Mikrovezérlők alkalmazástechnikája laboratóriumi gyakorlat

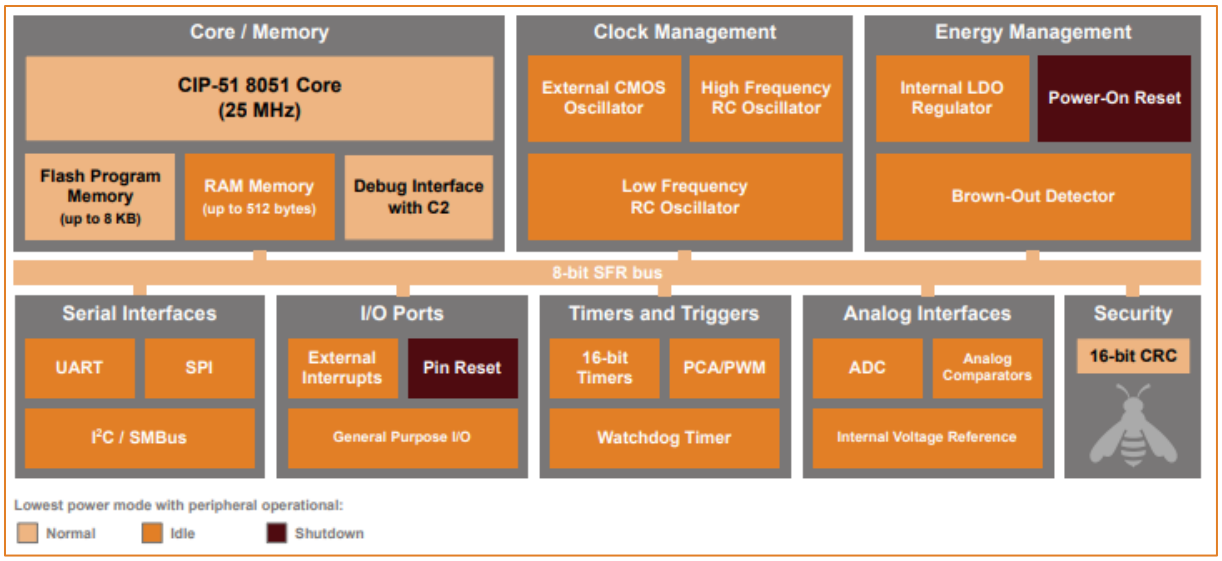
EFM8BB1LCK

Útmutató a fejlesztőkörnyezet használatához

EFM8BB1LCK MCU

A Silicon Labs által kiadott 8-bites, onboard debuggerel rendelkező mikrovezérlőt fogjuk használni a gyakorlat során (https://www.silabs.com/development-tools/mcu/8-bit/efm8bb1lck-starter-kit).

A mikrovezérlőben CIP-51-es core van, aminek utasításkészlete teljes mértékben kompatibilis a korábban használt 8051-gyel (F410), a maximum működési frekvenciája 25 MHz. A mikrovezérlő tartalmaz 4 darab 16 bites timert, analóg és digitális perifériákat, valamint 18 db multifunkcionális GPIO-t. Órajel forrásból több lehetőség is rendelkezésünkre áll, ezek közül 2 belső és egy külső forrásunk van. A két belső forrásból az egyik egy nagy frekvencián működő oszcillátor 24.5MHz, a másik pedig egy alacsony frekvencián működő (80 kHz). Egy 12 bites ADC is rendelkezésünkre áll, ami 12 bites módban 200 ksps mintavételezésre képes. Egy integrált hőmérsékletszenzort is megtalálhatunk a boardon. Kommunikációra használhatunk UART-ot, SPI-t, illetve I^2c-t is.



1. ábra: EFM8BB1LCK fontosabb tulajdonságai [1]

Csatlakoztatáshoz az onboard debugger miatt elég egy micro USB kábelt használnunk, az IDE felismeri és beállítja a debuggert.

Reference manual: https://www.silabs.com/documents/public/reference-manuals/efm8bb1-rm.pdf

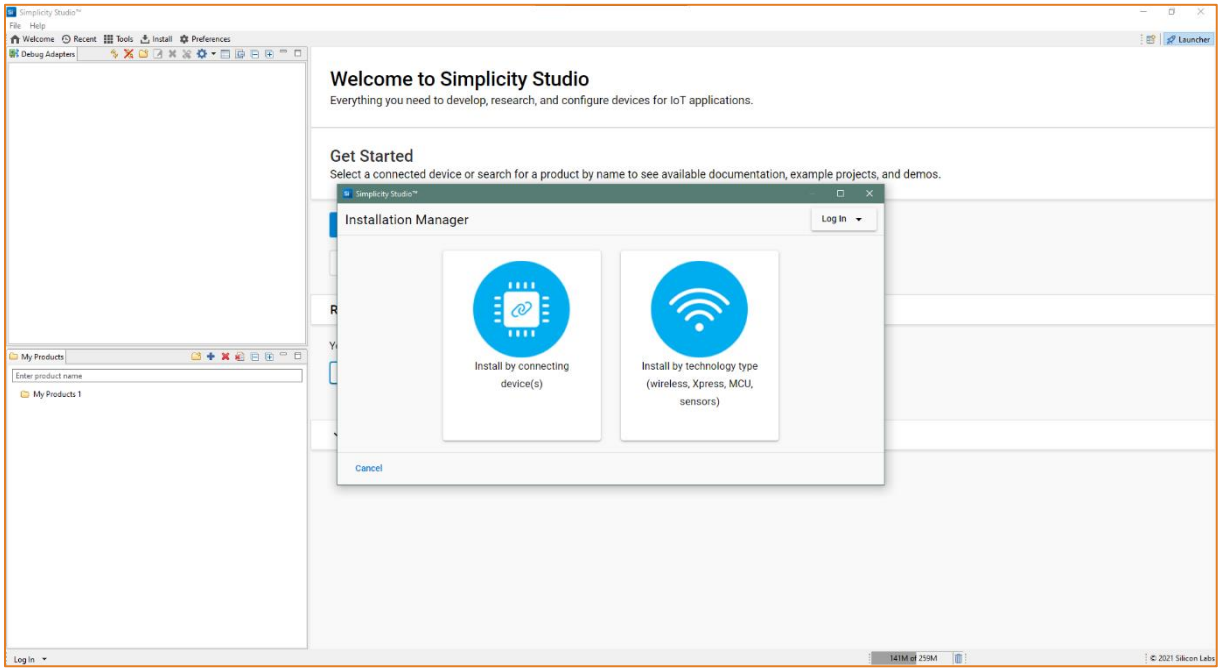
Adatlap: https://www.silabs.com/documents/public/data-sheets/efm8bb1-datasheet.pdf

User guide: https://www.silabs.com/documents/public/user-guides/ug377-efm8bb1lck.pdf

Simplicity Studio 5

A gyakorlaton a Silicon labs által kiadott új IDE-t a Simplicity Studio 5-öt fogjuk használni, ami regisztrációt követően letölthető az alábbi linken keresztül: https://www.silabs.com/developers/simplicity-studio.

A program letöltés és telepítés után bejelentkezés nélkül is használható. Elindítva az alábbi felülettel találkozhatunk:

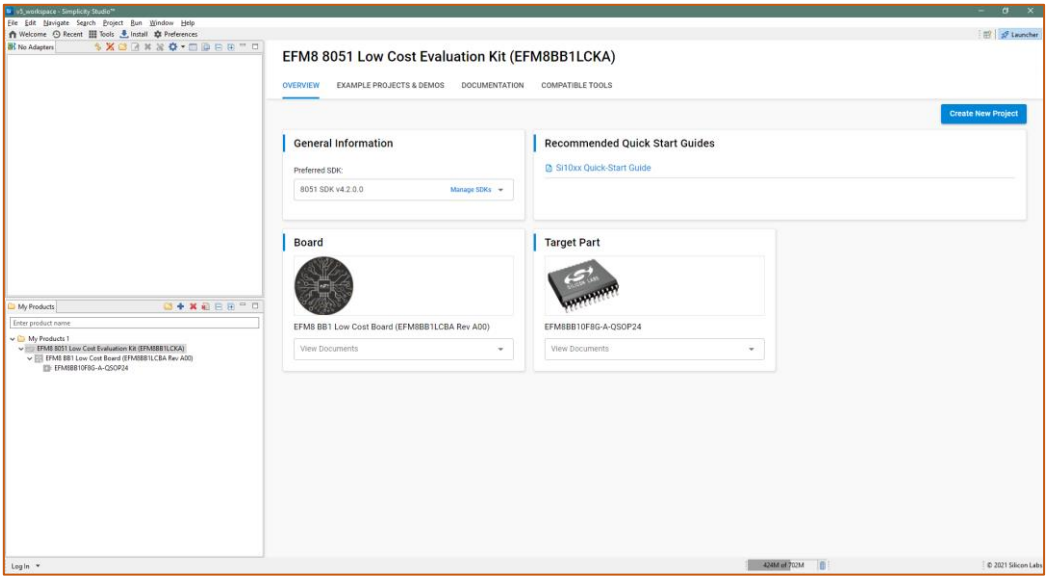


2. ábra: Telepítés utáni képernyő

Ha rendelkezünk a mikrovezérlővel akkor válasszuk az első lehetőséget, az eszköz csatlakoztatásával automatikusan beállítja a típust. Amennyiben nincs hardverünk válasszuk a második lehetőséget és a 8-bites MCU-t. Ez után mindent telepítsünk fel. Köszönhetően az IDE felhasználóbarát környezetének más dolgunk nincs.

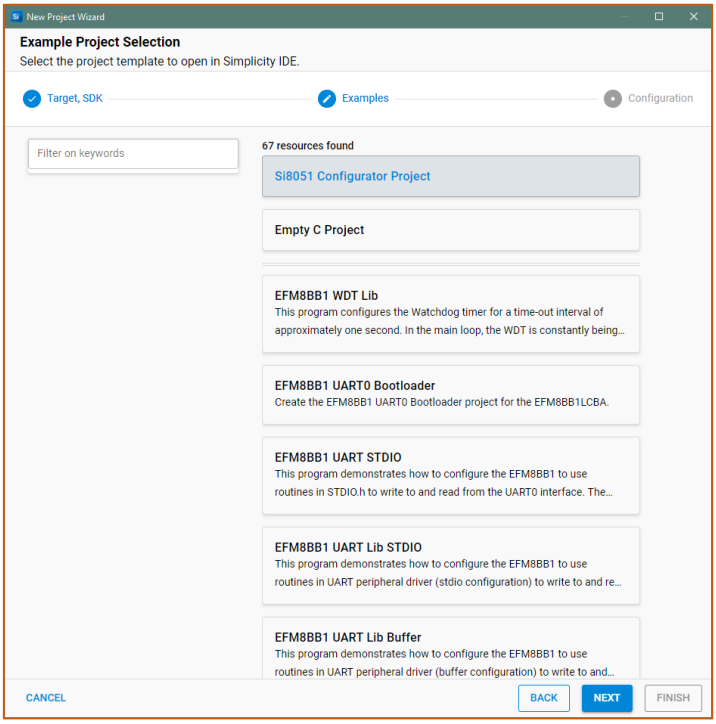
Projekt létrehozása

Projekt létrehozásához egyszerűen válasszuk ki a főoldalon az eszközt a *Connected devices* menüpontból, ha nincs fizikai eszközünk válasszuk ki az *All products* az alábbi kitet: EFM8 8051 Low Cost Evaluation Kit (EFM8BB1LCKA).



