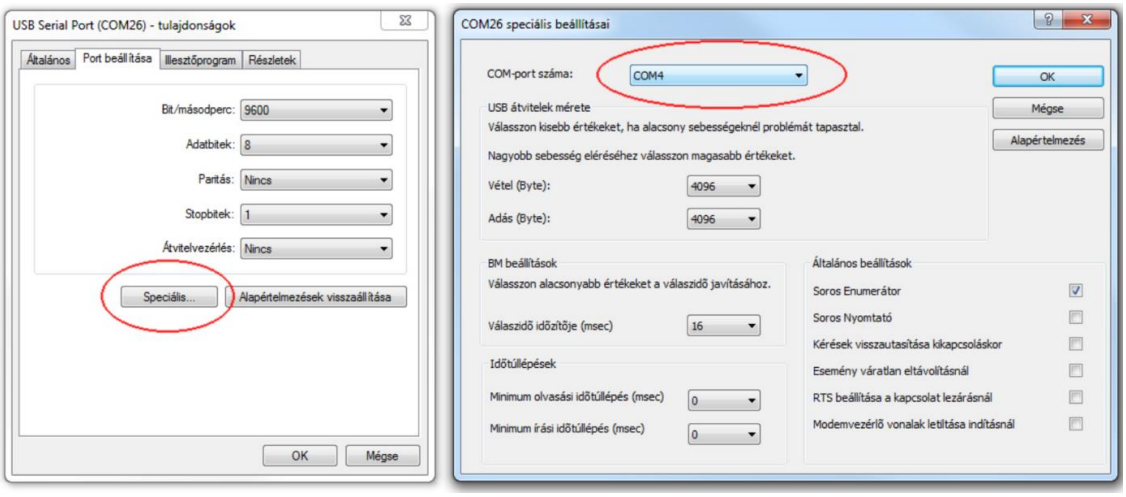
JTAG debugger használata

Az USB (JTAG) csatlakozón keresztül csatlakoztassuk a számítógéphez a fejlesztői panelt. Ekkor a számítógép automatikusan eszközillesztő szoftvert keres, az előre beállított mappákban. Amennyiben az eszközillesztő szoftver telepítése sikertelen, a legfrissebb illesztőprogram letöltése ajánlott a www.ftdichip.com oldalról, az FT232RL típusú USB-Soros illesztő IC-hez.
Figyelem! Egyes AVR Studio verziók (pl. v 4) nem képesek kezelni a magasabb port számokra kerülő virtuális soros porti JTAG debuggereket. Így szükséges lehet az eszközillesztő szoftver telepítése után a port számot módosítani, az alábbi módon:

1. Nyissuk meg a Számítógép > **Eszközkezelő** ablakot a módosítani kívánt portot (jelen esetben COM26)



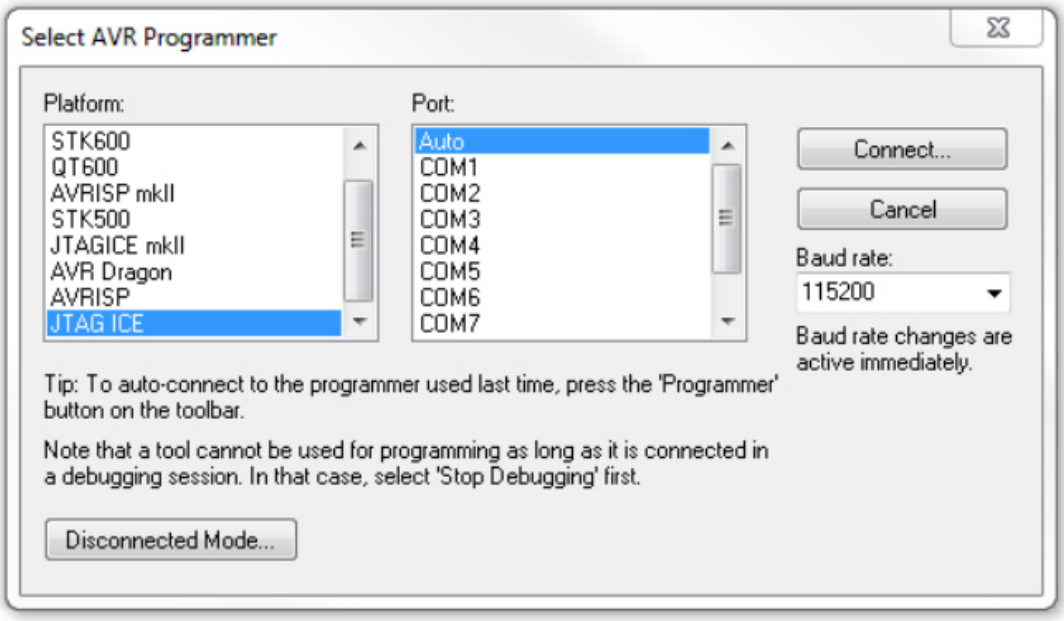
2. Jobb klikk, Tulajdonságok, majd a Port beállítása fülön kattintsunk a Speciális… gombra.
3. Állítsuk át a COM-port száma mezőt egy COM1-9-ig terjedő értékre (jelen példában COM4), majd az OK gomb megnyomásával térjünk vissza az eszközkezelőbe.



4. Ezzel a virtuális soros port mostantól COM4-en érhető el, ezt kell kiválasztani az AVR Studio programban.

A következő lépésként nyissuk meg az AVR Studio 4 programot.

Válasszuk ki a Tools > Program AVR > Connect menüpontot, majd a megjelenő listából válasszuk a JTAG ICE eszközt.



A Connect… gomb megnyomásával az AVR Studio csatlakozott is.

Fontos tudnivalók

A programozásnál és a Fuse-bit beállításoknál fokozottan ügyeljünk a JTAG, ISP és oszcillátor beállításokra. Amennyiben helytelen értékre állítjuk ezen biteket, a teljes fejlesztői panel működésképtelenné válhat, ez az eset nem tartozik a garanciális meghibásodások körébe.

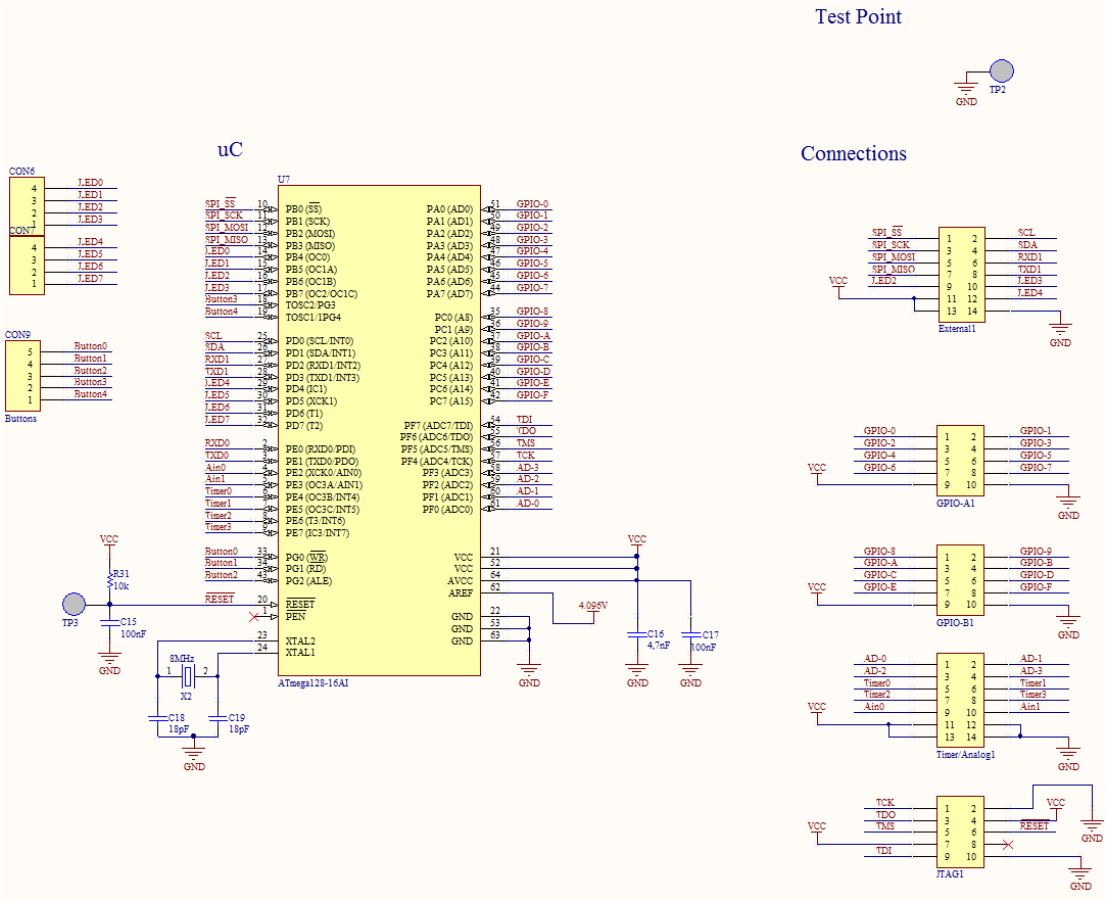
Amennyiben a JTAG és ISP Fuse biteket kikapcsoljuk, úgy abban az esetben az ATmega128 mikrovezérlő minden további programozását letiltjuk, így használhatatlanná válik a teljes fejlesztői panel. Fokozottan ügyeljünk ezen bitek beállításaira!

Amennyiben a PWR LED nem világít, de a tápfeszültséget valamelyik USB csatlakozón vagy külső tápfeszültség csatlakozón keresztül biztosítottuk, úgy abban az esetben az olvadó biztosíték szakadt meg rövidzár miatt. Távolítsuk el az áramkörből a fejlesztői panelt, és vizsgáljuk meg mivel okozhattuk a rövidzárat. Ezt követően az olvadó biztosíték cseréje szükséges.

A fejlesztői panel nem rendelkezik túlfeszültség védelemmel, így fokozottan figyeljünk külső tápfeszültség forrás alkalmazása esetén.

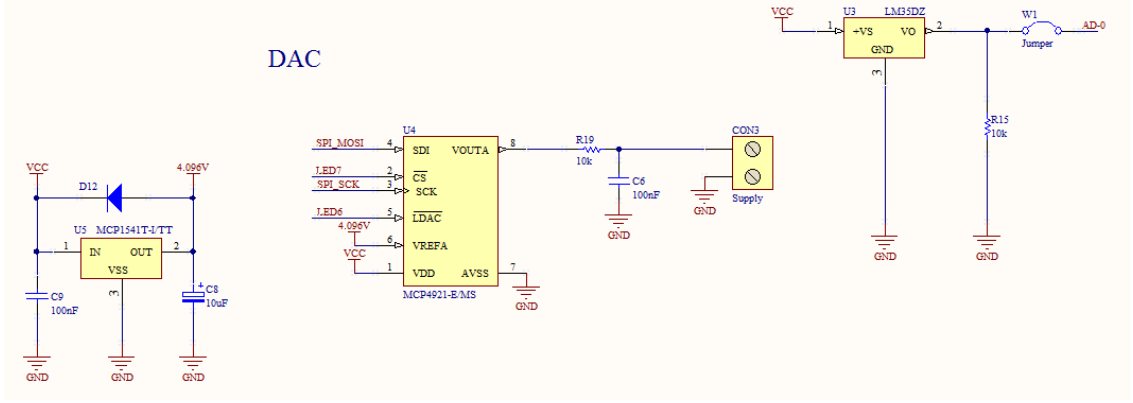
Minden T-Bird 3 fejlesztői panel részletesen, minden funkcióját tesztelve kerül forgalomba.

