Az MDK-ARM bemutatása

SZERZŐ: FILEP GÁBOR

COMPUTER CONTROLS

info@ccontrols.hu | www.ccontrols.hu

Az MDK-ARM egy teljes szoftver-fejlesztői környezetet biztosít Cortex™-M, Cortex-R4, ARM7™ és ARM9™ magos processzoron alapuló eszközökhöz.

Kifejezetten mikrokontrollerekhez tervezték, használatát könnyű elsajátítani, emellett a legtöbb beágyazott szoftver tervezéséhez használható.

Az alábbi útmutatóban lépésről lépésre bemutatjuk, hogyan hozzunk létre egy új projektet, illetve hogyan végezzük el a hibakeresést. Az alkalmazáson bemutatásra kerülnek egy debugger főbb funkciói is:

- Hozzáadjuk a szükséges komponenseket a projekthez

- Létrehozzuk a forráskód állományokat.

- Létrehozzuk és letöltjük az alkalmazást

- Használjuk a debuggert

TARTALOMJEGYZÉK

FEJLESZTŐESZKÖZ TÁMOGATÁS

Vannak olyan interfészek a fejlesztőeszközökön, amiket rendszeresen használunk, úgymint LED-ek, nyomógombok, joystick, A/D és D/A konverterek, LCD-k és érintőkijelzők, továbbá külső szenzorok: úgymint hőmérő, gyorsulás érzékelő, magnetométer és giroszkóp.

A **Board Support Interface API** egy szabványos hozzáférést biztosít ezekhez az elemekhez. Ez lehetővé teszi a szoftverfejlesztőknek, hogy az alkalmazás kódjukkal foglalkozzanak ahelyett, hogy az eszköz leírását forgatnák a szükséges regiszter beállításokért.

Sok Device Family Pack (DFP) tartalmaz fejlesztőeszköz támogatást (Board Support). A Manage Run-Time Environment ablakban kiválaszthatjuk a szükséges fejlesztőeszköz támogatásokat.

Ügyeljen arra, hogy a megfelelő **Variant** legyen kiválasztva, hogy a megfelelő láb kiosztással használja a fejlesztőeszközét.

ALKALMAZÁSOK LÉTREHOZÁSA

LED VILLOGÓ

Ez a rész elmagyarázza a projekt létrehozását a következő lépéseken keresztül:

- **A Projekt beállítása:** létrehozunk egy projekt fájlt és kiválasztjuk a mikrovezérlőt valamint a hozzátartozó CMSIS komponenseket.
- **Beállítjuk az eszköz rendszer órajelét.**
- **Létrehozzuk a forráskód állományokat:** létrehozzuk és hozzáadjuk az alkalmazás állományokat.
- **Beállítjuk a fordítási opciókat:** kiválasztjuk a fordítót, beállítjuk a szükséges memória térképet, a létrehozandó kimeneti fájlokat, a C/C++ fordító paramétereit
- **Létrehozzuk az alkalmazást:** lefordítjuk és linkeljük az alkalmazást a letöltéshez
- **Letöltjük az alkalmazást**

A Blinky alkalmazáshoz létre kell hozni a következő alkalmazás fájlt:

main.c Ez a fájl tartalmazza a *main()* függvényt, ami inicializálja a System Tick Timer-t és ennek a kezelő függvényét a *SysTick_Handler()*.

A PROJEKT BEÁLLÍTÁSA

A µVision menüsorból válassza ki a **Project – New µVision Project**.

- Hozzon létre egy üres könyvtárat és adja meg a projekt nevét, például *Blinky*.
  Kattintson a **Save**-re, ami létrehoz egy üres projekt állományt a megadott néven (*Blinky.uvprojx*).

Ezt követően megnyílik a **Select Device for Target** párbeszédablak.

- Válassza ki az EFM32GG990F1024 eszközt és kattintson az **OK**-ra.