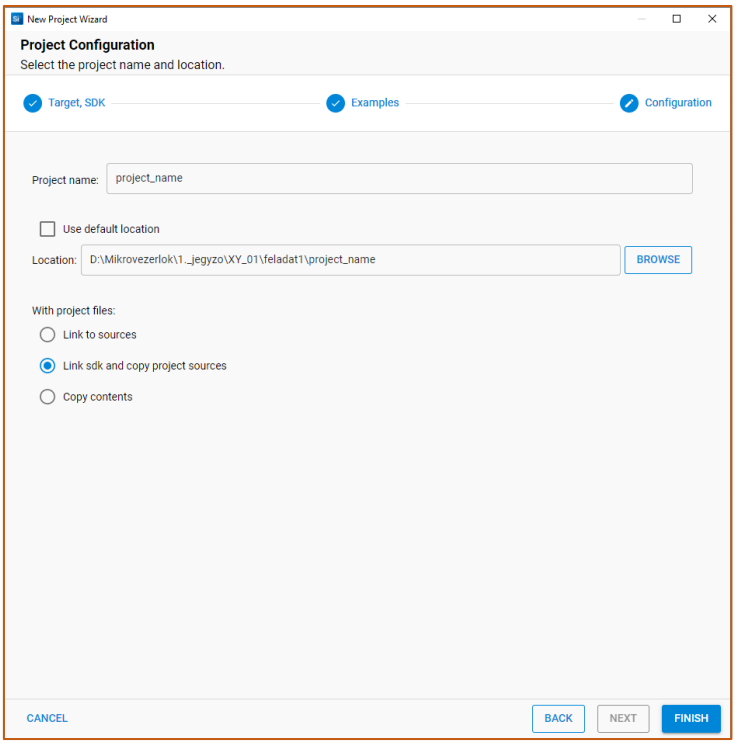
3. ábra: Projekt létrehozása | Mikrovezérlő típusának kiválasztása

Projekt létrehozásánál válasszuk az első lehetőséget:



4. ábra: Üres projekt létrehozása

Az elnevezésnél kerüljük az ékezeteket. Minden óra anyagnak egy külön mappát hozzunk létre azon belül pedig almappákban legyenek az egyes feladatokhoz tartozó projektek.



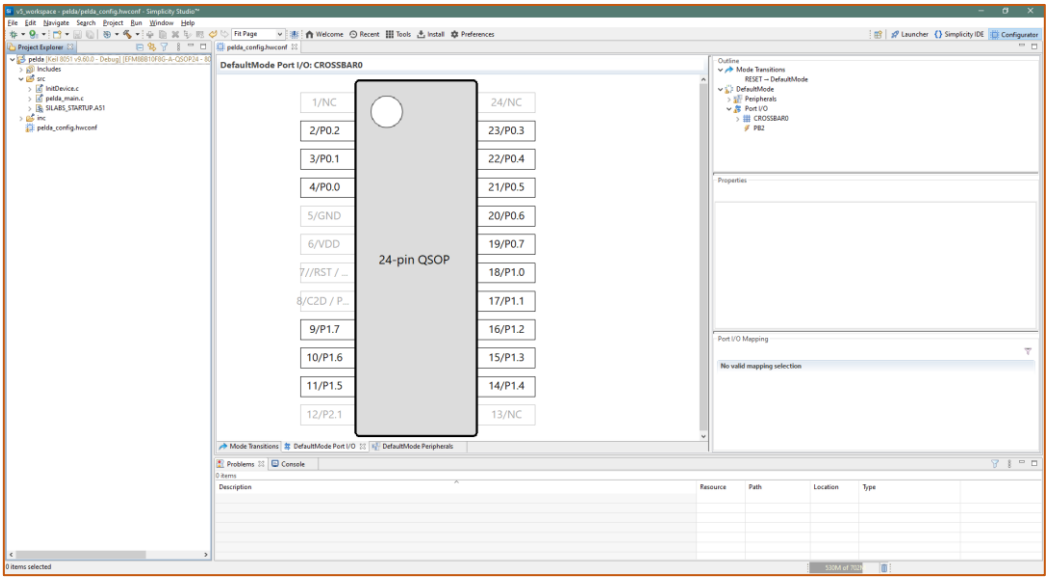
5. ábra: Projekt létrehozásának befejezése

Projektjeinkben használhatjuk a „Link sdk and copy project sources”, beállítást, ekkor csak azok a fájlok lesznek átmásolva, amiket biztosan módosítanánk, a könyvtárak linkelésével későbbiekben könnyebben áttérhetünk másik SDK verzióra is. Ha bluetooth-os projektünk lenne ott ajánlott mindent átmásolni (Copy contents).

Ha az adott gyakorlaton több feladat is van, akkor az a legegyszerűbb, ha a Windowsos fájlkezelőben másolgatjuk és átnevezzük a már meglévő projekt mappáját az aktuális feladat sorszáma szerint. Az új mappát importálhatjuk az IDE-be a Project Explorer/Import/MCU project menüben. A tallózás után figyeljünk oda, hogy ugyanaz a mappa legyen kiválasztva a Location mezőben, amit betallóztunk (ehhez lehet, hogy ki kell venni a pipát a Use default location checkboxból) és csak ezután kattintsunk a Finish gombra.

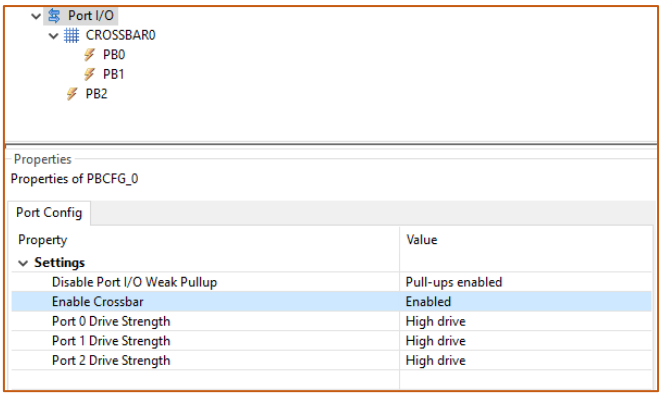
Mikrovezérlő konfigurálása

Ha helyesen hoztuk létre a projektet megjelenik egy *hwconf* kiterjesztésű fájl (ezt szabadon elnevezhetjük). Segítségével konfigurálni tudjuk a mikrovezérlőnket.



6. ábra: Config wizard | PORT I/O

Itt láthatjuk, a konfigurálható I/O lábakat, itt az adott lábat kiválasztva beállíthatjuk annak paramétereit. Figyeljünk arra, hogy csak azokat a lábakat konfiguráljuk, amiket használni szeretnénk, és hozzájuk helyesen csatoljuk a perifériákat. Ha kimenetnek szeretnénk használni az adott portot, akkor kapcsoljuk Push-Pull módba (pl.: LED meghajtása), viszont, ha egy bemenetnek szeretnénk, akkor Open-Drain módban kell lennie (pl.: gomb használata). Crossbar-t engedélyezni kell, ezt elérhetjük a Port I/O menüben:



7. ábra: Crossbar bekapcsolása

Mivel a gyakorlaton nem használjuk a Watchdog-timert, ezért azt kapcsoljuk ki: DefaultMode Peripherals / Timers /Watchdog Timer -ben. Így nem fogja újraindítani a mikrovezérlőnket. Nagyon hasznos, hogy a konfigurátor figyelmeztető kis sárga háromszöggel jelzi, ha valami még nincs jól beállítva, pl. valamit még engedélyezni kell ahhoz, hogy a használni kívánt periféria megfelelően működjön. Ha hiba van a beállításokban azt egy kis piros X-szel jelzi. Egy tipikus ilyen eset, ha elindítunk egy Timert, de nem pipáljuk be a Timers perifériát 

. Fontos: ha

egy periféria nincs bepipálva, akkor nem generálja le a hozzá tartozó forráskódot a konfigurátor. A konfigurátor akkor generálja le a forráskódot, ha elmentjük a változtatásokat.

A generált forrásfájlokban a következő kommentek között lévő részben a generált kódrészek találhatók. Ide ne írjunk saját kódot, mert a következő generálás során a konfigurátor felülírja az itt lévő kódot.

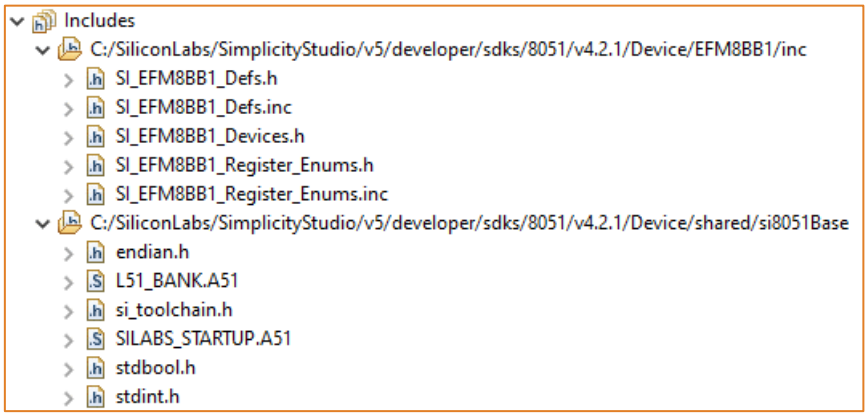
```
// $[Generated Includes]
// [Generated Includes]$
```

A konfigurátor hasznos tulajdonsága még, hogy legenerál egy *Interrupts.c* fájlt. Ide fognak kerülni az interrupt-handler függvényeink. A függvények akkor generálódnak le, ha az Core/Interrupts menüben engedélyezzük a *Code Generation*-t, illetve bekapcsoljuk a használni kívánt interrupt-ot.

Kis megszokást igényel, hogy egy beállítás módosításakor ténylegesen csak akkor veszi a módosítást a konfigurátor, ha kikattintunk a cellából vagy, ha entert nyomunk.

Fontosabb fájlok és makrók

Projekt létrehozásakor az IDE include-ol számos fontos fájlt, ami nélkül nehezebb lenne a mikrovezérlőnk programozása. Előre megírták a mikrovezérlőnk regisztereinek címeit, illetve néhány hasznos függvény is, mint például a regiszterek címzéséhez használható SI függvényeket (SI_SBIT, SI_SFR).



8. ábra: Csatolt fájlok

SI_EFM8BB1_Defs.h headerfájlban a MCU regiszter definícióit láthatjuk, illetve indirekt módon azok bitjei is definiálva vannak. Tartalmazza például a Timerek, Portok és azok pin latch-eit, ADC és Interrupts deklarációkat.

SI_EFM8BB1_Register_Enums.h fájlban megtalálhatjuk a regiszterek értékeinek változtatásához hasznos definíciókat. Ezek segítségével olvashatóbb lesz a kódunk és egyszerűbben tudjuk a regiszterek értékeit változtatni a bitműveletek segítségével.

A második mappában láthatjuk a si_toolchain.h headerfájlt, amiben makrókat találhatunk a KEIL toolchainhez, olyan fontos funkciók találhatók meg ebben a fájlban, mint a megszakításkezelés, változó és pointer deklarációk.

I/O kezelés

A portokat elérhetjük bitenként és bájtonként is (P1_B0, P1). A korábban használt F410-es mikrovezérlő és az SDCC fordító esetén egy #define -ba volt kiszervezve:

```
#define LED0 P1_B4 .
```

Viszont használhatjuk az `SI_SBIT(name, address, bitnum)`, függvényt, ami 3 paramétert vár:

name -> a változónk neve lesz,

address -> a byte címe, ami tartalmazza az adott bitet

bitnum -> egy bit számát adja meg a byte-on belül. (0…7 közötti szám)

Ennek segítségével sokkal átláthatóbb lesz a kódunk.

Nézzük meg a fenti példát ezzel a módszerrel:

```
SI_SBIT (LED0, SFR_P1, 4U);
```

Mi történik a háttérben?

```
SI_SBIT(name, address, bitnum)
{
    sbit name = address ^ bitnum;
}
```

Tehát ha a fenti paraméterekkel meghívjuk, akkor:

```
sbit LED0 = 0x90^4U;
```

Itt egy XOR műveletet hajtunk végre, ami kiválasztja az adott bitet a regiszter bitjei közül és az értékét az ellenkezőjére változtatja (cím megváltozik). Az SFR regiszterek közül vannak bit és byte címezhető regiszterek, a bit címezhető regiszterek, azok, amelyek címe 0x0 vagy 0x8-ra végződik. A nem ilyen végződésű regisztereknél csak a bytecímzést használhatjuk. Beágyazott rendszereknél bitműveleteket használnak az egyes regiszterek bitjeinek értékének állítására.