5.1.1 Alapáramkör

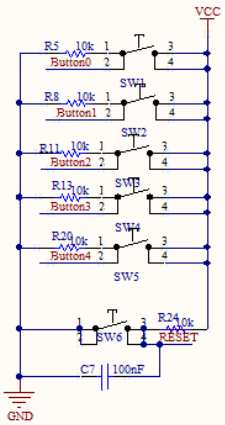


5.1.1.1 LED-ek és GOMB-ok

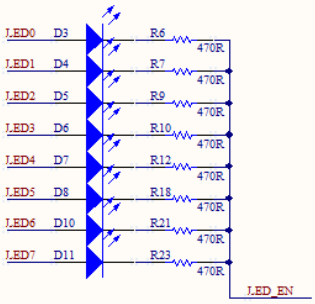
THERMAL



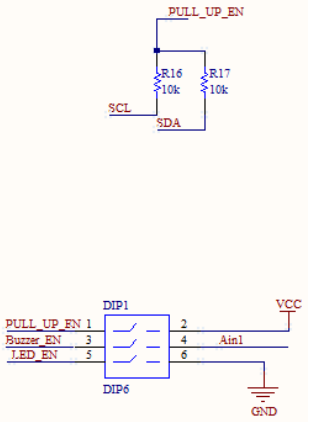
Buttons



LEDs



Pull UPs



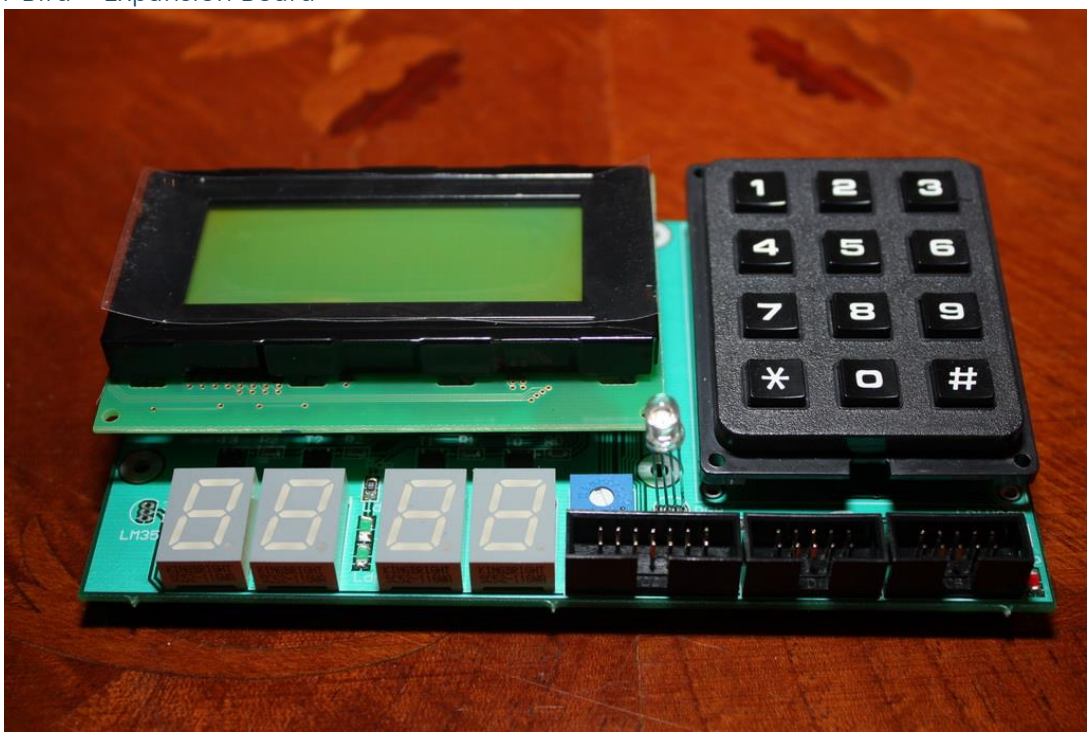
5.1.1.2 T-bird szalagkábel kiosztás

T-bird3 Ports Pins

bit	7	6	5	4	3	2	1	0	
PORT									
A	Enable	Dot	Digit Sel1	Digit Sel0	BCD Data3	BCD Data2	BCD Data1	BCD Data0	7 Segments Display
I/O	Out	Out	Out	Out	Out	Out	Out	Out	
B	LED3	LED2	LED1	LED0	SPI_MISO	SPI_MOSI	SPI_SCK	SPI_SS	LEDs first nibble
I/O	Out	Out	Out	Out					
C	RED	KBrow3	KBrow2	KBrow1	KBrow0	KBcol2	KBcol1	KBcol0	Keyboard
I/O	Out	Out	Out	Out	Out	In	In	In	
D	LED7	LED6	LED5	LED4	TXD1/INT3	RXD1/INT2	SDA/INT1	SCL/INT0	LEDs second nibble
I/O	Out	Out	Out	Out	Out/In	In	Out/In	Out/In	
E	LCD_Data7	LCD_Data6	LCD_Data5	LCD_Data4	GREEN	BLUE	TXD0/PDO	RXD0/PDI	LCD Data
I/O	In/Out	In/Out	In/Out	In/Out	Out	Out	Out	In	
F	TDI	TDO	TMS	TCK	LCD_Enable	LCD_R/W	LCD_RS	LM35	LCD Control
I/O	JTAG	JTAG	JTAG	JTAG	Out	Out	Out	In	Analog
G	NC	NC	NC	K4	K3	K2	K1	K0	Pushbutton
I/O				In	In	In	In	In	

5.1.2 Kiegészítő áramkör

5.1.3 T-Bird – Expansion Board



A T-Bird fejlesztőpanelhez készült kiegészítő board számos lehetőséget foglal magában. A panelen megjelenítésre és bevitelre alkalmas eszközök kerültek elhelyezésre.

5.1.4 Megjlenítő eszközök

5.1.4.1 LCD kijelző, 4x16 karakter, HD44780 kompatibilis

Az LCD kijelző 4 adatbites üzemmódban használható, így 3 vezérlő és 4 adat vonallal rendelkezik. Egy karakter kiírása ebből adódóan két írási ciklussal jár.
A port bitekkel történő takarékosság érdekében az LDC kijelző az 4 bites üzemmódban vezérelhető, így a teljes LCD modul vezérléséhez összesen 7 bit került felhasználásra (4 bit adat, és 3 bit vezérlőjel). Az LCD modul adatbitjei (D4-D7) a mikrokontroller PF4-PF7 bitjein találhatók. A vezérlőjelek pedig következőképpen kerültek bekötésre:

PF7	PF 6	PF 5	PF 4	PF 3	PF 2	PF 1	PF0
LCD D7	LCD D6	LCD D5	LCD D4	Enable	RW	RS	Hőszenzor

Az LCD kijelző háttérvilágítása a kijelző jobb alsó sarkában található potenciométer használatával szabályozható. Ennek beállítására esetlegesen akkor lehet szükség, ha a kijelzőn semmilyen információt nem látunk. A fényerő maximális értékre történő állítása esetén a 4 sorban az egyes karakterek foltjai fognak látszani, mutatva azt, hogy a kijelző bekapcsolt állapotban van.

http://en.wikipedia.org/wiki/HD44780_Character_LCD
http://lcd-linux.sourceforge.net/pdfdocs/hd44780.pdf

5.1.4.2 4db hétszegmenses kijelző

A hétszegmenses kijelzők multiplexerrel valamint meghajtó áramkörrel vannak ellátva. A meghjató áramkörnek BCD kódban kell megadni a kiírni kívánt számot, a multiplexernek pedig szintén BCD kódban kell megmondani, hogy melyik kijelzőre szeretnénk kiírni az adott számot. A legkisebb cím a jobboldali kijelzőhöz tartozik (0), majd balra növekszik (egészen háromig). Lehetőség van még két db LED-et is kiválasztani a multiplexerrel (4-es cím), amelyek a hétszegmenses kijelzők között találhatóak meg, így könnyedén kialakítható egy óra.
A 7 szegmenses kijelző vezérlése a mikrokontroller A portján keresztül történik. A kiküldendő

információ felépítése a táblázatban található. Adott digitre kiküldendő érték a bájt alsó 4 bitjén (PA0-PA3) lehet megadni binárisan. A négy digit közül, hogy melyikre történjen ezen információ kiírása, azt a digit választó bitek (PA4-PA5) segítségével lehet kijelölni.
A két-két digit között található LED vezérlése a PA6 bit segítségével valósítható meg. A kijelző engedélyezését a PA7 bit végzi. A kiíratás során ügyelni kell arra, hogy egyszerre csak egy digitre lehet információt kivinni, így ha folyamatos kiíratást akarunk elérni, akkor a digitekre való kiíratást javasolt kb. 400 Hz-es sűrűséggel ismételni.
A 4 db hétszegmenses kijelző vezérlő bájtja:

PA7	PA 6	PA 5	PA 4	PA 3	PA 2	PA 1	PA0
Hétszeg-menses ki-jelző engedélye-zés	Középső LED	Digit vá-lasztó 1	Digit vá-lasztó 0	BCD D	BCD C	BCD B	BCD A

A hétszegmenses kijelző digit választása:

Digit választó: 11	Digit választó: 10	Középső LED	Digit választó: 01	Digit választó: 00

A hétszegmenses kijelzőre való kiíratást ugyanúgy, mint a billentyűzet lekezelését is érdemes egy Timer megszakítási rutinban elhelyezni, amely megszakítás a rendszer időalapját is képezheti.

http://www.nxp.com/documents/data_sheet/HEF4511B.pdf
http://ics.nxp.com/products/hc/datasheet/74hc238.74hct238.pdf

5.1.4.3 3 színű LED

A panelen megtalálható egy 3 színű LED, amelyet például 3 szoftveres PWM jellel meghajtva könnyedén előállítható 256 (4 bites PWM) színkombináció vagy akár 16 Millió (8 bites PWM).
A háromszínű LED vezérlése a PC7 (RED), a PE2 (GREEN) és a PE3 (BLUE) port biteken keresztül történhet. A vezérlés során lehetőség van a LED-ek fényerejének változtatására is PWM jel segítségével. A három szín megfelelő arányú keverésével tetszőleges szín előállítható, illetve ezen szín fényereje is állítható.