Bitműveletek a regiszterek értékének állítására

Példának nézzük meg az interrupt enable regisztert. Ez a regiszter 8-bites, és minden bitet lehet olvasni (R) és írni (W). Az adatlapot megnézve láthatjuk, hogy a bit-fieldek milyen célt szolgálnak. A gyártók ezeknek a regisztereknek készítenek headerfileokat, itt defineokban megadnak olyan hasznos dolgokat, mint a bitmask, vagy az, hogy mennyivel kell shiftelnünk az adott értéket, hogy az a regiszter megfelelő bitét állítsa be (bitpozíció).

6.3.1 IE: Interrupt Enable

Bit	7	6	5	4	3	2	1	0
Name	EA	ESPI0	ET2	ES0	ET1	EX1	ET0	EX0
Access	RW	RW	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0
SFR Address: 0xA8 (bit-addressable)								

9. ábra: Interrupt control registers (Interrupt Enable) [1]

```
805 //-----------------------------------------------------------------------------
806 // IE Enums (Interrupt Enable @ 0xA8)
807 //-----------------------------------------------------------------------------
808 #define IE_EX0__BMASK      0x01 ///< External Interrupt 0 Enable
809 #define IE_EX0__SHIFT      0x00 ///< External Interrupt 0 Enable
810 #define IE_EX0__DISABLED   0x00 ///< Disable external interrupt 0.
811 #define IE_EX0__ENABLED    0x01 ///< Enable interrupt requests generated by the INT0
812                                 ///< input.
813
814 #define IE_ET0__BMASK      0x02 ///< Timer 0 Interrupt Enable
815 #define IE_ET0__SHIFT      0x01 ///< Timer 0 Interrupt Enable
816 #define IE_ET0__DISABLED   0x00 ///< Disable all Timer 0 interrupt.
817 #define IE_ET0__ENABLED    0x02 ///< Enable interrupt requests generated by the TF0
818                                 ///< flag.
```

10. ábra: Interrupt Enable regiszter headerfileban található definíciója

N-edik bit kiolvasása egy regiszterből:

```
bit_value = ([regiszter] >> n) & 1U;
```

Vegyük példának, hogy az IE regiszter ET0 fieldjében egy 1-es van. Ekkor IE regiszter értéke: 0000010b. Ha a fenti módszert használjuk kiolvashatjuk, hogy mi az adott regiszterben lévő bitnek az értéke:

```
   0000 0010
>>        1   (jobbra tolás n-el)
   0000 0001
&  0000 0001   (1 éselése az eltolt értékhez)
   0000 0001   (adott bit értéke)
```

Ezt kódban a registers_enum.h-t felhasználva az alábbi módon valósíthatjuk meg:

```
bit_value = (IE >> IE_ET0__SHIFT) & 1U;
```

N-edik bit törlése egy regiszterből:

```
registzter &= ~(1U << n);
```

Az előző példában kiolvasott bitet most töröljük ki, annyi változással, hogy most az EA field is aktív. Tehát törlés előtt IE regiszter értéke 1000010b.

	0000 0001	(1U binárisan)
<<	1	(balra tolás n-el)
~	0000 0010	(érték negálása)
	1111 1101	
&	1000 0010	(éselés a regiszter értékkel)
	1000 0000	

Ezt kódban a registers_enum.h-t felhasználva az alábbi módon valósíthatjuk meg:

```
IE &= (1U << IE_ET0__SHIFT);
```

N-edik bit 1-re állítása egy regiszterben:

```
registzter |= (1U << n);
```

Törlés után a regiszterünk értéke most 1000000b. Kapcsoljuk be az EX0-át:

	0000 0001	(1U binárisan)
<<	0	(balra tolás n-el)
	0000 0001	
|	1000 0000	(vagyolás a regiszter értékével)
	1000 0001	

Ezt kódban a registers_enum.h-t felhasználva az alábbi módon valósíthatjuk meg:

```
IE |= (1U << IE_EX0__SHIFT);
```

N-edik bit kapcsolása egy regiszterben (0 -> 1 , 1 -> 0) :

```
registzter ^= (1U << n);
```

XOR művelet segítségével kapcsolni tudjuk egy regiszter bitét, kapcsoljuk EX0-át:

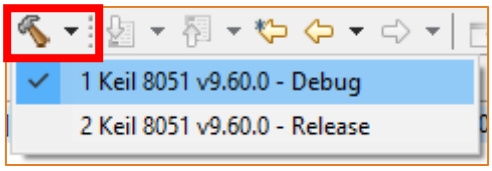
	0000 0001	(1U binárisan)
<<	0	(balra tolás n-el)
	0000 0001	
^	1000 0001	(XOR a regiszter értékével)
	1000 0000	

Ezt kódban a registers_enum.h-t felhasználva az alábbi módon valósíthatjuk meg:

```
IE ^= (1U << IE_EX0__SHIFT);
```

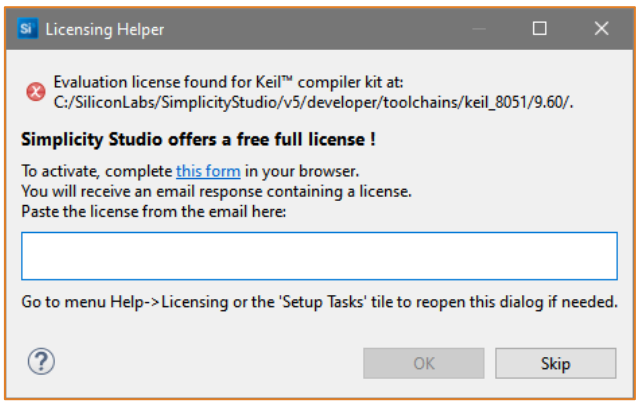
Kód fordítása és futtatása

Lehetőség van a kódot lefordítani ezt a menüsorban található build ikonra kattintva tehetjük meg:



11. ábra: Kód buildelésének elindítása

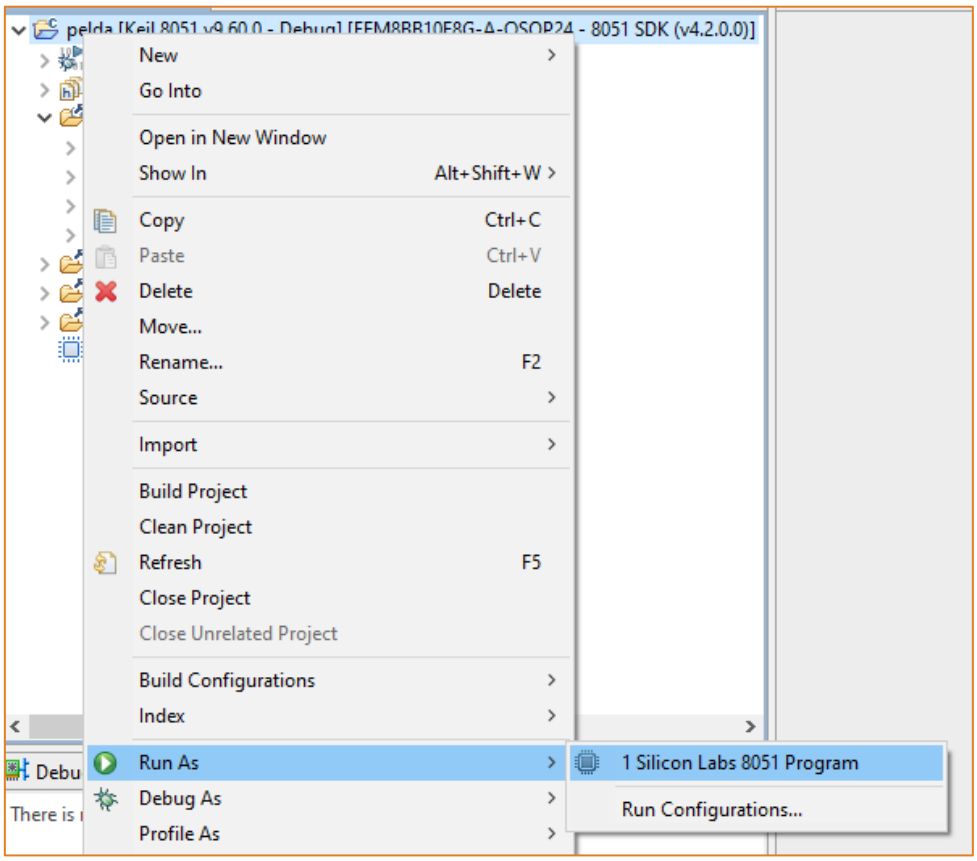
Buildeléshez kérni fogja a program, hogy regisztráljunk egy oldalon, így megkapjuk a KEIL licenszet:



12. ábra: Licensz aktiválása

A linkre kattintva a KEIL oldalán egy formot kitöltve az ott megadott emailre elküldik az aktiváló kódot. Ez a kód egy felhasználó/PC-re használható. Ha nem aktiváljuk akkor is lefordul a kódunk, viszont korlátozás van megadva a kódméretre, viszont viszonylag nagy kódot is le lehet így fordítani. (Ha a szerverek leterheltek akár 1 nap is lehet a licensz megérkezése).

A kód futtatásához csatlakoztassuk a mikrovezérlőt, és a projektünk mappájára kattintva elérhetjük a „Run As” menüpontot:



13. ábra: Kód futtatása a mikrovezérlőn

Ilyenkor lebuildeli a kódunkat és feltölti a mikrovezérlőre, viszont a debug mód nem aktív, van lehetőség debug módban is futtatni, ezt a *Debug As* menün keresztül tehetjük meg.

Kód debugolása

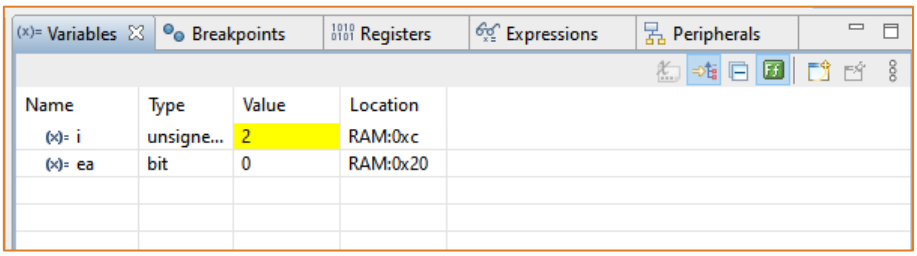
A debug mode-ot érdemes használni a sima „Run as”, helyett, mivel itt jobban kontrollálhatjuk a mikrovezérlőnket, az esetleges hibákat visszakövethetjük, figyelemmel kísérhetjük a kódunk futása közben a regiszterek értékét.

Ikon	Leírás
	[Debug] gomb, elindítja a debugolást a kiválasztott MCU-n, egyszerre egy debug session futhat.
	[Skip All Breakpoints] gomb ignorálja a beállított breakpointsokat
	[Resume] gomb elindítja a kódot, ha az egy reset miatt leállt volna, vagy egy breakpointot elért, folyamatos futást eredményez amíg nem ütközik breakpointba
	[Suspend] gomb felfüggeszti a mikrovezérlőn futó kódot.
	[Terminate] gomb leállítja a debugolást
	[Disconnect] gomb megszakítja a jelenleg futó debugolási szekciót és lecsatlakoztatja a MCU-t
	[Reset the Device] gomb hardware resetet generál, újraindítja a MCU-t
	[Step Into] gomb belép a függvény első sorába és soronként halad
	[Step Over] gomb átlép a függvényen és végrehajtja azt
	[Step Return] gomb kilép a függvényből (return point) és végrehajtja azt
	[Instruction Stepping Mode] gomb szintén egyesével lépked, viszont itt az assembly utasításokon, ezt egy külön disassembly nézeten követhetjük

Ha a kódunkban szeretnénk töréspontot elhelyezni, akkor azt az oldalsó sorszámra duplán kattintva tehetjük meg, ilyenkor a futás során a vezérlés megáll a kijelölt pontnál, érdemes a vizsgálni kívánt függvény első kódsorában elhelyezni ezt a pontot. Ha megállítjuk a kódunk futását akkor a kódban a változók nevére húzva az egeret megjelenik annak értéke, ez a függvényekre is igaz.

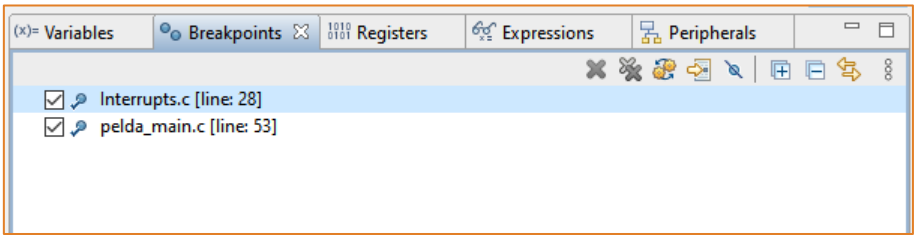
A debugolásnál a jobb oldalon elhelyezkedő nézet nagyon hasznos információkat tartalmaz. Segítségükkel jobban megérthetjük a kódunk működését, illetve a hibákat is jobban felderíthetjük. (Az ablak elhelyezkedése beállítás függő lehet.)

Első menüpont a *Variables*, itt a kódban lévő változók értékét figyelhetjük meg. Ha fut a kód nem fog megjelenni adat [nem frissül]. A változásokat az IDE sárga színnel emeli ki.



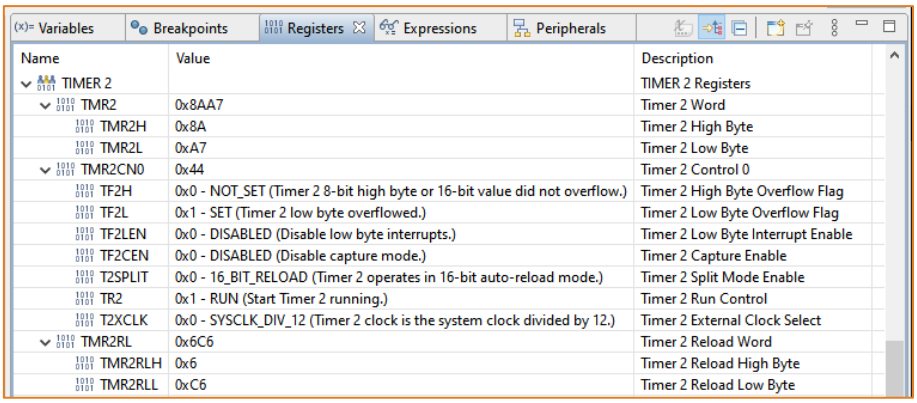
14. ábra: Variables menüpont

A *Breakpoints*-ban követhetjük, hova helyeztünk el töréspontokat a kódunkon belül. Kikapcsolhatjuk és kitörölhetjük innen őket, illetve gyorshivatkozásként az adott sorra ugorhatunk.



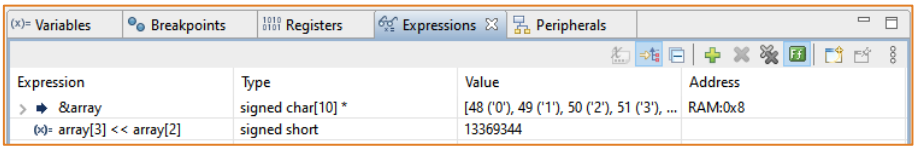
15. ábra: Breakpoints menüpont

A *Registers*-ben a regisztereink értékét figyelhetjük meg, a Value mezőbe kattintva átírhatjuk azokat.



16. ábra: Registers menüpont

Expressions-ban figyelőket írhatunk, kisebb függvényeket számolásokat.



17. ábra: Expressions menüpont

Változók és kódolási követelmények

Mivel korlátozott hely áll rendelkezésünkre, ezért a változónk méreteit is át kell gondolnunk. Mindig csak akkora helyet foglaljunk le a változóink számára, amennyi minimálisan indokolt, de ügyeljünk rá, hogy ne csordulhasson túl.

Változók:

C kódban használt változat	Mikrovezérlőn használt megfelelő	Méret
unsigned char	uint8_t	1 byte
unsigned short	uint16_t	2 byte
unsigned long	uint32_t	4 byte
signed char	int8_t	1 byte (signed)
signed short	int16_t	2 byte (signed)
signed long	int32_t	4 byte (signed)

Miért írjunk U karaktert egy pozitív szám után?

Mivel így jelezzük a fordítónak, hogy az adott változó előjel nélküli lesz, ha ezt nem tesszük meg nagy számoknál túlcsordulhat az integer értéke, és negatív értéket kaphatunk. (alapértelmezetten (signed) int-ként foglalja le a beírt számoknak a helyet), ami hibát okozhat.

Használati példa:

```
if(variable != 0U)
{
    //User code
}
```

Formai követelmények: CooSpace: MicLab-programozas.pptx

KEIL vs SDCC

A laboron a korábban használt F410 mikrovezérlőhöz SDCC compiler használtunk, aminek sok előnye van, többek között az, hogy ingyenes. Az új EFM8BB1 MCU-t csak az új Simplicity Studióval és KEIL compilerrel tudjuk használni. (A KEIL az ipar által használt egyik legnépszerűbb compiler, segít a memória bankok kezelésében és az optimalizálásban is fejlettebb, mint az SDCC, ezek használatával gyorsabbá tehetjük a kódunkat).

KEIL Cx51 dokumentáció: https://www.keil.com/support/man/docs/c51

A Laboratory Practicals-ban az SDCC szintaxisa van használva, viszont az új IDE és compiler új szintaxist ad a mikrovezérlőnk programozásához. Néhány fontosabb függvényre nézzünk meg néhány példát:

Interrupt:

```
SI_INTERRUPT(name, vector)
{
	// User code
}
```

A makrót az si_toolchain biztosítja számunkra, két paramétere van a függvénynek, az első egy name, ami a megszakítás kezelő függvényünk neve lesz, a második pedig egy vector, ami az Interrupt definitons-ban található meg.

Port-bit (SBIT):

```
SI_SBIT(name, address, bitnum);
```

Korábban már említésre került a dokumentumban a működése. Mivel itt gyakoribb egy-egy egyedi elnevezés (például egy gomb/LED neve), ezért ezt a függvényt gyakrabban hozzuk létre mi, mint az előre Includeolt SI_SFR társát.

Bit típusú változó:

```
bit activeState = 0;
```

Teljes port (SFR):

```
SI_SFR(name, address)
{
	sfr name = address;
}
```

Egy 8 bites SFR változót tudunk így deklarálni. De erre nincs szükség mivel az Includes tartalmazza az összes ilyen regiszter deklarációt. Például a P1 port: `SI_SFR (P1, 0x90);` természetesen, ha egyéni nevet szeretnénk adni egy ilyen speciális funkció regiszternek, akkor használjuk a függvényt. A deklarálás után szabadon adhatunk értéket a regiszterünknek (name paraméter alapján tudunk rá hivatkozni), írhatjuk, törölhetjük azt (erre szintén találhatunk példát a dokumentumban).

Példa feladat

LED vezérlés megvalósítása interrupt segítségével

Egy projekthez kell egy állapot jelző LED-et készítenünk. A megrendelő azokat a feltételeket szabta meg nekünk, hogy a LED-nek 250 ms-ként kell állapotot váltania, és az időzítéshez Timer-2-őt kell használnunk. Az órajel maradjon alapértelmezett értéken (SYSCLK/8). Fontos feltétel még, hogy hibakezelést kell alkalmaznunk. A hibás működést most a board-on lévő gomb szimulálja, ha annak állapota lenyomott, akkor állítsuk le a Timert és a LED ne világítson. (Ha a hiba elmúlik folytatódjon az állapotról való visszajelzés).

A megrendelő azt is szeretné, hogy *Debug* módban nézzük meg milyen lenne, ha 25%-kal gyorsabban villogna a LED. Vezessük le a számítás menetét!

Config (a konfigurációs .hwconf fájl egy xml, amit jegyzettömbbel is megnyithatunk):

```
<?xml version="1.0" encoding="ASCII"?>
<device:XMLDevice xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:device="http://www.silabs.com/ss/hwconfig/document/device.ecore" name="EFM8BB10F8G-A-QSOP24"
partId="mcu.8051.efm8.bb1.efm8bb10f8g-a-qsop24" version="4.0.0" contextId="%DEFAULT%">
  <mode name="DefaultMode">
    <property object="DefaultMode" propertyId="mode.diagramLocation" value="100, 100"/>
    <property object="INTERRUPT_0" propertyId="ABPeripheral.included" value="true"/>
    <property object="INTERRUPT_0" propertyId="interrupt.interruptenable.enableallinterrupts"
value="Enabled"/>
    <property object="INTERRUPT_0" propertyId="interrupt.interruptenable.enabletimer2interrupt"
value="Enabled"/>
    <property object="P0.2" propertyId="ports.settings.label" value="BTN0"/>
    <property object="P1.4" propertyId="ports.settings.iomode" value="Digital Push-Pull Output"/>
    <property object="P1.4" propertyId="ports.settings.label" value="LED0"/>
    <property object="P1.4" propertyId="ports.settings.outputmode" value="Push-pull"/>
    <property object="PBCFG_0" propertyId="pbcfg.settings.enablecrossbar" value="Enabled"/>
    <property object="TIMER01_0" propertyId="ABPeripheral.included" value="true"/>
    <property object="TIMER16_2" propertyId="ABPeripheral.included" value="true"/>
    <property object="TIMER16_2" propertyId="timer16.control.runcontrol" value="Start"/>
    <property object="TIMER16_2" propertyId="timer16.control.timerrunningstate" value="Timer is
Running"/>
    <property object="TIMER16_2" propertyId="timer16.initandreloadvalue.targetoverflowfrequency"
value="4"/>
    <property object="TIMER16_2" propertyId="timer16.initandreloadvalue.timerreloadvalue"
value="1734"/>
    <property object="TIMER16_2" propertyId="timer16.reloadhighbyte.reloadhighbyte" value="6"/>
    <property object="TIMER16_2" propertyId="timer16.reloadlowbyte.reloadlowbyte" value="198"/>
    <property object="TIMER16_3" propertyId="ABPeripheral.included" value="true"/>
    <property object="TIMER_SETUP_0" propertyId="ABPeripheral.included" value="true"/>
    <property object="WDT_0" propertyId="ABPeripheral.included" value="true"/>
    <property object="WDT_0" propertyId="wdt.watchdogcontrol.wdtenable" value="Disable"/>
    <property object="WDT_0" propertyId="wdt.watchdogcontrol.wdtinitialvalue" value="5"/>
    <property object="WDT_0" propertyId="wdt.watchdogcontrol.wdtperiodactual" value="6.554 s"/>
  </mode>
  <modeTransition>
    <property object="RESET &#x2192; DefaultMode" propertyId="modeTransition.source" value="RESET"/>
    <property object="RESET &#x2192; DefaultMode" propertyId="modeTransition.target"
value="DefaultMode"/>
  </modeTransition>
</device:XMLDevice>
```