5.1.5 Bemeneti eszközök

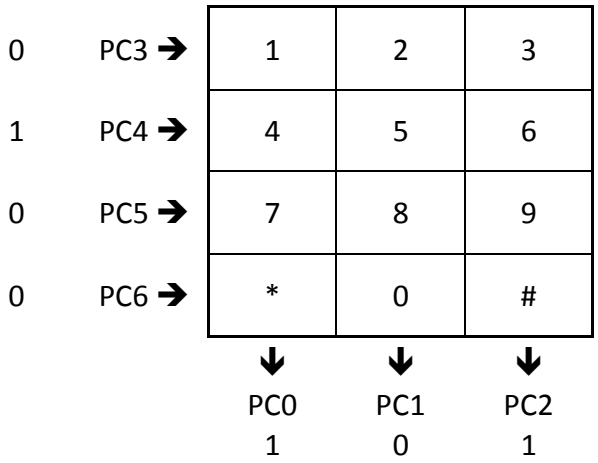
5.1.5.1 3x4-es billentyűzet mátrix

A billentyűzet mátrix soraira aktív jelet adva (időben egyszerre mindíg csak egyre), és az oszlop jeleket beolvasva könnyedén megállapítható, hogy adott pillanatban melyik gomb került lenyomásra. A billentyűzet mátrix megcímzése a mikrokontroller PC3-PC6 port bitjein keresztül lehetséges. A kiválasztani kívánt sorra logikai 1-et, a másik három címző vezetékre logikai 0-t adva, az adott sorban levő lenyomott billentyűzet értékét tudjuk vissza olvasni a PC0-PC2 port vezetékeken keresztül.
Ciklikusan végigcímezve a többi vezetéket, az éppen megcímzett sorokban is vizsgálhatjuk a lenyomott billentyűket. A ciklikusság gyakoriságát érdemes olyan sűrűre választani, hogy az egyes

billentyűk pergése már ne okozzon problémát, de ne is kelljen túl sokáig nyomni a gombot.

PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0
RGB LED RED	Billentyű-zet címző D	Billentyű-zet címző C	Billentyű-zet címző B	Billentyű-zet címző A	Billentyű-zet olvasó C	Billentyű-zet olvasó B	Billentyű-zet olvasó A

Példa:
PC3-PC6-ra küldjünk ki 0010 értéket – ez a második sort címzi meg – és olvassuk vissza a PC0-PC2 vonalakat. Ha a visszaolvasott érték maszkolás után 101, akkor a középső oszlopban levő billentyű került lenyomásra, a második sor esetén az 5-ös billentyű.



5.1.5.2 LM35 hőszenzor

A hőmérséklet érzékelő szenzor kimenete a mikrokontroller analóg bemenetére van kötve. A kontroller belső A/D átalakítóját használva lehet digitalizálni, majd egy skálatényezővel megmondani, hogy mekkora a hőmérséklet értéke.
Az LM35-ös hőmérsékletérzékelő a mikrokontroller PF0 analóg bemenetére kapcsolódik, amely analóg jelet a belső referencia tápfeszültség felhasználásával vizsgálhatjuk meg.

http://www.national.com/ds/LM/LM35.pdf

5.1.5.3 Csatlakozó kiosztás

A kiegészítő board teljes egészében illeszkedik a T-Bird csatlakozó kiosztásához. A lenti táblázatban található meg a csatlakozók kiosztása:

Csatlakozó	[lábszám]	I/O	Funkció
GPIO-A	[1]	Output	7seg. meghajtó BCD kód „A”
	[2]	Output	7seg. meghajtó BCD kód „B”
	[3]	Output	7seg. meghajtó BCD kód „C”
	[4]	Output	7seg. meghajtó BCD kód „D”
	[5]	Output	7seg. demux „A”
	[6]	Output	7seg. demux „B”
	[7]	Output	7seg. demux „C”
	[8]	Output	7seg. demux enable
	[9]	Power	5V
	[10]	Power	GND
GPIO-B	[1]	Input	Bill. mátrix bal oldali oszlop
	[2]	Input	Bill. mátrix középső oszlop
	[3]	Input	Bill. mátrix jobb oldali oszlop
	[4]	Output	Bill. mátrix első sor
	[5]	Output	Bill. mátrix második sor
	[6]	Output	Bill. mátrix harmadik sor
	[7]	Output	Bill. mátrix negyedik sor
	[8]	Output	RGB LED „Red”
	[9]	Power	5V
	[10]	Power	GND
Timer/Analog	[1]	Input	Hőszenzor, analóg bemenet
	[2]	Output	LCD RS
	[3]	Output	LCD R/W
	[4]	Output	LCD Enable
	[5]	I/O	LCD data[4]
	[6]	I/O	LCD data[5]
	[7]	I/O	LCD data[6]
	[8]	I/O	LCD data[7]
	[9]	Output	RGB LED „Blue”
	[10]	Output	RGB LED „Green”
	[11]	Power	5V
	[12]	Power	GND
	[13]	Power	5V
	[14]	Power	GND

T-bird	Csatlakozó	Kiegészítő panel
	GPIO-A	
PA3-PA0	4-1	BCD kód a kiválasztott digitre
PA5-PA4	6-5	digit választó
PA6	7	digitek közötti középső pontok
PA7	8	demux enable
	GPIO-B	
PC2-PC0	3-1	Billentyűzetmátrix olvasása
PC6-PC3	7-4	Billentyűzetmátrix címzése (normál)
PC7	8	RGB LED RED
	Timer-Analog	
PF0	1	Hőszenzor
PF1	2	LCD RS
PF2	3	LCD RW
PF3	4	LCD Enable
PF7-PF4	8-5	LCD data 7-4
PE3	9	RGB LED BLUE
PE2	10	RGB LED GREEN
	TWI	
PD0	SCL	
PD1	SDA	
	USART	
PD2	RX	
PD3	TX	
	SPI	
PB0	SS	
PB1	SCK	
PB2	MOSI	
PB3	MISO	

6 Assembly nyelvű programozás

6.1 Assembly nyelv és az assembler

Az assembly programozási nyelv a mikrokontroller utasításkészletére épül (Atmega128 esetében ez 133 utasítás), és ezeket az utasításokat 2, 3 vagy 4 betűs memonikok segítségével írja le.
Az assembler az egy compiler típusú fordító program, amely a forráskódú állományból állít elő tárgykodú állományt, amelyből a linker fogja előállítani a letölthető hex állományt.

6.2 Nyelvi elemek

6.2.1 Utasítások

- Fordítónak szóló utasítások
- Programutasítások

6.2.1.1 Fordítónak szóló utasítások

Direktíva vagy pszeudó utasítás.

6.2.1.2 Programutasítások, utasításcsoportok

1. Aritmetikai és logikai utasítások
2. Vezérlésátadó utasítások
    a. Ugró utasítások
    b. Szubrutin hívások
    c. Komparáló utasítások
3. Bitállító és -tesztelő utasítások
4. Adatmozgató utasítások
5. MCU vezérlő utasítások
6. Interrupt utasítások

6.2.2 Címzési módok

3 féle kódterület címzéssel találkozhatunk:

- Közvetlen kódterület cím
    - Abszolút cím
        - Feltételes ugró utasítás
        - Feltétel nélküli ugró utasítás
        - Szubrutin hívás
    - Relatív cím
        - Feltételes ugró utasítás
        - Feltétel nélküli ugró utasítás
        - Szubrutin hívás
- Indirekt cím
    - Indirekt szubrutin hívás

5 féle adatterület címzési mód van:

- Regiszter címzés
- Regiszter indirekt címzés
- Memória címzés
- Memória indirekt címzés
- Verem terület címzése

6.2.3 Assembly programsor felépítése

6.2.3.1 Fordítónak szóló utasítássor

6.2.3.2 Programutasítás sor

6.2.4 Assembly forráskódú program fordításának lépései

6.2.4.1 Parancssori fordítás

6.2.4.2 Fejlesztői környezet használata

Forráskód *.asm
Compiler
Tárgykód *.obj
Linker
Hex kód *.hex
Debugger
Debugger kód *.elf

6.2.5 Programutasítások csoportok

6.2.5.1 Adatmozgató utasítások

- Általános hozzáférésű I/O regiszter
  - 0x00 – 0x1F közvetlen bitenkénti kezelés (SBI, CBI)
  - 0x00 – 0x3F IN vagy OUT utasítással, Közvetlen elérésű I/O terület
  - 0x60 – 0xFF ST/STS/STD, LD/LDS/LDD, Külső elérésű (SRAM) I/O terület
- GPIOR0, GPIOR1, GPIOR2: R/W

MOV Rd, Rr; Rd ← Rr
MOV Rd, Rr; Rd+1:Rd ← Rr+1:Rr

- Load

LDI Rd, K; Rd ← K
SRAM
LDS Rd, k; Rd ← (k), SRAM
STS k, Rr; (k) ← Rr
X, Y, Z
LD Rd, X; Rd ← (X)
LD Rd, X+; Rd ← (X), X ← X + 1
LD Rd, -X; X ← X - 1, Rd ← (X)
Y, Z
LDD Rd, Y+q; Rd ← (Y + q)

- Store

X, Y, Z
ST X, Rd ;(X) ← Rr
ST X+, Rd; (X) ← Rr, X ← X + 1
ST –X, Rd; X ← X - 1, (X) ← Rr
Y, Z
STD Y+q, Rd; (Y + q) ← Rr

- Load Program Memory,
- Store Program Memory

LPM; R0 ← (Z)
LPM Rd, Z; Rd ← (Z)
LPM Rd, Z+; Rd ← (Z), Z ← Z+1
SPM; (Z) ← R1:R0

- Veremkezelés

PUSH Rr; STACK ← Rr
POP Rd; Rd ← STACK

6.2.5.2 Aritmetikai utasítások

- Összeadás

ADD Rd, Rr; Rd ← Rd + Rr
ADC Rd, Rr; Rd ← Rd + Rr + C
ADIW RdI, K; Rdh:Rdl ← Rdh:Rdl + K
INC Rd; Rd ← Rd + 1

- Kivonás

SUB Rd, Rr; Rd ← Rd - Rr
SUBI Rd, K; Rd ← Rd – K
SBC Rd, Rr; Rd ← Rd – Rr – C
SBCI Rd, K; Rd ← Rd – K – C
SBIW RdI, K; Rdh:Rdl ← Rdh:Rdl – K
DEC Rd; Rd ← Rd − 1

- Szorzás
  - Előjel nélkül

MUL Rd, Rr; R1:R0 ← Rd x Rr
  - Előjeles

MULS Rd, Rr; R1:R0 ← Rd x Rr
  - Előjeles előjel nélkülivel

MULSU Rd, Rr; R1:R0 ← Rd x Rr
  - Fractional

FMUL Rd, Rr; R1:R0 ← Rd x Rr << 1
FMULS Rd, Rr; R1:R0 ← Rd x Rr << 1
FMULSU Rd, Rr; R1:R0 ← Rd x Rr << 1