Kódrész:

main.c

```
//-----------------------------------------------------------------------------
// Includes
//-----------------------------------------------------------------------------
#include <SI_EFM8BB1_Register_Enums.h>                  // SFR declarations
#include "InitDevice.h"

//-----------------------------------------------------------------------------
// Global constants
//-----------------------------------------------------------------------------

#define BTN_PRESSED 0U

#define LED_OFF 1U

#define ON  1U
#define OFF 0U

//-----------------------------------------------------------------------------
// Pin declaration
//-----------------------------------------------------------------------------

#define BTN0 P0_B2   //P0.2 Button definition
#define LED0 P1_B4   //P1.4 LED definition

/*
 * An other method to define a constant for pins
 *
    SI_SBIT(SW, SFR_P0, 2);      //P0.2-es porton talalhato gomb
    SI_SBIT(LED0, SFR_P1, 4);    //P1.4-es porton talalhato LED
 *
 */

//-----------------------------------------------------------------------------
// SiLabs_Startup() Routine
// ----------------------------------------------------------------------------

void
SiLabs_Startup (void)
{
}

//-----------------------------------------------------------------------------
// main() Routine
// ----------------------------------------------------------------------------
int main (void)
{
  // Call hardware initialization routine
  enter_DefaultMode_from_RESET ();

  while (1)
  {
      if (BTN0 == BTN_PRESSED)
      {
          TMR2CN0_TR2 = OFF; //Turn off Timer 2 run-control
          LED0 = LED_OFF;    //Turn off LED0
      }
```

```
      else
      {
          TMR2CN0_TR2 = ON; //Turn on Timer 2 run-control
      }
  }
}
```

Interrupts.c

```
//-----------------------------------------------------------
// src/Interrupts.c: generated by Hardware Configurator
//
// This file will be regenerated when saving a document.
// leave the sections inside the "$[...]" comment tags alone
// or they will be overwritten!
//-----------------------------------------------------------

// USER INCLUDES
#include <SI_EFM8BB1_Register_Enums.h>
//-------------------------------------------------------------------------------
// Pin declaration
//-------------------------------------------------------------------------------

#define LED0 P1_B4 //P1.4 LED definition

/*
 * An other method to define a constant for pins
 *
 *  SI_SBIT (LED0, SFR_P1, 4);
 *
 */

//-------------------------------------------------------------------------------
// TIMER2_ISR
//-------------------------------------------------------------------------------
//
// TIMER2 ISR Content goes here. Remember to clear flag bits:
// TMR2CN0::TF2H (Timer # High Byte Overflow Flag)
// TMR2CN0::TF2L (Timer # Low Byte Overflow Flag)
//
//-------------------------------------------------------------------------------

SI_INTERRUPT (TIMER2_ISR, TIMER2_IRQn)
{
  TMR2CN0_TF2H = 0; //Timer 2 High Byte Overflow Flag clear

  LED0 = !LED0;    //Switch LED0 state [ON/OFF]
}
```

Debug-mód, timer-reload-value átállítása:

Az útmutatásnak megfelelően 5 Hz-es frekvenciát kell elérnünk (ez 25%-os emelkedés az előző frekvenciához képest), az alábbiak alapján határozzuk meg az értéket:

SYSCLK: 24.5 MHz = 24 500 000 Hz
SYSCLK_PRE_SCALER: 8
TIMER2_PRE_SCALER: 12
F_TIMER2: 5 Hz

Elsőnek számoljuk ki a bemenő órajelet:

$$F_{INPUT_{CLK}} = \frac{SYSCLK}{SYSCLK_{PRE_{SCALER}} * TIMER2_{PRE_{SCALER}}} =$$

$$= \frac{24\ 500\ 000\ \text{Hz}}{8 * 12} \approx 255\ 208\ \text{Hz}$$

Ez után számoljuk ki a Timer-reload értékét:

$$TIMER_{RELOAD_{VALUE}} = 2^{16} - \left(\frac{F_{INPUT_{CLK}}}{F_{TIMER2}}\right) =$$

$$= 65536 - \left(\frac{255\ 208\ \text{Hz}}{5\ \text{Hz}}\right) \approx 14494$$

Az így kapott decimális értéket átváltjuk hexadecimálisba, ami 0x389E. Ezt az értéket ezután beillesztjük debug módban a TMR2RL regiszterbe:

TMR2RL	0x389E	Timer 2 Reload Word
TMR2RLH	0x38	Timer 2 Reload High Byte
TMR2RLL	0x9E	Timer 2 Reload Low Byte

1. *mintafeladat ábra: Debug mód | TIMER2 reload regisztere*

Az elkészült programot be kell mutatni!

A gyakorlatvezető ellenőrizte:

- *igen*
- *nem*

A program működött:

- *igen*
- *nem*

USB - UART soros kommunikáció

Az UART és a Debugger chip segítségével képesek vagyunk USB-n keresztül adatot fogadni és küldeni. Hogy működőképes legyen ez a mód pár módosítást végre kell hajtanunk.

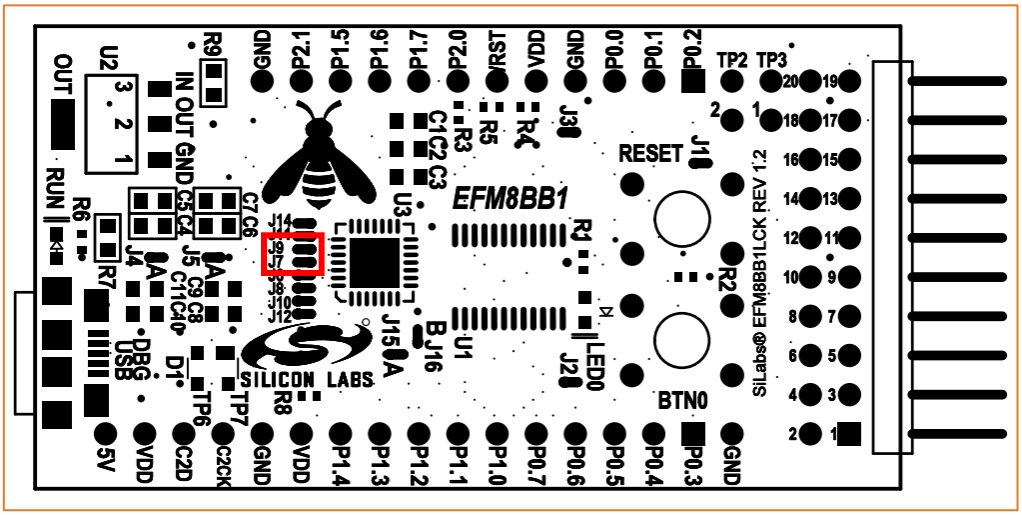
Mindkét módszer előfeltétele, hogy az UART-ot megfelelően felkonfiguráljuk, a flageket helyesen kezeljük a kódunkban.

1. módszer:

   Itt nem kell forrasztáshoz folyamodnunk, viszont modult kell beszereznünk. *USB to UART bridge* a kiegészítő neve, ezzel kell összekötnünk a P0.4 és P0.5 pineket. Ha ezt megtettük akkor egy virtuális COM port fog megjelenni, és egy terminállal tudunk majd küldeni és fogadni adatokat (BAUD rate, paritás, bitek ismeretében). Előnye, hogy Linuxon is tökéletesen működik, viszont drágább, mint a másik megoldás.

2. módszer:

   Forrasztást igényel ez a módszer, a J7 és J9 jelzésű padokat kell összeforrasztani.



18. ábra: Összeforrasztandó pad-ok [2]

Ha ezt megtettük töltsük le a Toolstick Terminal-t, amin csatlakozzunk a mikrovezérlőhöz, ha mindent helyesen beállítottunk tudunk a terminálon keresztül adatokat küldeni és fogadni. Fontos, hogy a terminál csak 1 eszközhöz tud csatlakozni, illetve, ha aktív a kapcsolat a terminál és az MCU között nem tudunk kódot flashelni rá.

ToolStick-Terminal:
https://www.silabs.com/documents/login/software/ToolStick_Setup.exe

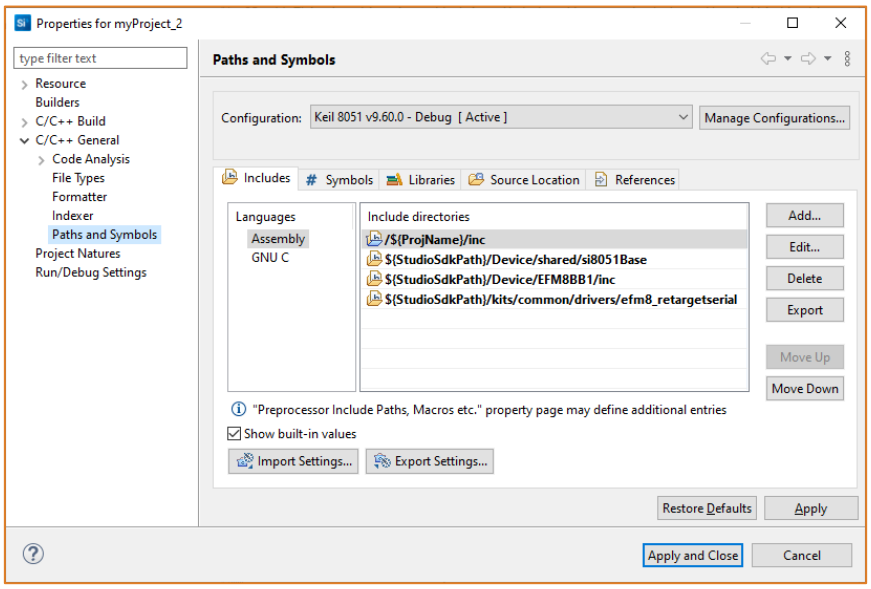
Adatkiírás a terminálra printf segítségével

Hogy printf segítségével adatokat tudjunk kiírni, a projekthez kell adni a retargetserial-t. Ezt úgy tudjuk megtenni, hogy a projektnek a properties-ébe navigálunk, és a *Paths and Symbols* menüpontban az *Assembly* és *GNU C* nyelvekhez hozzáadjuk a következő include directory-t: *${StudioSdkPath}/kits/common/drivers/efm8_retargetserial* .

Ha ezt megtettük a headert include-olnunk kell, a `RETARGET_PRINTF()` függvényt akkor tudjuk használni, ha a Transmit Interrupt flagben jelezzük, hogy az UART transmit megtörtént:

```
SCON0_TI = SCON0_TI__SET;
```

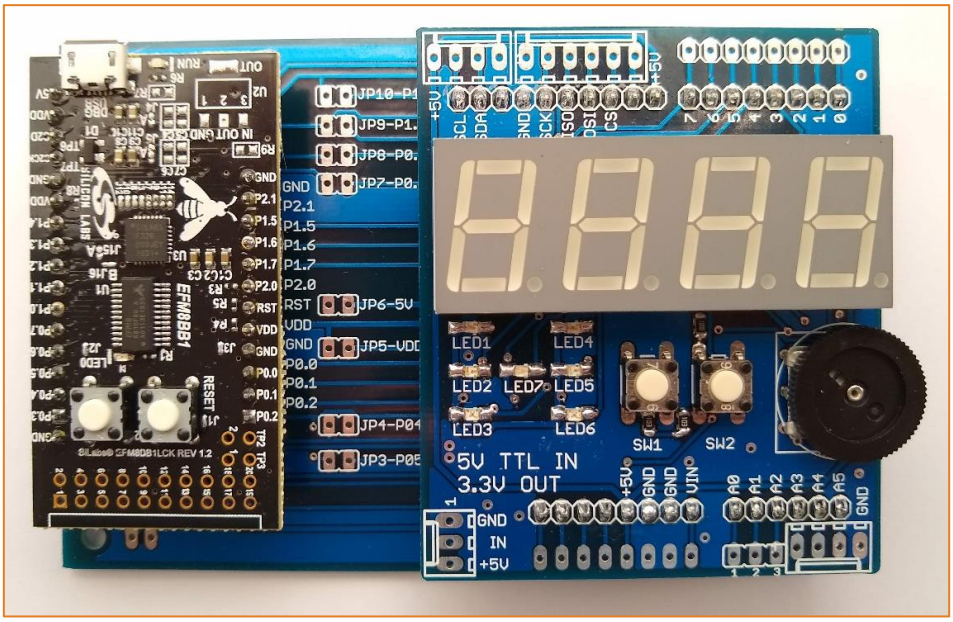
Ez után, mint egy sima printf-et használva tudunk írni a terminálra.



19. ábra: Projekt properties

Extension board

A laborhoz egy kiegészítő áramkört is készítettünk. Ezen a boardon többek között 4 darab hétszegmenses kijelző, 7 darab LED, 2 nyomógomb és egy potenciométer található. A mikrovezérlőt az adapter panel segítségével tudjuk rákötni erre az áramkörre. Az alul lévő adapter panel a EFM8-as kit lábkiosztását Arduino UNO lábkiosztásúvá alakítja, maga kiegészítő áramkör is egy Arduino UNO kompatibilis shield.

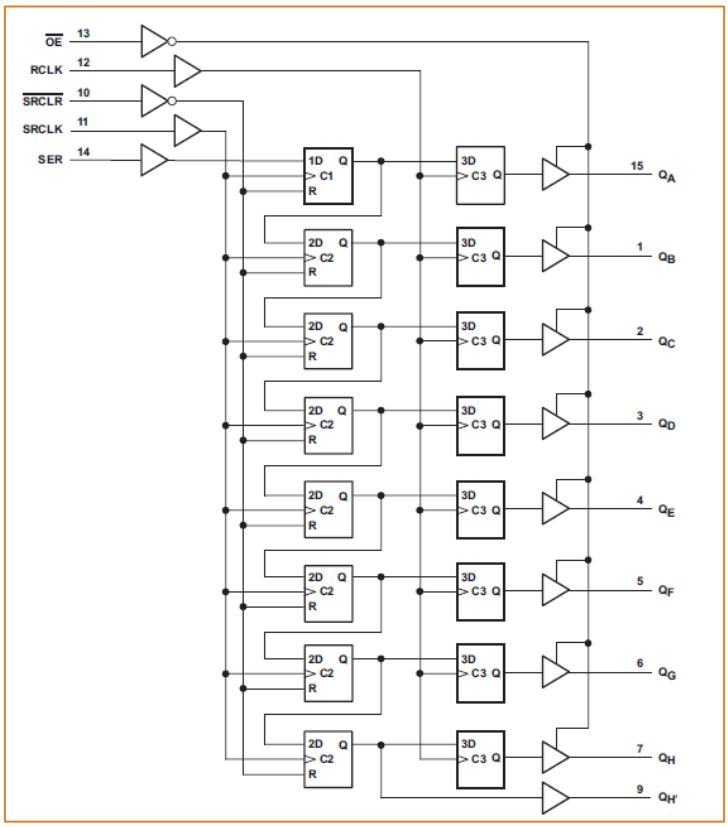


20. ábra: Kiegészítő áramkör

Kivezetésekhez tartozó külső perifériák és azok funkciói:

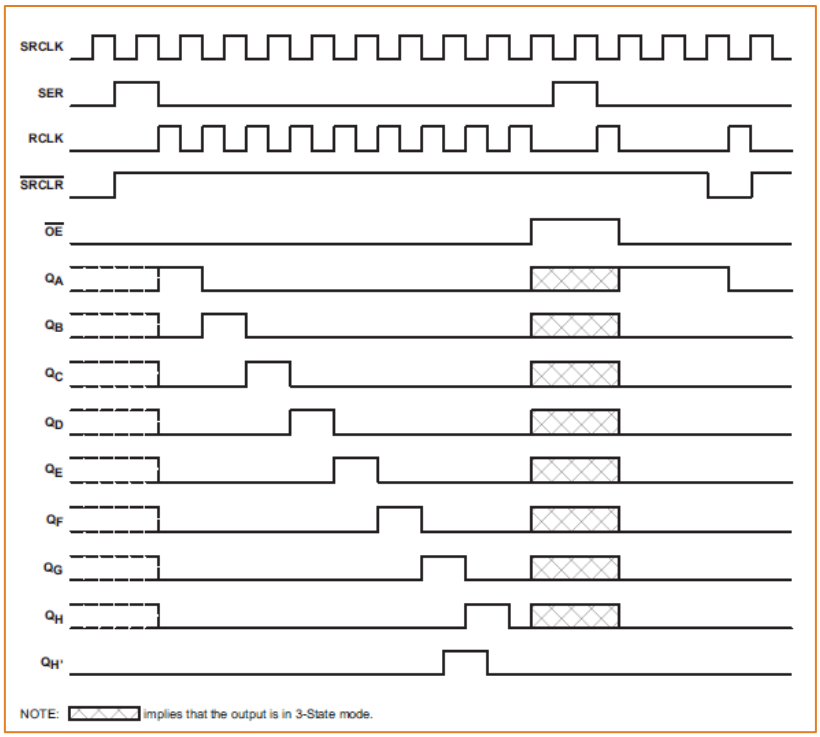
MCU Port Pin	EFM8BB1-LCK	4x7 segment shield pin function
P0.0		DispClock
P0.1		SW1
P0.2	Push Button 0 (BTN0)	DispData
P0.3		DispOutEnable
P0.4		Decoder-C, TX
P0.5		RX
P0.6		SPI-SCK
P0.7		SW2, SPI-MISO, A5
P1.0		SPI-MOSI
P1.1		Decoder-A, SPI-CS
P1.2		Decoder-B
P1.3		A0
P1.4	LED0, PCA CEX0	A1
P1.5		A2, IN
P1.6		A3
P1.7		A4, POTENTIOMETER

A kijelzőt és a különálló LED-eket meghajtó shift regiszter két legfontosabb diagrammja:



21.ábra: A shift regiszter funkcionális diagramja [3]

Bal oldalon az adat az órajel és az engedélyező bemenetek találhatók, jobb oldalon pedig a kimenetek, amikre a különálló LED-ek és a 7-szegmenses kijelzők vannak kötve egy digitális kapcsoló (decoder) IC-n keresztül. Az első fokozat a shift regiszter (fentről lefelé haladva, minden egyes órajelre a sorban a következő tárolóba kerül az adat). A második fokozatban 8 db független D-tároló található. A kimeneten még egy 3 állapotú puffer erősítő is található a D-tárolók után a nagyobb kivehető áramerősség érdekében.



22.ábra: Shift regiszter – Idődiagramm. Az SRCLK a shift regiszter fokozat órajele [3]

A 22-es ábrán a shift regiszter idődiagrammjára látható egy példa, amint a 0x01 értékű 8 bites számot megkapja és a kimeneteire (Q_A - Q_H) teszi. Az órajel (SRCLK) minden egyes felfutó élre beolvassa az adatbemeneten (SER) lévő logikai értéket a shift regiszterbe és továbbadja a fentről lefelé sorban következő tárolónak. A kimeneteken az RCLK felfutó élére jelenik meg az adat, feltéve, ha a kimenet engedélyezve van az OE bemenettel (logikai alacsony állapot).

Decoder adatlapja: pdf
7 szegmenses kijelző adatlapja: pdf
Shift-regiszter adatlapja: pdf

Segítség a kijelzővezérléshez:

- A különálló LED-eket és a hétszegmenses kijelző blokkjait egy decoder IC segítségével választhatjuk ki. Ennek a logikai táblázata a következő:

Decoder C (P0.4)	Decoder B (P1.2)	Decoder A (P1.1)	Selected
0	0	0	T1
0	0	1	T2
0	1	0	T3
0	1	1	T4
1	0	0	T5 (LEDs)

(T1…T5 megtalálható a mellékletekben a kapcsolási rajzon.)

- Figyelni kell arra, hogy a digitek számozása balról jobbra növekszik 1-től 4-ig.

- A zavaró vibrálás elkerülése érdekében a kijelzésnél gyorsan kell léptetnünk az aktív digitet (> 50 Hz).
- Shift regiszter használata bit-bang módszerrel (szoftveres protokoll):
    - Az egyes digiteken belül a szegmens LED-eket és a különálló LED-eket negatív logikával kell meghajtani (akkor világít egy LED, ha logikai alacsony van a shift regiszter adott kimenetén).
    - A kontraszt növeléséhez a kimeneteket kapcsoljuk le amíg feltöltjük a shift regisztert a DispOutEnable portbittel (negatív logikával üzemel).
    - Töltsük fel a shift regisztert a DispData nevű láb segítségével. Ehhez a 8 bites változónknak balról jobbra haladva mindig ki kell venni a következő bitjét és ezt kell kiírnunk a DispData portbitre. Az órajelet is létre kell hoznunk, amit a DispClock portbit felbillentésével tehetünk meg a DispData portbit írása után a kódban (a shift regiszter az aktuális adatbitet az SRCLK órajel felfutó élére olvassa be: lásd 22. ábra).
    - A shift regiszter feltöltése után szükség van egy órajelre az RCLK bemeneten, amit szintén a DispOutEnable portbit (P0.3) fel és le billentésével állíthatunk elő (össze vannak kötve az áramköri lapon). (Erre csak a 8 bites szám kiadása után van szükség (ellentétben az idődiagrammal), az egyes bitek kiadása között nem kell változtatni az RCLK vonalat.)
    - Végül engedélyezzük a kimenetet (DispOutEnable, negatív logikával üzemel).

Mikrovezérlő megrendelése

A kit beszerezhető (ajánlás):

- Farnell
- FDH (magyar)

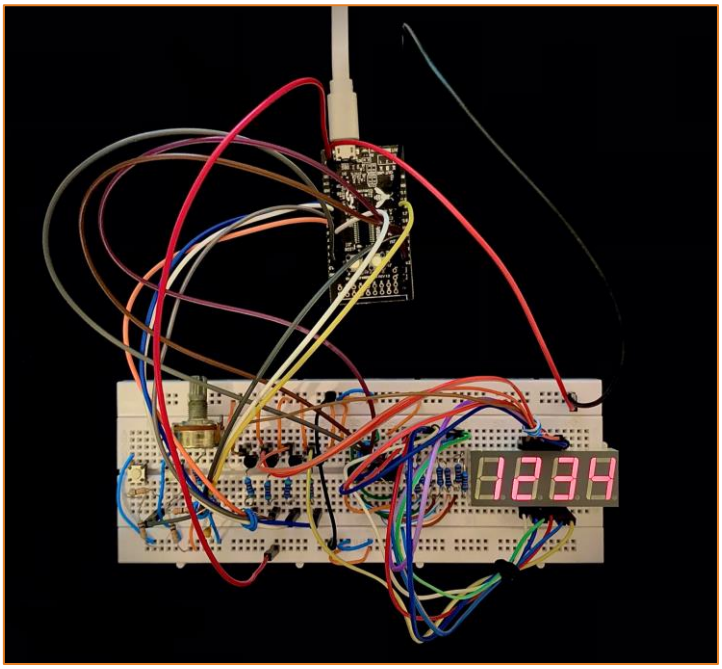
Kit tartalma:

- EFM8BB1LCK 8-bit MCU

Külön megvásárlandó:

- Micro USB kábel (adatátvitel)
- tüskesor (lábtávolság: 2,54mm)