6.2.5.3 *Logikai utasítások*

- AND

  AND Rd, Rr; Rd ← Rd • Rr
  ANDI Rd, Rr; Rd ← Rd • Rr

- OR

  OR Rd, Rr; Rd ← Rd v Rr
  ORI Rd, Rr; Rd ← Rd v Rr

- EOR

  OR Rd, Rr; Rd ← Rd ⊕ Rr

- COM, 1' complement

  COM Rd; Rd ← 0xFF – Rd

- NEG , 2' complement

  NEG Rd; Rd ← 0x00 – Rd

- SBR

  SBR Rd, K; Rd ← Rd v K

- CBR

  CBR Rd, K; Rd ← Rd • (0xFF - K)

- TST

  TST Rd; Rd ← Rd • Rd
  CLR Rd; Rd ← Rd ⊕ Rd
  SER Rd; Rd ← 0xFF

6.2.5.4 *Vezérlés átadó utasítások*

- Ugró utasítás
  - Feltétel nélküli
  - Feltételes
    - Branch if
    - Skip if
- Szubrutin hívás
- Komparáló utasítás

6.2.5.5 *Feltétel nélküli ugró utasítások*

- Direkt ugrás

  JMP k; PC ← k

- Indirekt ugrás

  IJMP; PC ← Z

- Relatív ugrás

  RJMP k; PC ← PC + k + 1

6.2.5.6 Feltételes ugró utasítások

- Branch if
  - Branch if Status Flag SET/CLEAR

    BRBS s, k; if (SREG(s) = 1) then PC←PC+k + 1
    BRBC s, k; if (SREG(s) = 0) then PC←PC+k + 1
    SBIC P,b; if (P(b)=0) PC ← PC + 2 or 3
    SBIS P,b; if (P(b)=1) PC ← PC + 2 or 3
- Branch if …

  aritmetikai: EQ (=), NE(≠), SH (>=), LO (<), GE, LT
  flag: C, M, P, HC, T, Ov, I

6.2.5.7 Szubrutin hívó utasítások

- Feltétel nélküli szubrutin hívás
  - Direkt szubrutin hívás

    CALL k; PC ← k
  - Indirekt szubrutin hívás

    ICALL; PC ← Z
  - Relatív szubrutin hívás

    RCALL k; PC ← PC + k + 1
  - Szubrutinból való visszatérés

    RET

6.2.5.8 Bitállító utasítások

SBI P,b; I/O(P,b) ← 1
CBI P,b; I/O(P,b) ← 0
BSET s; SREG(s) ← 1
BCLR s; SREG(s) ← 0
BST Rr, b T ← Rr(b)
BLD Rr, b Rd(b) ← T
Set, Clear C, N, Z, I, S, O, T, H

6.2.5.9 Bitléptető és bitforgató utasítások

- Logikai shift

  LSL Rd; Rd(n+1) ← Rd(n), Rd(0) ← 0
  LSR Rd; Rd(n) ← Rd(n+1), Rd(7) ← 0
- Rotálás Carry-n keresztül

  ROL Rd; Rd(0)←C,Rd(n+1)← Rd(n),C←Rd(7)
  ROR Rd; Rd(7)←C,Rd(n)← Rd(n+1),C←Rd(0)

- Aritmetikai shift

ASR Rd; Rd(n) ← Rd(n+1), n=0..6
SWAP Rd; Rd(3..0)←Rd(7..4),Rd(7..4)←Rd(3..0)

6.2.5.10 Bemeneti és kimeneti utasítások

Input
IN Rd, P ; Rd ← P
Output
OUT P, Rr; P ← Rr

6.2.5.11 Megszakítások

- Global Interrupt Enable
- Interrupt Vector table

RETI
- Típusai
  - Esemény által triggerelt
  - Interrupt feltételek fennállása

6.2.5.12 Processzorvezérlő utasítások

NOP;
SLEEP;
WDR; Watchdog Reset
BREAK; On-chip Debug

7 T-bird3 programozása (példaprogramok)

7.1 LED-ek kezelése

7.1.1 Futófény variációk

```
//----------------------Run----------------------------------

#define F_CPU 8000000UL                                    //órajel frekvencia

#include <avr/io.h>
#include <avr/delay.h>

void Init(void);

int main(void)
{

unsigned char i;
        Init();

        i=1;
        while(1)
        {
                i=i<<1;
                led_out(i);

                if(i>=128){
                led_out(i);
                _delay_ms(50);
                i=1; led_out(i);}
                _delay_ms(50);
        }

        return 0;

}

void led_out(unsigned char ertek)
{

        PORTD=ertek;
        PORTB=ertek<<4;
}

void Init(void)
{

        DDRB=0xf0;;
        DDRD=0xf0;
}
```

```
//----------------------Run---------------------------------


//----------------------Left-Right---------------------------------

#define F_CPU 8000000UL                                 //órajel frekvencia

#include <avr/io.h>
#include <avr/delay.h>

void Init(void);

int main(void)
{

unsigned char i, irany=0;
        Init();

        i=1;
        while(1)
        {
                if(irany==0) {i=i<<1; led_out(i); _delay_ms(50); if(i==128) {irany=1;}}
                if(irany==1) {i=i>>1; led_out(i); _delay_ms(50); if(i==1) {irany=0;}}

        }

        return 0;

}

void led_out(unsigned char ertek)
{

        PORTD=ertek;
        PORTB=ertek<<4;
}

void Init(void)
{

        DDRB=0xf0;;
        DDRD=0xf0;


}

//----------------------Left-Right---------------------------------



//----------------------Knight Rider---------------------------------
#define F_CPU 8000000UL                                 //órajel frekvencia
```

```
#include <avr/io.h>
#include <avr/delay.h>

void Init(void);

int main(void)
{

unsigned char i,j, irany=0;
	Init();

	i=1; j=128;
	while(1)
	{
		if(irany==0) {i=i<<1; j=j>>1; led_out(i|j); _delay_ms(50); if(i==128) {irany=1;}}
		if(irany==1) {i=i>>1; j=j<<1; led_out(i|j); _delay_ms(50); if(i==1) {irany=0;}}

	}

	return 0;

}

void led_out(unsigned char ertek)
{

	PORTD=ertek;
	PORTB=ertek<<4;
}

void Init(void)
{

	DDRB=0xf0;;
	DDRD=0xf0;


}
//----------------------Knight Rider----------------------------------
```

7.1.2 PWM-zet LED

7.2 GOMB-ok kezelése

7.3 Időzítők üzemmódjai

7.4 Számlálók üzemmódjai

7.5 USB kezelés

7.6 Hétszegmenses

7.6.1 Assembly

```
.INCLUDE "m128def.inc"                  ; Include fájl (Atmega128)

.DSEG
;--------------------
; VÁLTOZÓK
;--------------------

.org 0x100
       digit_s:  .db 4
       number_s: .db 2

.CSEG
;--------------------
; PROGRAMKÓD TERÜLET
;-------------------

;---------------------------------------
; DEFINÍCIÓS RÉSZ, HIVATKOZÁSOK
;---------------------------------------
;ebben lesznek a kijelző cuccai!

.def cnt = r22 ;digit azonosító
.def tmp2 = r21
.def tmp = r20
.def timer_cnt = r19

.equ fq = 16000000

.ORG 0x00
       rjmp   main
.ORG OC0addr
       rjmp   TOCH

                                   ; ugrás a main címkére
.ORG 0x100                        ; pozicionálás a megszakítás vektor tábla utánra
;--------------------
; MACRO
;--------------------
.macro Init

;IO

       ldi tmp,(1<<DDB5)|(1<<DDB6)
       out DDRB,tmp
       ldi tmp,(1<<DDD7)
       out DDRD,tmp
       ldi tmp,0xFF
       out DDRA,tmp               ; 7 szegmenses kimenet

;Timer PWM config
       ldi tmp,(1<<COM1A1)|(1<<WGM10)    ; FPWM.A1 csatornán
       out TCCR1A,tmp
       ldi tmp,(1<<WGM12)|(1<<CS12)
       out TCCR1B,tmp

;Timer0 CTC config
       ldi tmp,(1<<WGM01)|(1<<CS02)
       out TCCR0, tmp
       ldi tmp,78
```

```
        out OCR0,tmp
        ldi tmp,(1<<OCIE0)
        out TIMSK,tmp
        sei
.endmacro



;--------------------
; SUBRUTIN
;--------------------
nullaz:
        ldi           r26, LOW(digit_s)
        eor    tmp ,tmp
        eor    tmp2 ,tmp2
tolt:
        st            X+,tmp2
        inc           tmp
        cpi    tmp, 4
        brne   tolt
        ldi    r26, LOW(digit_s)
        ret
;--------------------
; INTERRUPT RUTIN
;--------------------

TOCH:

        dec    timer_cnt
        brne kiir

        ldi r26,low(digit_s)
        ldi    tmp,3
        add    r26,tmp

        ld     tmp,X
        inc tmp
        cpi tmp, 10
        breq tizes
        st     X, tmp
        rjmp kiir
tizes:
        eor    tmp, tmp
        st     X, tmp
        ld     tmp, -X
        inc    tmp
        cpi tmp, 10
        breq szazas
        st     X, tmp
        rjmp kiir
szazas:
        eor    tmp, tmp
        st     X, tmp
        ld     tmp, -X
        inc    tmp
        cpi tmp, 10
        breq ezres
        st     X, tmp
        rjmp kiir
ezres:
        eor    tmp, tmp
        st     X, tmp
        ld     tmp, -X
        inc    tmp
        cpi tmp, 10
```

```
	breq torol
	st	X, tmp
	rjmp kiir
torol:
	rcall nullaz
	ldi r26,low(digit_s)
	ldi	tmp,3
	add	r26,tmp
kiir:

	dec	cnt

	ld		tmp,X+
	andi	tmp,0x0F	;tmp-ben benne a maszkolt digitre írandó!

	mov	tmp2,cnt		;tmp2-be bemasolom a mux-ot
	andi	tmp,0x07		;maszkolom a muxot
	swap	tmp2			;<<4 sfitelem
	or		tmp,tmp2			;egymásra rakom őket tmp2-t használhatom

	ldi	tmp2,(1<<7)		;az enablet eloallítom
	or		tmp,tmp2		; ráteszem az eddigi cuccra
	out	PORTA,tmp	;majd kiírom a portra

	cpi	cnt,0xff
	brne	next_digit
	ldi	cnt, 4
	ldi	r26,low(digit_s)
next_digit:
	reti

;--------------------
; PROGRAM
;--------------------


;--------------------
; STACK INIT
;--------------------
main:
	ldi	tmp, HIGH(RAMEND)
	out	SPH, tmp
	ldi	tmp, LOW(RAMEND)
	out	SPL, tmp
;--------------------
; INIT
;--------------------
	Init
	rcall nullaz

;MEGSZAKÍTÁSOK, INTERFACE, IDOZÍTOK

;--------------------
; PROGRAM
;--------------------
loop:

	rjmp	loop
```