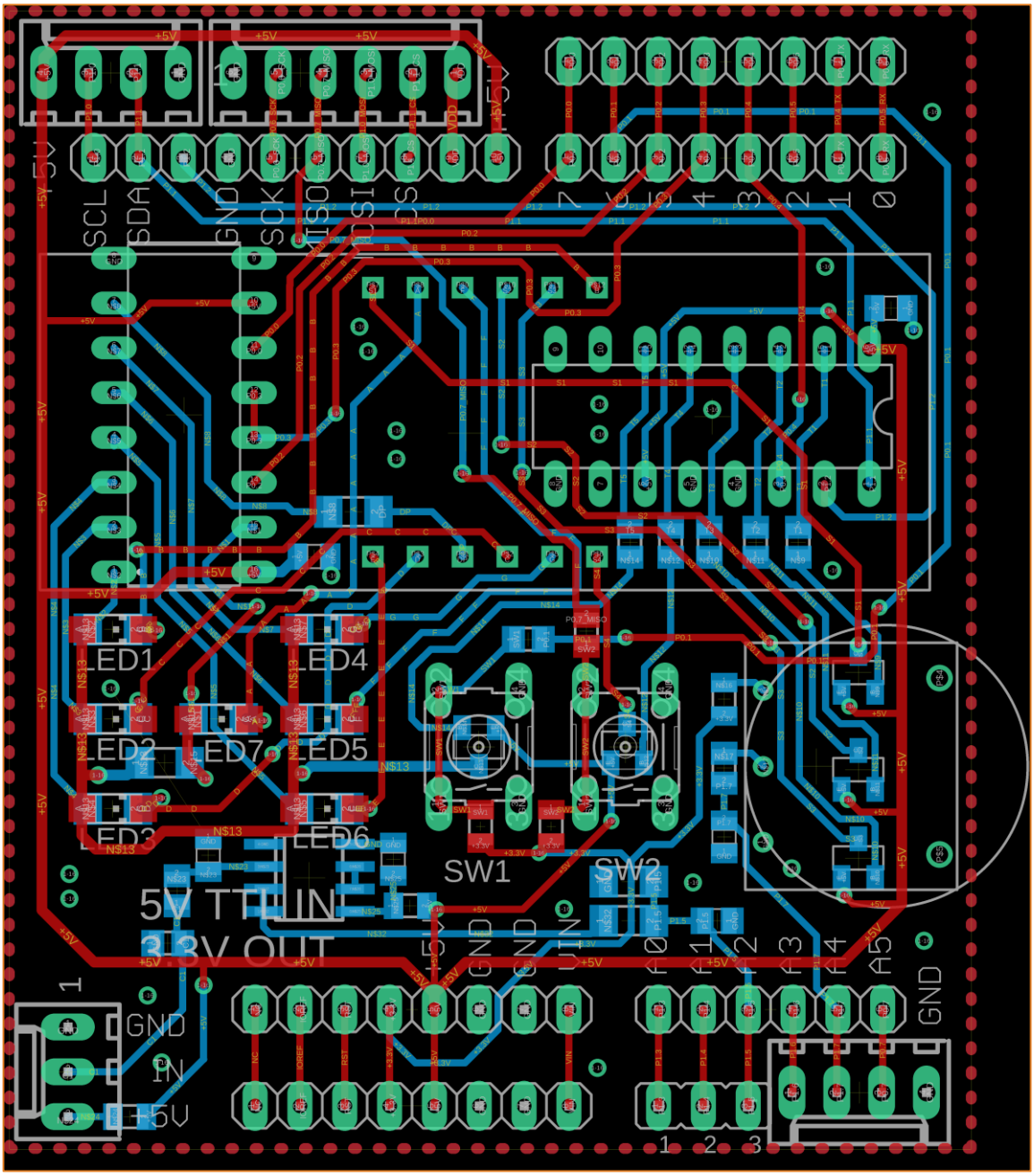
Extension-board otthoni összeállítása



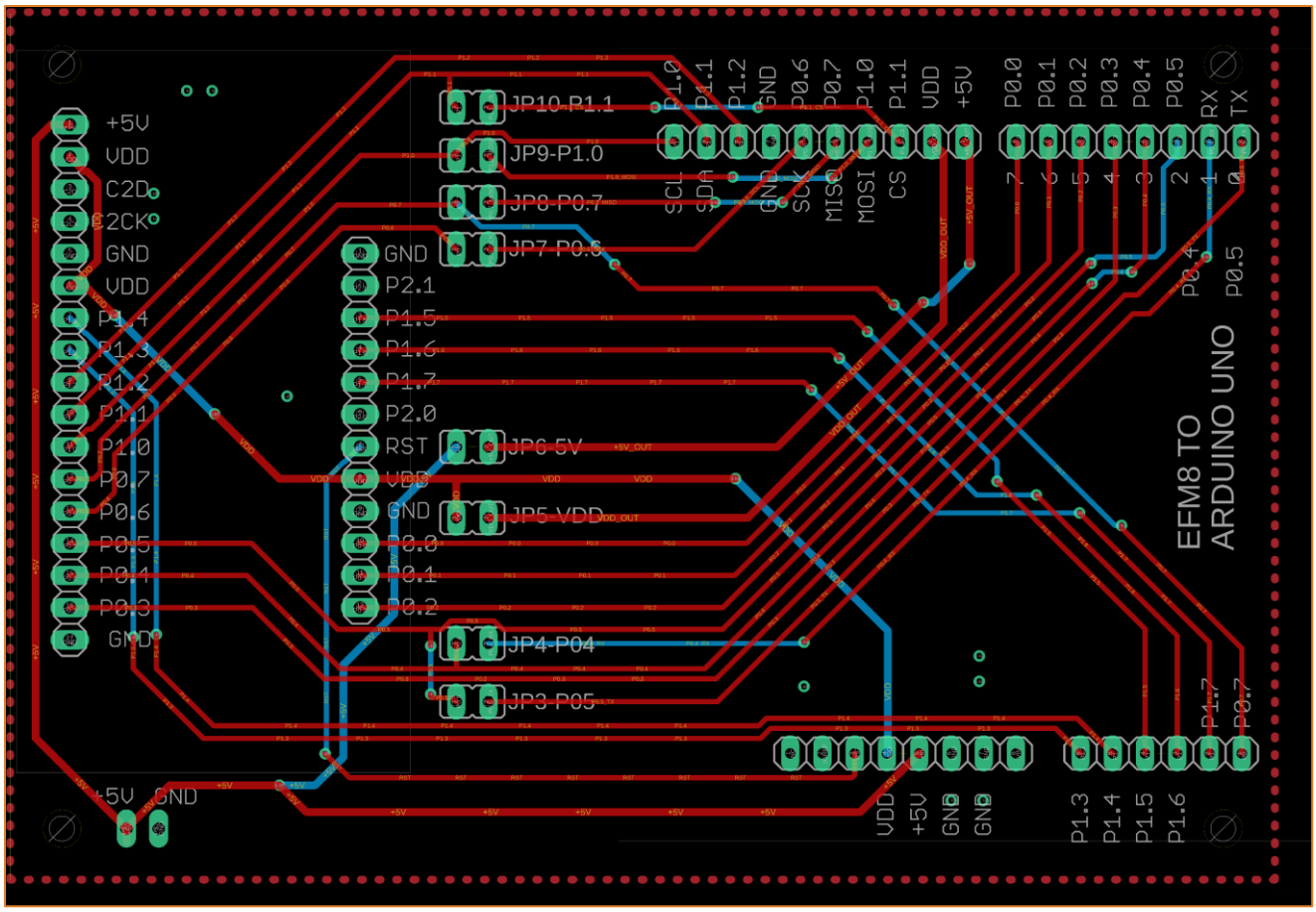
23. ábra: Breadboardon összeállított kapcsolás

Tanácsok a kapcsolás összeállításhoz:

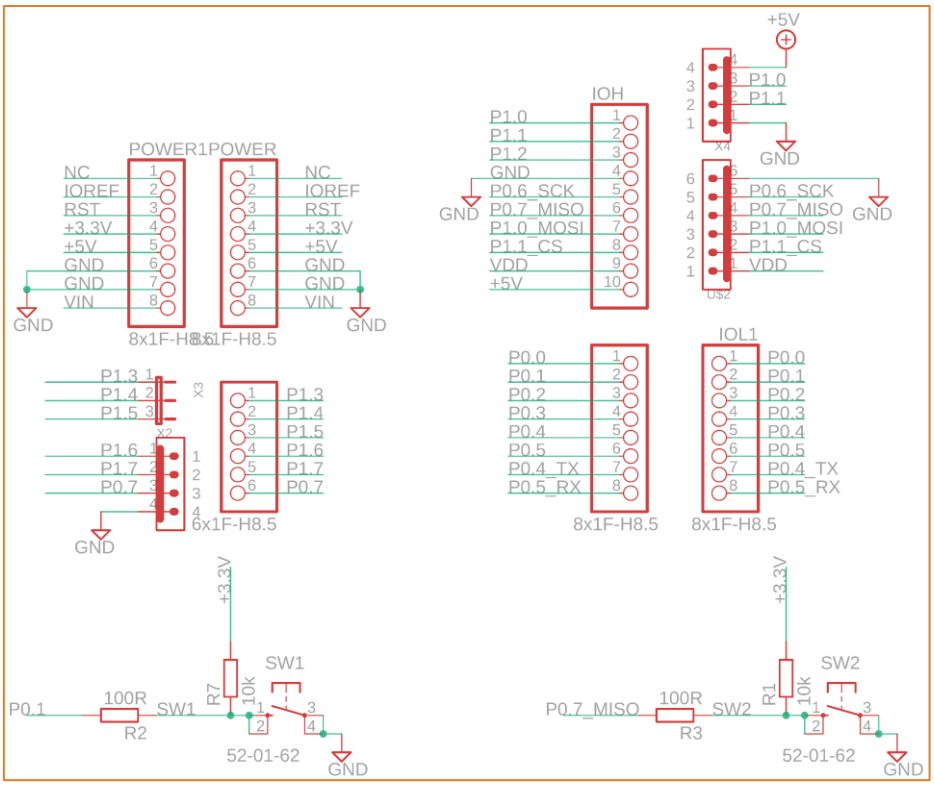
- Érdemes blokkokban elrendezni az elemeket a boardon, a fenti ábrán láthatjuk, hogy 3 egységre lett osztva a felület:
    - 1. Beviteli user interfész – itt található a potenciométer és a két gomb, fontos, hogy minél kevesebb vezeték legyen ezen a részen, mivel a perifériák használata közben elmozdíthatjuk kihúzhatjuk azokat.
    - 2. Logikai vezérlés – decoder IC helyett 4 darab NPN típusú tranzisztort ajánlunk, ezek segítségével egyszerűen kiválaszthatjuk a kijelzőket, a kijelző vezérlését egy 8 bites shift regiszter biztosítja.
    - 3. Kimeneti user interfész – szintén fontos, hogy minél kevesebb vezeték akadályozza az itt található 4 digites 7 szegmenses kijelző leolvasását, ezért a kábeleket célszerű rendezni.
- A rendezettebb megjelenés érdekében kötegeljük össze az azonos csoportba tartozó vezetékeket, használjunk egymás mellett eltérő színű kábeleket, rövidebb kábelek használata ajánlott. A +3.3/+5V piros vagy meleg színű vezetékkel legyen bekötve, a GND pedig feketével. A configurátorban címkékkel (label) láthatjuk el a pineket, ezt érdemes használni. Pl.: DISP_0 (orange).
- Tápsíneket használjuk, a 3.3V és 5V vonalat érdemes kivinni külön-külön tápsínre, itt vegyük figyelembe a használt breadboard kapcsolási rajzát (középen a tápsín meglehet szakítva).