7.6.2 C-ben

```
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
//============================DEFINES
#define LED_DDR             DDRB
#define LED_PORT   PORTB
#define LED_PIN             PB5
//7 segment:
#define DISP_PORT  PORTA
#define DISP_DDR   DDRA
#define DISP_EN             PA7
#define DISP_MASK0xFF
#define DIG_NUM             3       // 4db 0->3 ig

//============================MACROS
//7 segment:
#define DISP_OUT() DISP_DDR = DISP_MASK
#define DISP_ON()   DISP_PORT |= (1<<DISP_EN)
#define DISP_OFF()  DISP_PORT &=~(1<<DISP_EN)

#define LEDOUT()    LED_DDR |= (1<<LED_PIN)
#define         LED_ON()    LED_PORT|= (1<<LED_PIN)
#define         LED_OFF()   LED_PORT&=~(1<<LED_PIN)
//============================GLOB.VAR
volatile unsigned char tmp;
volatile unsigned char disp_data[4];  //Kijelzo adat
volatile unsigned char disp_idx;
//============================PREDEC
void Init(void);
void delay_10ms(unsigned char val);
void kiir(unsigned char num);
void printnum(unsigned short num);
//============================MAIN
int main(){

        Init();
        kiir(0x81);
        printnum(1254);
        for(;;){

        }
}
//============================FUNCTIONS
void Init(void){
        LEDOUT();
        DDRD = 0xF0;
        DDRB = 0xF0;
        //Timer0 CTC:
        TCCR0 |= (1<<WGM01)|(1<<CS02);          //DIV256 CTC mode
        OCR0   = F_CPU/256/800-1;               //~400 Hz
        TIMSK |= (1<<OCIE0);                    //CTC IRQ ENABLE
```

```
	//GLobal IRQ handling!
	sei();
	DISP_OUT();
	DISP_ON();
}
//delay
void delay_10ms(unsigned char val){
	while(val--){
		_delay_ms(10);
	}
}
//kiir
void kiir(unsigned char num){
	PORTD = num & 0xF0;
	PORTB = (num << 4) & 0xF0;
}

void printnum(unsigned short num){

	disp_data[0] = num/1000; num%=1000;
	disp_data[1] = num/100;  num%=100;
	disp_data[2] = num/10;   num%=10;
	disp_data[3] = num;
}

//============================IRQ VECT.
ISR(TIMER0_COMP_vect){
	DISP_OFF();				//LEKAPCS!
						//ADAT RÉSZ
	INDEX RÉSZ
	DISP_PORT = (disp_data[DIG_NUM-disp_idx] & 0x0F) | ((disp_idx<<4) & 0x70);
	DISP_ON();
	//határolás!
	disp_idx++;
	if(disp_idx>DIG_NUM)	disp_idx = 0;
}
```

7.7 Billentyűzet mátrix

7.7.1 Assembly

```
.include "m128def.inc"

.CSEG

.def   tmp  = r16
.def   tmp2 = r17
.def   tmp3 = r18
.def   tmp4 = r19
.def   tmp5 = r20
.def   tmp6 = r21
.def   cikl = r24

.org 0x00

 rjmp start

.org 0x100
.macro       stack_init
       ldi          tmp, high(RAMEND)
       out          SPH, tmp
       ldi          tmp, low(RAMEND)
       out          SPL, tmp
.endmacro
.macro       port_init
       ldi          tmp,   0xff
       out          DDRA,  tmp
       ldi          tmp,   0x78
       out          DDRC,  tmp
.endmacro

delays:
       ldi          tmp6, 10
       eor          tmp5, tmp5
       eor          tmp4, tmp4
delays_next:
       dec          tmp4
       brne   delays_next
       dec          tmp5
       brne   delays_next
       dec          tmp6
       brne   delays_next
       ret

start:
       stack_init
       port_init

       ldi          tmp2,  0x08
       ldi          ZH,          high(keyb_nums<<1)
       ldi          ZL,          low(keyb_nums<<1)
       eor          tmp3,  tmp3
       ldi          cikl,  12

keyb_write:
       ori          tmp3,  0x80
       out          PORTA, tmp3
       call   delays
keyb_read:
       cpi          tmp2,  0x80
       breq   keyb_base
       ldi          cikl,  12
```

```
        ldi             ZL,             low(keyb_nums<<1)
        out     PORTC, tmp2
        in              tmp,    PINC
        lsl             tmp2
        rjmp    keyb_array
keyb_base:
        ldi             tmp2,   0x08
        rjmp    keyb_read
keyb_array:
        lpm             tmp3,   Z+
        cp              tmp,    tmp3
        brne    next_key
        lpm             tmp3,   Z+
        rjmp    keyb_write
next_key:
        lpm             tmp3,   Z+
        dec             cikl
        brne    keyb_array
        rjmp    keyb_read


keyb_nums:      .DB 69,0,14,1,13,2,11,3,22,4,21,5,19,6,38,7,37,8,35,9,67,10,70,11

//-------------------------------------------------------------------------------


//-------------------------------------------------------------------------------
.include "m128def.inc"

.def    tmp             =       r16
.def    cim             =       r17
.def    bill    =       r18
.def    cikl_valt       =       r19
.def    hetszeg                 =       r20

.org 0x00
                rjmp    start

.org 0x100

.macro stack_init
                ldi             tmp,    high(RAMEND)
                out             SPH,    tmp
                ldi             tmp,    low(RAMEND)
                out             SPL,    tmp
.endmacro
.macro port_init
                ldi             tmp,    0xff
                out             DDRA,           tmp
                ldi             tmp,    0x78
                out             DDRC,           tmp
.endmacro


start:
                stack_init
                port_init
```

```
                ldi             ZH,             high(keyb_nums)
                ldi             ZL,             low(keyb_nums)
kezd:
                ldi             cim,    0x08
ujra:
                out             PORTC,          cim
                in              bill,   PINC
                ldi             cikl_valt,12
                eor             hetszeg,        hetszeg
ciklus:
                cpi             bill,   Z+
                breq    kesz
                inc             ZL
                dec             cikl_valt
                brne    ciklus

kesz:
                ori             hetszeg,        0x80
                ld              tmp,            Z
                or              hetszeg,        tmp
                out             PORTA,                  hetszeg
                lsl             cim
                cpi             cim,            0x80
                breq    kezd
                rjmp    ujra
keyb_nums: .DB 69,0,14,1,13,2,11,3,22,4,21,5,19,6,38,7,37,8,35,9,67,10,70,11

//-------------------------------------------------------------------------------
```

```
.include "m128def.inc"

.DSEG
.org 0x100
szamok: .db 4

.CSEG
.def digit          =       r8

.def tmp            =       r16
.def tmp2           =       r17
.def tmp3           =       r18
.def tmp4           =       r19
.def tmp5           =       r20
.def tmp6           =       r21

.ORG 0x00
       rjmp   start

.ORG 0x100
.macro init
               ; 7 szegmenses kijelző
       ori             tmp,    0xff
       out     DDRA,   tmp
       ldi             tmp,    high(RAMEND)
       out             SPH,    tmp
       ldi             tmp,    low(RAMEND)
       out             SPL,    tmp
```

```
.endmacro
keslelteto:
	eor		tmp2,	tmp2
	eor		tmp3,	tmp3
	ldi		tmp4,	10
k1:
	dec		tmp2
	brne	k1
;	dec		tmp3
;	brne	k1
	dec		tmp4
	brne	k1
	ret
start:
	init
	ldi		r27,	high(szamok)	;X <- szamok címe
	ldi		r26,	low(szamok)
	eor		tmp,	tmp
	st		X+,		tmp
	inc		tmp
	st		X+,		tmp
	inc		tmp
	st		X+,		tmp
	inc		tmp
	st		X,		tmp
	ldi		r27,	high(szamok)	;X <- szamok címe
	ldi		r26,	low(szamok)
bcd_kiir:
	swap	digit
	ld		tmp6,	X+
	ori		tmp6,	0x80
	or		tmp6,	digit
	out		PORTA,	tmp6
	swap	digit
	call	keslelteto
	push	tmp
	ldi		tmp,	3
	cp		digit,	tmp
	pop		tmp
	breq	bcd_alap
	inc		digit
	rjmp	bcd_kiir

bcd_alap:
	eor		digit,	digit
	ldi		r26,	low(szamok)
	rjmp	bcd_kiir
```

7.7.2 AVR C

7.8 Óra

7.8.1 AVR C

```
#include <avr/io.h>
#include <avr/interrupt.h>

volatile unsigned char dot=1,digit=0,i=255,ertek[4],keyb_row=8, keyb_c;
volatile unsigned char
keyb_num[24]={69,0,14,1,13,2,11,3,22,4,21,5,19,6,38,7,37,8,35,9,67,10,70,11};
volatile unsigned char keyb_but, keyb_p=0;
void init(void);

void main(void)
{
	while(digit<4)ertek[digit++]=digit+1;
	digit=0;
	init();

	while(1)
	{

	}
}

void init(void)
{
	DDRA=0xFF;
	DDRC=0x78;
	DDRB=0xF0;
	DDRD=0xF0;

	TCCR0 =(0<<CS00)|(1<<CS01) |(0<<CS02);  //TIMER0_OVF megszakításhoz az osztási érték
beállítása

	TIMSK |=  (1<<TOIE0);  //TIMER0_OVF engedélyezése
	sei();                 // általános megszakítás engedélyezés

}

void LED_out(unsigned char led)
{
	PORTD=led;
	PORTB=(led<<4);

}

ISR(TIMER0_OVF_vect)
{
	if(!szamlalo){LED_out(LED^0x1);szamlalo=20;}
	else   szamlalo--;

	PORTA=0x80 | digit<<4 | ertek[3-digit];
	if(digit==3) digit=0; else digit++;


	PORTC=keyb_row;
	keyb_but=PINC;
	keyb_row=(keyb_row==0x40)?keyb_row=0x08:keyb_row<<1;
	keyb_c=0;
	while(keyb_c<24)
	{
		if(keyb_but==keyb_num[keyb_c++]) ertek[3-keyb_p]=keyb_num[keyb_c];
		else {if(keyb_c<24)keyb_c++; else keyb_c=0;}

	}
```

```
}
```

7.9 LCD

7.9.1 Assembly

```
.include "m128def.inc"

.CSEG

.def   lcd_com      =      r16
.def   lcd_dat      =      r17
.def   tmp          =      r18
.def   tmp2   =      r19
.def   tmp3   =      r20

.org   0x00
              rjmp   start

.org   0x100

.macro        stack_init
       ldi           tmp,   high(RAMEND)
       out           SPH,   tmp
       ldi           tmp,   low(RAMEND)
       out           SPL,   tmp
.endmacro

.macro EN
       call   delays_ms
       in            tmp,   PINF
       ori           tmp,   0b00001000
       sts           PORTF, tmp
       call   delays_ms
       in            tmp,   PINF
       andi   tmp,   0b11110111
       sts           PORTF, tmp
.endmacro

.macro        port_init
       ldi           tmp,   0b11110000
       out           DDRE,  tmp
       ldi           tmp,   0b00001110
       sts           DDRF,  tmp
.endmacro

lcd_init:
       call   delays_ms
       call   delays_ms
       ldi           lcd_com, 0b00110000
       call   lcd_command
       call   delays_ms
       call   delays_ms
       ldi           lcd_com, 0b00110000
       call   lcd_command
       call   delays_ms
       call   delays_ms
       ldi           lcd_com, 0b00110000
       call   lcd_command
       call   delays_ms
       call   delays_ms
       ldi           lcd_com, 0b00001110
       call   lcd_command
       call   delays_ms
       call   delays_ms
```

```
        ldi         lcd_com, 0x02
        call    lcd_command
        ldi         lcd_com, 0x06
        call    lcd_command
        ldi         lcd_com, 0x01
        call    lcd_command
        ldi         lcd_com, 0x0C
        call    lcd_command
        ret

lcd_command:
        push    tmp
        in          tmp,   PINF
        andi    tmp,   0b11111101
        sts         PORTF, tmp
        mov         tmp,   lcd_com
        andi    tmp,   0xf0;
        out         PORTE, tmp
        EN
        swap    lcd_com
        mov         tmp,   lcd_com
        andi    tmp,   0xf0;
        out         PORTE, tmp
        EN
        swap    lcd_com
        pop         tmp
        ret

lcd_data:
        push    tmp
        in          tmp,   PINF
        ori         tmp,   0b00000010
        sts         PORTF, tmp
        mov         tmp,   lcd_dat
        andi    tmp,   0xf0;
        out         PORTE, tmp
        EN
        swap    lcd_dat
        mov         tmp,   lcd_dat
        andi    tmp,   0xf0;
        out         PORTE, tmp
        EN
        swap    lcd_dat
        pop         tmp
        ret

delays_ms:
        eor         tmp2,  tmp2
        ldi         tmp3,  0x30
k_d:
        dec         tmp2
        brne    k_d
        dec         tmp3
        brne    k_d
        ret

start:
        stack_init
        port_init
        call    lcd_init

;       rjmp    loop

        ldi         ZH,         high(szoveg<<1)
        ldi         ZL,         low(szoveg<<1)
```

```
kiir:
	lpm		lcd_dat,	Z+
	cpi		lcd_dat,	0
	breq	loop
	call	lcd_data
	rjmp	kiir
loop:

	rjmp	loop

szoveg:		.db "Kis Ibolya", 0
```

7.9.2 AVR C

```
// LCD C sample code
#include <avr\io.h>
#include <avr\delay.h>
#define ENABLE {_delay_ms(1); PORTF|=0b00001000;_delay_ms(1);PORTF&=0b11110111;}
void lcd_init(void);
void lcd_data(char adat);
void lcd_cmd(char parancs);
int main()
{
	unsigned char c='1', j=20, a=0, d=0x80;
	lcd_init();
	while(1)
	{
		lcd_data(c++);
		while(j--)_delay_ms(20);
		if(!j) j=10;
		a++;
		if(a>0x0f)
		{
			switch (d)
			{
				case 0x80: d=0xC0; lcd_cmd(d); break;
				case 0xC0: d=0x90; lcd_cmd(d); break;
				case 0x90: d=0xD0; lcd_cmd(d); break;
				case 0xD0: d=0x80; lcd_cmd(d); break;
			}
			a=0;
		}
	}
}
//-------------------------------------------------------------------------------------
void lcd_init(void){
	//-------------------------------------------------------------------------------------
-
	_delay_ms(10);
	DDRF	=	0b00001110;
	DDRE	=	0b11110000;
	lcd_cmd(0b00110000);
	_delay_ms(20);
	lcd_cmd(0b00110000);
	_delay_ms(20);
	lcd_cmd(0b00110000);
	_delay_ms(20);
	lcd_cmd(0b00001110);
	lcd_cmd(0x02);
	lcd_cmd(0x06);
	lcd_cmd(0x01);
	lcd_cmd(0x0C);
	DDRG=0x00;
	DDRB=0xFF;
```

```
}
//-------------------------------------------------------------------------------------
void lcd_cmd(char parancs){
	//-------------------------------------------------------------------------------------
-
	PORTF  &=	0b11111101;
	PORTE  =	parancs&0xF0;
	ENABLE
	parancs	=	(parancs<<4);
	PORTE  =	parancs&0xF0;
	ENABLE
}
//-------------------------------------------------------------------------------------
void lcd_data(char adat){
	//-------------------------------------------------------------------------------------
-
	PORTF |= 0b00000010;
	PORTE  =	adat&0xF0;
	ENABLE
	adat   =	(adat<<4);
	PORTE  =	adat&0xF0;
	ENABLE
}
//-------------------------------------------------------------------------------------
```

7.10 Assembly ZH-ra való felkészüléshez

1. Készítsen menüt, amelyben a G0-G4-ig levő gombokkal választhatja a megvalósítandó feladatot:
G0: Töltse fel az r2 regiszter 12-vel, az r12 regisztert 3-mal! Abban az esetben, ha r2-ben levő érték maradék nélkül osztható r12-vel, akkor LED-ekre írjon 3 értéket, különben pedig 4 értéket!
G1: Töltse fel az adatmemória 0x50 db bájtját a 0x200-as címtől kezdődően páros számokkal 0x10-től kezdődően!
G2: Adja össze az előző feladatrészben feltöltött terület első 10 bájtján levő értéket szubrutin segítségével, és az összeget írassa ki a LED-ekre!
G3: Olvassa be a billentyűzet mátrix 2-es billentyűzetet tartalmazó oszlopának billentyűjének bármelyikét, és írja ki a hétszegmenses kijelző 0. digitjére!
G4: A gomb lenyomásának idejére világítson a 3 színű LED-en a piros szín!

2. Készítsen menüt, amelyben a G0-G4-ig levő gombokkal választhatja a megvalósítandó feladatot:
G0: Töltse fel az r3 regiszter 15-tel, az r4 regisztert 3-mal! Abban az esetben, ha r3-ban levő érték maradék nélkül osztható r4-gyel, akkor a hétszegmenses kijelző 1. digitjére írjon 3 értéket, különben pedig 4 értéket!
G1: Töltse fel az adatmemória 0x50 db bájtját a 0x200-as címtől kezdődően páratlan számokkal 0x10-től kezdődően!
G2: Adja össze az előző feladatrészben feltöltött terület első 10 bájtján levő értéket szubrutin segítségével, és az összeget írassa ki a LED-ekre!
G3: Olvassa be a billentyűzet mátrix 1-es billentyűzetet tartalmazó oszlopának billentyűjének bármelyikét, és írja ki a hétszegmenses kijelző 1. digitjére!
G4: A gomb lenyomásának idejére világítson a 3 színű LED-en a zöld szín!

3. Készítsen menüt, amelyben a G0-G4-ig levő gombokkal választhatja a megvalósítandó feladatot:
G0: Töltse fel az r10 regiszter 12-vel, az r12 regisztert 5-tel! Abban az esetben, ha r10-ben levő érték 4-gyel osztható maradék nélkül és r12 páros, akkor LED-ekre írjon 5 értéket, különben pedig 6 értéket!
G1: Töltse fel az adatmemória 0x50 db bájtját a 0x200-as címtől kezdődően páros számokkal 0xF0-tól kezdődően visszafele!
G2: Adja össze az előző feladatrészben feltöltött terület első 10 bájtján levő értéket szubrutin segítségével, és az összeget írassa ki a LED-ekre!
G3: Olvassa be a billentyűzet mátrix 3-as billentyűzetet tartalmazó oszlopának billentyűjének bármelyikét, és írja ki a hétszegmenses kijelző 2. digitjére!
G4: A gomb lenyomásának idejére világítson a 3 színű LED-en a kék szín!

4. Készítsen menüt, amelyben a G0-G4-ig levő gombokkal választhatja a megvalósítandó feladatot:
G0: Töltse fel az r11 regiszter 32-vel, az r12 regisztert 4-gyel! Abban az esetben, ha r11-ben levő érték 8-cal osztható maradék nélkül és r12 páros, akkor a hétszegmenses kijelző 2. digitjére írjon 7 értéket, különben pedig 8 értéket!
G1: Töltse fel az adatmemória 0x50 db bájtját a 0x200-as címtől kezdődően páratlan számokkal 0xF0-tól kezdődően visszafele!
G2: Adja össze az előző feladatrészben feltöltött terület első 10 bájtján levő értéket szubrutin

segítségével, és az összeget írassa ki a LED-ekre!
G3: Olvassa be a billentyűzet mátrix 7-es billentyűzetet tartalmazó sorának billentyűjének bármelyikét, és írja ki a hétszegmenses kijelző 3. digitjére!
G4: A gomb lenyomásának idejére villogjon a 3 színű LED-en a piros szín!

5. Készítsen menüt, amelyben a G0-G4-ig levő gombokkal választhatja a megvalósítandó feladatot:
G0: Töltse fel az r14 regiszter 32-vel, az r15 regisztert 4-gyel! Abban az esetben, ha r14-ben levő érték 2-vel osztható maradék nélkül és r12 páros, akkor a hétszegmenses kijelző 2. digitjére írjon 7 értéket, különben pedig 8 értéket!
G1: Töltse fel az adatmemória 0x50 db bájtját a 0x200-as címtől kezdődően számokkal 0xF0-tól kezdődően visszafele!
G2: Adja össze az előző feladatrészben feltöltött terület utolsó 10 bájtján levő értéket szubrutin segítségével, és az összeget írassa ki a LED-ekre!
G3: Olvassa be a billentyűzet mátrix 4-es billentyűzetet tartalmazó sorának billentyűjének bármelyikét, és írja ki a hétszegmenses kijelző 3. digitjére!
G4: A gomb lenyomásának idejére villogjon a 3 színű LED-en a zöld szín!

6. Készítsen menüt, amelyben a 0-4-ig levő gombokkal választhatja a megvalósítandó feladatot a mátrix billentyűzeten:
G0: Töltse fel az r2 regiszter 12-vel, az r12 regisztert 3-mal! Abban az esetben, ha r2-ben levő érték maradék nélkül osztható r12-vel, akkor LED-ekre írjon 3 értéket, különben pedig 4 értéket!
G1: Töltse fel az adatmemória 0x50 db bájtját a 0x200-as címtől kezdődően számokkal 0xF0-tól kezdődően visszafele!
G2: Adja össze az előző feladatrészben feltöltött terület utolsó 10 bájtján levő értéket szubrutin segítségével, és az összeget írassa ki a LED-ekre!
G3: Olvassa be a billentyűzet mátrix 4-es billentyűzetet tartalmazó sorának billentyűjének bármelyikét, és írja ki a hétszegmenses kijelző 3. digitjére!
G4: A gomb lenyomásának idejére villogjon a 3 színű LED-en a zöld szín!