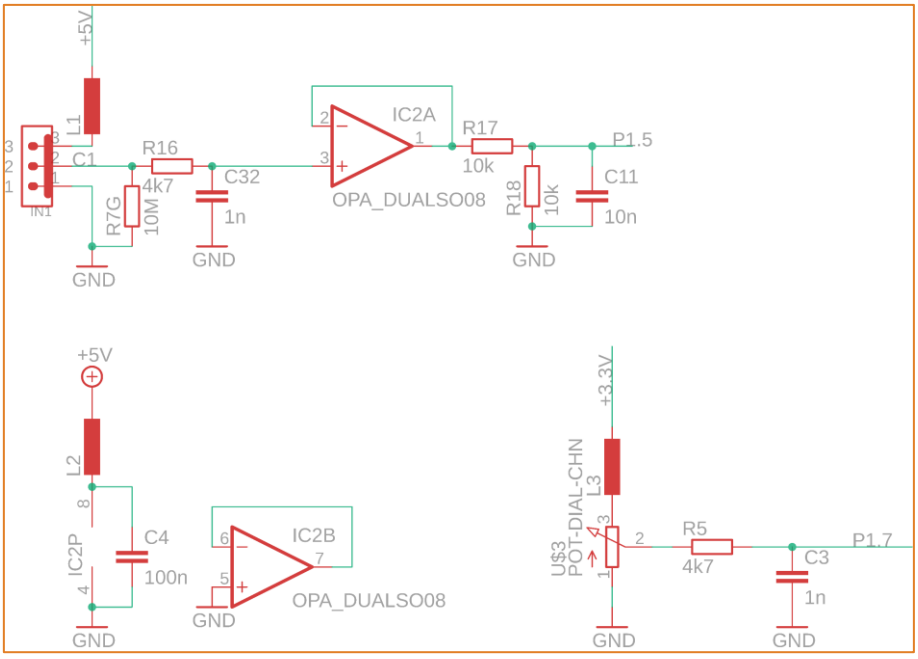
24.ábra: Kiegészítő board - nyákterv



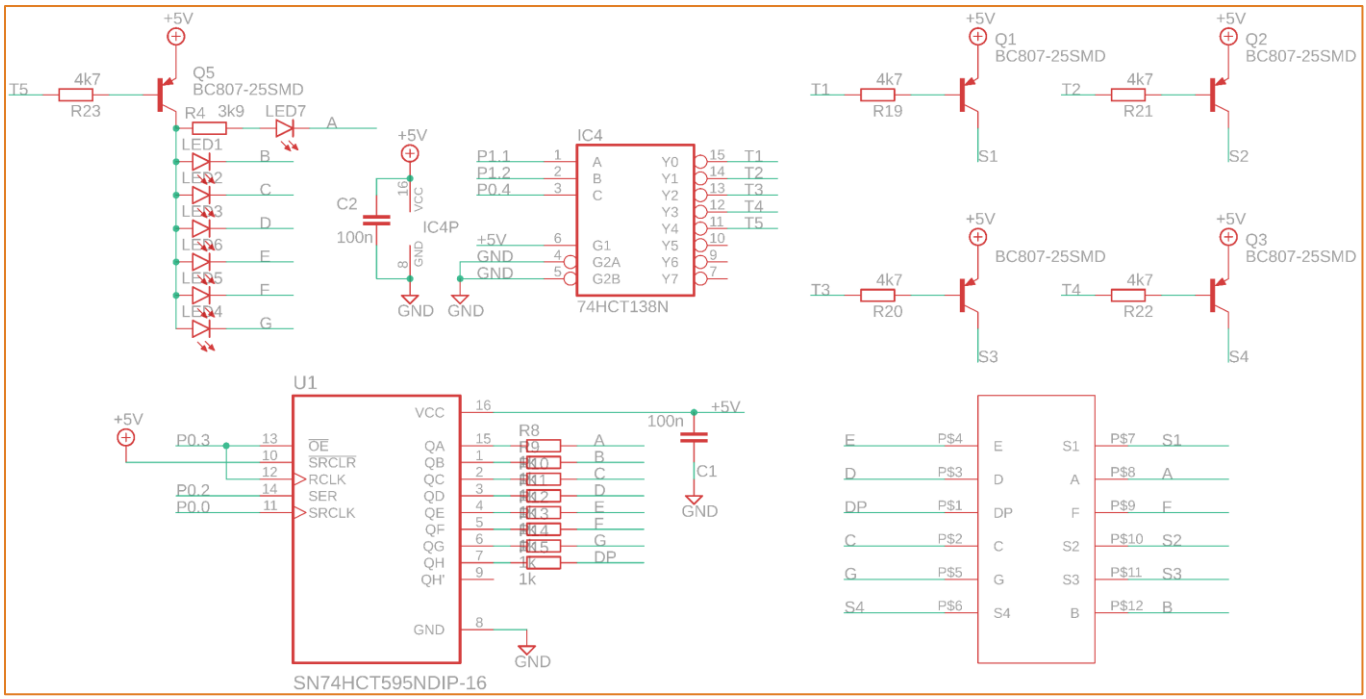
25.ábra: Adapter nyákterv



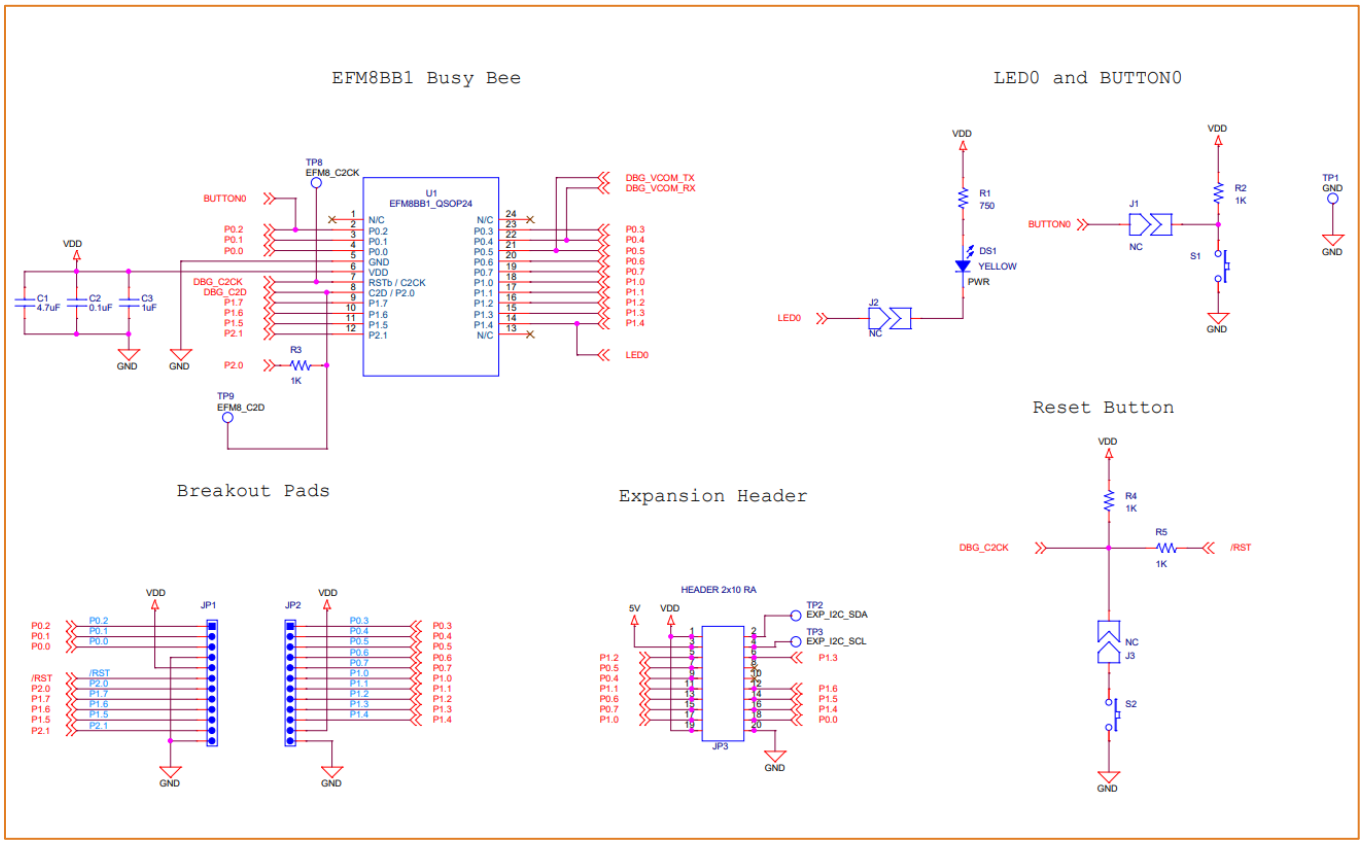
26.ábra: Kiegészítő áramkör kapcsolási rajz (1/3)



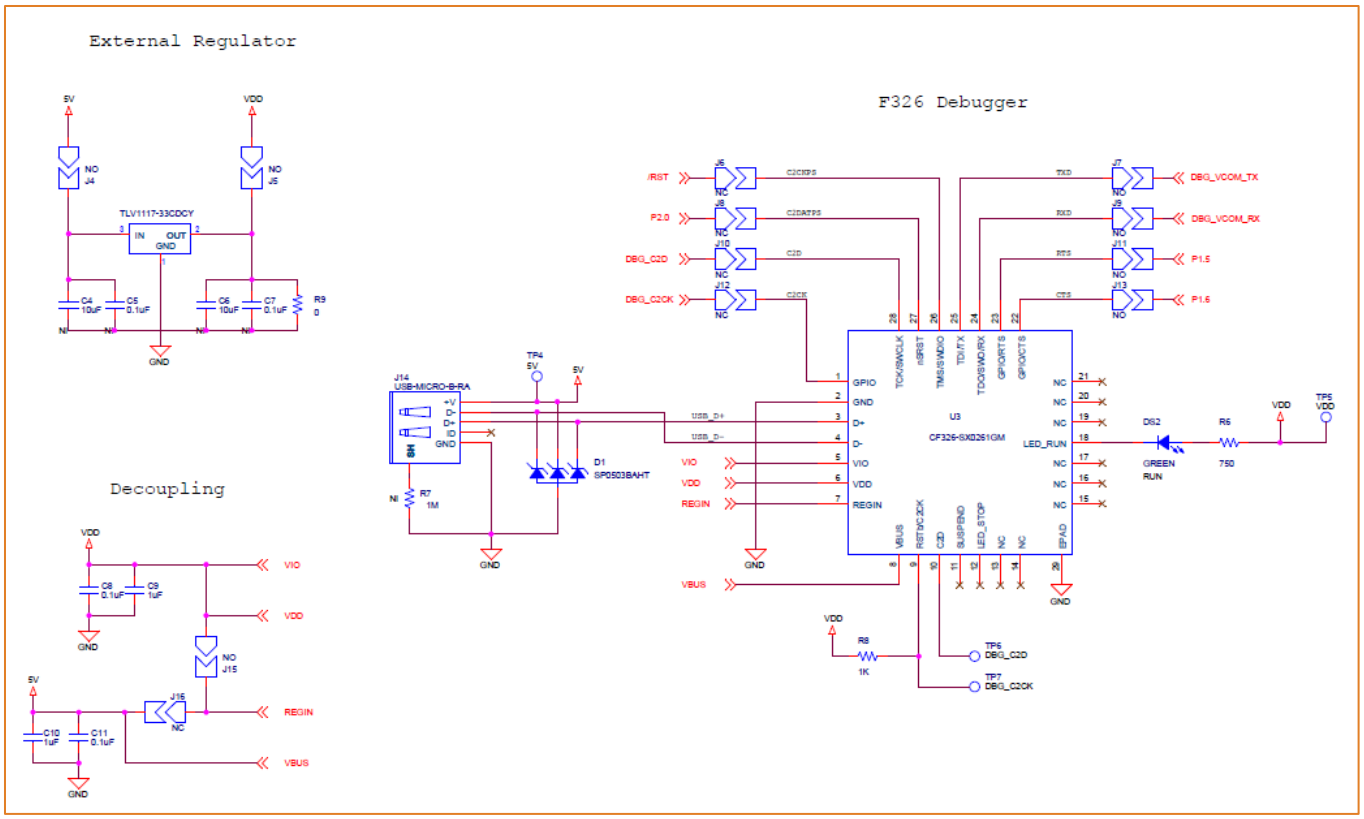
27.ábra: Kiegészítő áramkör kapcsolási rajz (3/3)



28.ábra: Kiegészítő áramkör kapcsolási rajz (2/3)



29.ábra: EFM8BB1LCK schematic (1/2) [2]



30.ábra: EFM8BB1LCK schematic (2/2) [2]

Irodalomjegyzék

[1] „EFM8BB1 - Reference Manual,” [Online]. Elérhető: https://www.silabs.com/documents/public/reference-manuals/efm8bb1-rm.pdf

[2] „EFM8BB1LCK-design-files,” [Online]. Elérhető: https://www.silabs.com/documents/public/schematic-files/efm8bb1lck-design-files.zip

[3] „SNx4AHCT595 8-Bit Shift Register,” [Online]. Elérhető: https://www.ti.com/lit/ds/symlink/sn74ahct595.pdf