7. Készítsen menüt, amelyben a 0-4-ig levő gombokkal választhatja a megvalósítandó feladatot a mátrix billentyűzeten:
G0: Töltse fel az r3 regiszter 15-tel, az r4 regisztert 3-mal! Abban az esetben, ha r3-ban levő érték maradék nélkül osztható r4-gyel, akkor a hétszegmenses kijelző 1. digitjére írjon 3 értéket, különben pedig 4 értéket!
G1: Töltse fel az adatmemória 0x50 db bájtját a 0x200-as címtől kezdődően számokkal 0xF0-tól kezdődően visszafele!
G2: Adja össze az előző feladatrészben feltöltött terület utolsó 10 bájtján levő értéket szubrutin segítségével, és az összeget írassa ki a LED-ekre!
G3: Olvassa be a billentyűzet mátrix 4-es billentyűzetet tartalmazó sorának billentyűjének bármelyikét, és írja ki a hétszegmenses kijelző 3. digitjére!
G4: A gomb lenyomásának idejére villogjon a 3 színű LED-en a zöld szín!

8. Készítsen menüt, amelyben a 0-4-ig levő gombokkal választhatja a megvalósítandó feladatot a mátrix billentyűzeten:
G0: Töltse fel az r10 regiszter 12-vel, az r12 regisztert 5-tel! Abban az esetben, ha r10-ben levő érték 4-gyel osztható maradék nélkül és r12 páros, akkor LED-ekre írjon 5 értéket, különben

pedig 6 értéket!
G1: Töltse fel az adatmemória 0x50 db bájtját a 0x200-as címtől kezdődően számokkal 0x200-tól kezdődően visszafele!
G2: Adja össze az előző feladatrészben feltöltött terület utolsó 10 bájtján levő értéket szubrutin segítségével, és az összeget írassa ki a LED-ekre!
G3: Olvassa be a billentyűzet mátrix 4-es billentyűzetet tartalmazó sorának billentyűjének bármelyikét, és írja ki a hétszegmenses kijelző 3. digitjére!
G4: A gomb lenyomásának idejére villogjon a 3 színű LED-en a kék szín!

9. Készítsen menüt, amelyben a mátrix billentyűzet 0-4-ig levő gombokkal választhatja a megvalósítandó feladatot:
G0: Töltse fel az r11 regiszter 32-vel, az r12 regisztert 8-cal! Abban az esetben, ha r11-ben levő érték 8-cal osztható maradék nélkül és r12 páros, akkor a hétszegmenses kijelző 2. digitjére írjon 7 értéket, különben pedig 8 értéket!
G1: Töltse fel az adatmemória 0x50 db bájtját a 0x200-as címtől kezdődően számokkal 0xF0-tól kezdődően visszafele!
G2: Adja össze az előző feladatrészben feltöltött terület utolsó 10 bájtján levő értéket szubrutin segítségével, és az összeget írassa ki a LED-ekre!
G3: Olvassa be a billentyűzet mátrix 7-es billentyűzetet tartalmazó sorának billentyűjének bármelyikét, és írja ki a hétszegmenses kijelző 2. digitjére!
G4: A gomb lenyomásának idejére villogjon a 3 színű LED-en a zöld szín!

10. Készítsen menüt, amelyben a 0-4-ig levő gombokkal választhatja a megvalósítandó feladatot a mátrix billentyűzeten:
G0: Töltse fel az r14 regiszter 32-vel, az r15 regisztert 4-gyel! Abban az esetben, ha r14-ben levő érték 2-vel osztható maradék nélkül és r12 páros, akkor a hétszegmenses kijelző 2. digitjére írjon 7 értéket, különben pedig 8 értéket!
G1: Töltse fel az adatmemória 0x50 db bájtját a 0x200-as címtől kezdődően számokkal 0x200-tól kezdődően visszafele!
G2: Adja össze az előző feladatrészben feltöltött terület utolsó 10 bájtján levő értéket szubrutin segítségével, és az összeget írassa ki a LED-ekre!
G3: Olvassa be a billentyűzet mátrix 4-es billentyűzetet tartalmazó sorának billentyűjének bármelyikét, és írja ki a hétszegmenses kijelző 3. digitjére!
G4: A gomb lenyomásának idejére villogjon a 3 színű LED-en a piros szín!

11. Készítsen menüt, amelyben a G0-G4-ig levő gombokkal választhatja a megvalósítandó feladatot:
G0: Töltse fel az r2 regiszter 12-vel, az r12 regisztert 3-mal! Abban az esetben, ha r2-ben levő érték maradék nélkül osztható r12-vel, akkor LED-ekre írjon 3 értéket, különben pedig 4 értéket!
G1: Adja össze 0x50 db páros szám értéket 0x10 értéktől kezdődően szubrutin segítségével, és az összeget írassa ki a LED-ekre!
G2: Számolja össze 0x100 és 0x200 tartományban hány darab 8-cal maradék nélkül osztható szám van szubrutin segítségével, és az összeget írassa ki a hétszegmenses kijelző 0. digitjére!
G3: Olvassa be a billentyűzet mátrix 2-es billentyűzetet tartalmazó oszlopának billentyűjének bármelyikét, és írja ki a hétszegmenses kijelző 0. digitjére!
G4: A gomb lenyomásának idejére világítson a 3 színű LED-en a piros szín!

12. Készítsen menüt, amelyben a G0-G4-ig levő gombokkal választhatja a megvalósítandó feladatot:
G0: Töltse fel az r3 regiszter 15-tel, az r4 regisztert 3-mal! Abban az esetben, ha r3-ban levő érték maradék nélkül osztható r4-gyel, akkor a hétszegmenses kijelző 1. digitjére írjon 3 értéket, különben pedig 4 értéket!
G1: Számolja össze 0x120 és 0x210 tartományban hány darab 4-gyel maradék nélkül osztható szám van szubrutin segítségével, és az összeget írassa ki a hétszegmenses kijelző 1. digitjére!
G2: Adja össze 0x100 db páratlan szám értéket 0xFF értéktől kezdődően csökkenő értékekkel szubrutin segítségével, és az összeget írassa ki a LED-ekre!
G3: Olvassa be a billentyűzet mátrix 1-es billentyűzetet tartalmazó oszlopának billentyűjének bármelyikét, és írja ki a hétszegmenses kijelző 1. digitjére!
G4: A gomb lenyomásának idejére világítson a 3 színű LED-en a zöld szín!

13. Készítsen menüt, amelyben a G0-G4-ig levő gombokkal választhatja a megvalósítandó feladatot:
G0: Töltse fel az r10 regiszter 12-vel, az r12 regisztert 5-tel! Abban az esetben, ha r10-ben levő érték 4-gyel osztható maradék nélkül és r12 páros, akkor LED-ekre írjon 5 értéket, különben pedig 6 értéket!
G1: Számolja össze 0x210 és 0x120 tartományban hány darab 16-tal maradék nélkül osztható szám van szubrutin segítségével, és az összeget írassa ki a hétszegmenses kijelző 2. digitjére!
G2: Adja össze 0x100 db páros szám értéket 0xFF értéktől kezdődően csökkenő értékekkel szubrutin segítségével, és az összeget írassa ki a LED-ekre!
G3: Olvassa be a billentyűzet mátrix 3-as billentyűzetet tartalmazó oszlopának billentyűjének bármelyikét, és írja ki a hétszegmenses kijelző 2. digitjére!
G4: A gomb lenyomásának idejére világítson a 3 színű LED-en a kék szín!

14. Készítsen menüt, amelyben a mátrix billentyűzet 0-4-ig levő gombokkal választhatja a megvalósítandó feladatot:
G0: Töltse fel az r11 regiszter 32-vel, az r12 regisztert 8-cal! Abban az esetben, ha r11-ben levő érték 8-cal osztható maradék nélkül és r12 páros, akkor a hétszegmenses kijelző 2. digitjére írjon 7 értéket, különben pedig 8 értéket!
G1: Számolja össze 0x210 és 0x120 tartományban hány darab 32-vel maradék nélkül osztható szám van szubrutin segítségével, és az összeget írassa ki a hétszegmenses kijelző 3. digitjére!
G2: Adja össze 0x100 db páratlan szám értéket 0x10 értéktől kezdődően növekvő értékekkel szubrutin segítségével, és az összeget írassa ki a LED-ekre!
G3: Olvassa be a billentyűzet mátrix 7-es billentyűzetet tartalmazó sorának billentyűjének bármelyikét, és írja ki a hétszegmenses kijelző 2. digitjére!
G4: A gomb lenyomásának idejére villogjon a 3 színű LED-en a zöld szín!

15. Készítsen menüt, amelyben a 0-4-ig levő gombokkal választhatja a megvalósítandó feladatot a mátrix billentyűzeten:
G0: Töltse fel az r14 regiszter 32-vel, az r15 regisztert 4-gyel! Abban az esetben, ha r14-ben levő érték 2-vel osztható maradék nélkül és r12 páros, akkor a hétszegmenses kijelző 2. digitjére írjon 7 értéket, különben pedig 8 értéket!
G1: Számolja össze szubrutin segítségével a 0x210 és 0x120 tartományban hány darab olyan szám van, amelyik esetén a 2. és a 3. biten 1 érték van, és az összeget írassa ki a hétszegmenses kijelző 0. digitjére!
G2: Adja össze 0x100 db páratlan szám értéket 0x10 értéktől kezdődően csökkenő értékekkel szubrutin segítségével, és az összeget írassa ki a LED-ekre!

G3: Olvassa be a billentyűzet mátrix 4-es billentyűzetet tartalmazó sorának billentyűjének bármelyikét, és írja ki a hétszegmenses kijelző 3. digitjére!
G4: A gomb lenyomásának idejére villogjon a 3 színű LED-en a piros szín!

7.11 AVR C ZH-ra való felkészüléshez

1. Készítsen két műveletes számológépet, amely 1, 4, 7 számokkal végez összeadást vagy szorzást, ahol az összeadás műveletét # jel, a szorzás műveletét a * szolgálja. Az operanduszok bevitelére használja a mátrix billentyűzetet. Az első operandus a 3. digiten a műveleti jel a LED-eken, a második operandus a 2. digiten, az eredmény a 0. és 1. digiteken jelenjen meg a hétszegmenses kijelzőn:
G0: Első operandus bevitele (bevitelre csak 1,4,7 billentyű használható).
G1: Műveleti jel bevitele (bevitelre csak a *, # billentyűk használhatók).
G2: Második operandus bevitele (bevitelre csak 1,4,7 billentyű használható).
G3: Eredmény megjelenítése vezető nulla kikapcsolásával.
G4: A gomb lenyomásának idejére villogjon 1 másodperces ütemben a 3 színű LED-en a piros szín!

2. Készítsen két műveletes számológépet, amely 2, 5, 8 számokkal végez kivonást vagy szorzást, ahol a kivonás műveletét # jel, a szorzás műveletét a * szolgálja. Az operanduszok bevitelére használja a mátrix billentyűzetet. Az első operandus a 3. digiten a műveleti jel a LED-eken, a második operandus a 2. digiten, az eredmény a 0. és 1. digiteken jelenjen meg a hétszegmenses kijelzőn:
G0: Első operandus bevitele (bevitelre csak 2, 5, 8 billentyű használható).
G1: Műveleti jel bevitele (bevitelre csak a *, # billentyűk használhatók).
G2: Második operandus bevitele (bevitelre csak 2, 5, 8 billentyű használható).
G3: Eredmény megjelenítése vezető nulla kikapcsolásával.
G4: A gomb lenyomásának idejére villogjon 1 másodperces ütemben a 3 színű LED-en a zöld szín!

3. Készítsen két műveletes számológépet, amely 3, 6, 9 számokkal végez összeadást vagy kivonást, ahol az összeadás műveletét # jel, a kivonás műveletét a * szolgálja. Az operanduszok bevitelére használja a mátrix billentyűzetet. Az első operandus a 3. digiten a műveleti jel a LED-eken, a második operandus a 2. digiten, az eredmény a 0. és 1. digiteken jelenjen meg a hétszegmenses kijelzőn:
G0: Első operandus bevitele (bevitelre csak 3, 6, 9 billentyű használható).
G1: Műveleti jel bevitele (bevitelre csak a *, # billentyűk használhatók).
G2: Második operandus bevitele (bevitelre csak 3, 6, 9 billentyű használható).
G3: Eredmény megjelenítése vezető nulla kikapcsolásával.
G4: A gomb lenyomásának idejére villogjon 1 másodperces ütemben a 3 színű LED-en a kék szín!

4. Készítsen összeadót vagy szorzót, amely csak a 1, 4, 7 és a 3, 6, 9 számjegyekből álló 3 jegyű számmal végez összeadást. Az operanduszok bevitelére használja a mátrix billentyűzetet. Az első operandus, a második operandus és az eredmény jelenjen meg a hétszegmenses kijelzőn:
G0: Első 2 jegyű operandus bevitele (bevitelre csak 3, 6, 9, 1, 4, 7 billentyűk használható).
G1: Műveleti jel bevitele (bevitelre csak a *, # billentyűk használhatók).
G2: Második 2 jegyű operandus bevitele (bevitelre csak 3, 6, 9, 1, 4, 7 billentyűk használható).
G3: Eredmény megjelenítése vezető nulla kikapcsolásával.
G4: A gomb lenyomásának idejére villogjon 1 másodperces ütemben a 3 színű LED-en a kék szín!

5. Készítsen összeadót vagy kivonót, amely csak a 1, 4, 7 és a 3, 6, 9 számjegyekből álló 3 jegyű számmal végez összeadást. Az operanduszok bevitelére használja a mátrix billentyűzetet. Az első operandus, a második operandus és az eredmény jelenjen meg a hétszegmenses kijelzőn:
G0: Első 3 jegyű operandus bevitele (bevitelre csak 3, 6, 9, 1, 4, 7 billentyűk használható).
G1: Műveleti jel bevitele (bevitelre csak a *, # billentyűk használhatók).
G2: Második 3 jegyű operandus bevitele (bevitelre csak 3, 6, 9, 1, 4, 7 billentyűk használható).
G3: Eredmény megjelenítése vezető nulla kikapcsolásával.
G4: A gomb lenyomásának idejére villogjon 1 másodperces ütemben a 3 színű LED-en a piros szín!

6. Készítsen összeadót vagy kivonót, amely csak a 1, 7 és a 6, számjegyekből álló 3 jegyű számmal végez összeadást. Az operanduszok bevitelére használja a mátrix billentyűzetet. Az első operandus, a második operandus és az eredmény jelenjen meg a hétszegmenses kijelzőn:
G0: Első 3 jegyű operandus bevitele (bevitelre csak 1, 6, 7 billentyűk használható).
G1: Műveleti jel bevitele (bevitelre csak a *, # billentyűk használhatók).
G2: Második 3 jegyű operandus bevitele (bevitelre csak 1, 6, 7 billentyűk használható).
G3: Eredmény megjelenítése vezető nulla kikapcsolásával.
G4: A gomb lenyomásának idejére villogjon 1 másodperces ütemben a 3 színű LED-en a zöld szín!

7. Készítsen órát, amely a hétszegmenses kijelzőn a digit 0-án és digit 1-en a percet, a digit 3-on és digit 2-ön az órát mutatja. A másodperc mutatása a digit 2 és digit 1 közötti két LED-en történjen:
2-es osztályzat: Óra és perc kijelzése a hétszegmenses kijelzőn, az óra időzítéséhez késleltető függvényt használhat.
3-as osztályzat: Az előző feladatrész mellett a másodperc kijelzés megjelenítése.
4-es osztályzat: Óra és perc kijelzése a hétszegmenses kijelzőn, az óra időzítéséhez timer interruptot használjon!
5-ös osztályzat: Az előző feladatrész mellett a másodperc kijelzés megjelenítése.

8. Készítsen órát, amely a hétszegmenses kijelzőn a digit 0-án és digit 1-en a másodpercet, a digit 3-on és digit 2-ön a percet mutatja. A másodperc mutatása a digit 2 és digit 1 közötti két LEDen is történjen:
2-es osztályzat: Másodperc és perc kijelzése a hétszegmenses kijelzőn, az óra időzítéséhez késleltető függvényt használhat.
3-as osztályzat: Az előző feladatrész mellett a másodperc kijelzés megjelenítése a digitek közötti LED-eken is.
4-es osztályzat: Másodperc és perc kijelzése a hétszegmenses kijelzőn, az óra időzítéséhez timer interruptot használjon!
5-ös osztályzat: Az előző feladatrész mellett a másodperc kijelzés megjelenítése a digitek közötti LED-eken.

9. Készítsen számrendszerek közötti átszámítót, amely 2, 4, 8 és 10 számrendszerek között átszámítja a billentyűzet mátrixon bevitt és a hétszegmenses kijelzőn megjelenített decimális számértéket:
G0: 2 digites decimális szám bevitele a mátrix billentyűn és megjelenítése a digit 0 és digit 1 kijelzőn.

G1: a bevitt érték oktális megjelenítése a hétszegmenses kijelzőn.
G2: a bevitt érték 4-es számrendszerbeli megjelenítése a hétszegmenses kijelzőn.
G3: a bevitt érték felső bájtjának bináris megjelenítése a LED-eken.
G4: a bevitt érték alsó bájtjának bináris megjelenítése a LED-eken.

10. Készítsen számrendszerek közötti átszámítót, amely 2, 4, 8 és 10 számrendszerek között átszámítja a billentyűzet mátrixon bevitt és a hétszegmenses kijelzőn megjelenített oktális számértéket:
G0: 2 digites oktális szám bevitele a mátrix billentyűn és megjelenítése a digit 0 és digit 1 kijelzőn.
G1: a bevitt érték deicmális megjelenítése a hétszegmenses kijelzőn.
G2: a bevitt érték 4-es számrendszerbeli megjelenítése a hétszegmenses kijelzőn.
G3: a bevitt érték felső digitjének bináris megjelenítése a LED-eken.
G4: a bevitt érték alsó digitjének bináris megjelenítése a LED-eken.

11. Készítsen reakció időmérőt, amely G0-ás gomb hatására bekapcsolja a LED0-át, és elindítja a hétszegmenses kijelzőn a századmásodperc futását, majd a G1 gomb hatására ezt leállítja a számlálást:
2-es osztályzat: az időmérő megoldása késleltető függvény segítségével.
3-as osztályzat: az előző feladat megoldása, 5-ször ismételt mérés átlagát kijelezve a hétszegmenses kijelzőn.
4-es osztályzat: az időmérő megoldása timer interrupt segítségével történjen.
5-ös osztályzat: az előző feladat megoldása, 5-ször ismételt mérés átlagát kijelezve a hétszegmenses kijelzőn.

12. Készítsen gyakoriság számlálót, amely a mátrix billentyűn bevitt 10 db számjegy közül kiírja a leggyakrabban bevitt számjegyet, illetve annak a gyakoriságát:
2-es osztályzat: 10 db számjegy bevitele a mátrix billentyűn, majd G0 hatására a digit 2-ön a leggyakoribb szám megjelenítése.
3-as osztályzat: az előző feladat megoldása, és digit 0-án és digit 1-en a leggyakrabban előforduló számjegy gyakoriságának kiíratása.
4-es osztályzat: az előző feladat megoldása, a bevitt számjegyek átlagát kijelezve a hétszegmenses kijelzőn.
5-ös osztályzat: az előző feladat megoldása, de az időzítések megoldása timer interrupt segítségével történjen.

Megszakítás vektortábla

Vector No.	Program Address[2]	Source	Interrupt Definition
1	$0000[1]	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$0002	INT0	External Interrupt Request 0
3	$0004	INT1	External Interrupt Request 1
4	$0006	INT2	External Interrupt Request 2
5	$0008	INT3	External Interrupt Request 3
6	$000A	INT4	External Interrupt Request 4
7	$000C	INT5	External Interrupt Request 5
8	$000E	INT6	External Interrupt Request 6
9	$0010	INT7	External Interrupt Request 7
10	$0012	TIMER2 COMP	Timer/Counter2 Compare Match
11	$0014	TIMER2 OVF	Timer/Counter2 Overflow
12	$0016	TIMER1 CAPT	Timer/Counter1 Capture Event
13	$0018	TIMER1 COMPA	Timer/Counter1 Compare Match A
14	$001A	TIMER1 COMPB	Timer/Counter1 Compare Match B
15	$001C	TIMER1 OVF	Timer/Counter1 Overflow
16	$001E	TIMER0 COMP	Timer/Counter0 Compare Match
17	$0020	TIMER0 OVF	Timer/Counter0 Overflow
18	$0022	SPI, STC	SPI Serial Transfer Complete
19	$0024	USART0, RX	USART0, Rx Complete
20	$0026	USART0, UDRE	USART0 Data Register Empty
21	$0028	USART0, TX	USART0, Tx Complete
22	$002A	ADC	ADC Conversion Complete
23	$002C	EE READY	EEPROM Ready
24	$002E	ANALOG COMP	Analog Comparator
25	$0030[3]	TIMER1 COMPC	Timer/Countre1 Compare Match C
26	$0032[3]	TIMER3 CAPT	Timer/Counter3 Capture Event
27	$0034[3]	TIMER3 COMPA	Timer/Counter3 Compare Match A
28	$0036[3]	TIMER3 COMPB	Timer/Counter3 Compare Match B
29	$0038[3]	TIMER3 COMPC	Timer/Counter3 Compare Match C
30	$003A[3]	TIMER3 OVF	Timer/Counter3 Overflow

Vector No.	Program Address[2]	Source	Interrupt Definition
31	$003C[3]	USART1, RX	USART1, Rx Complete
32	$003E[3]	USART1, UDRE	USART1 Data Register Empty
33	$0040[3]	USART1, TX	USART1, Tx Complete
34	$0042[3]	TWI	Two-wire Serial Interface
35	$0044[3]	SPM READY	Store Program Memory Ready