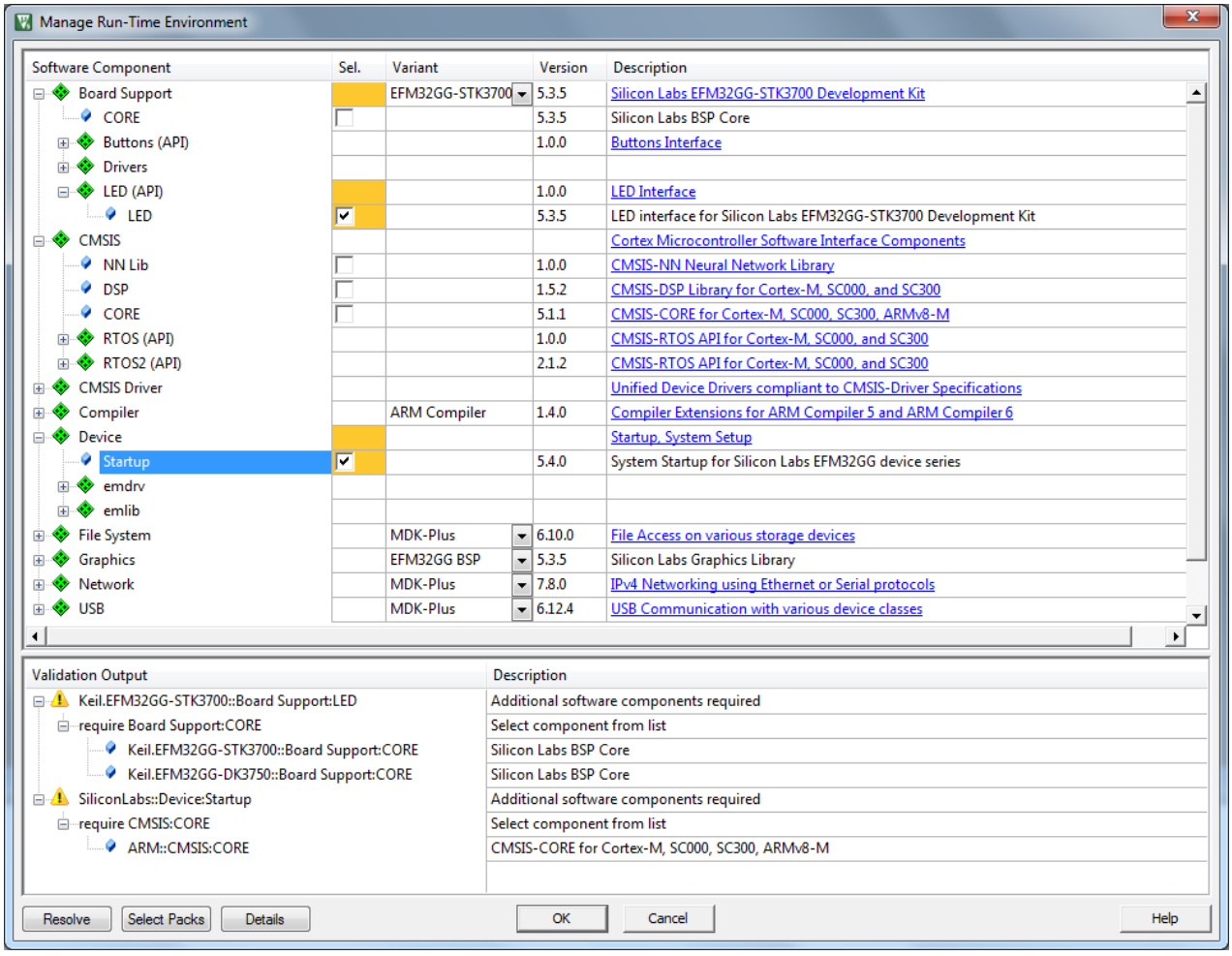
Az eszköz kiválasztása meghatározza az alapvető eszközbeállításokat, úgymint a fordítóvezérlőket, a memória térképet a linker számára és a Flash programozáshoz szükséges algoritmusokat.

Ezt követően megnyílik a **Manage Run-Time Environment** párbeszédablak és azokat a szoftver komponenseket láthatjuk, amik előzőleg már installálva lettek és elérhetőek a kiválasztott eszközhöz. A gyorsabb fejlesztés támogatva a gyártó **Middleware**-eket bocsát rendelkezésre az adott Starter kit-ekhez.

- Állítsa be a **::Board Support:EFM32GG-STK3700** opciót.
  Nyissa ki a **::Board Support:LED (API)** és engedélyezze a **LED** elemet.
  Nyissa ki a **::Device** és engedélyezze a **:Startup** elemet.

A **Validation Output** mezőben láthatóak a kiválasztott szoftver komponensek függőségei. Ebben az esetben az **ARM::CMSIS:CORE** és a **::Board Support:CORE** komponensek szükségesek.

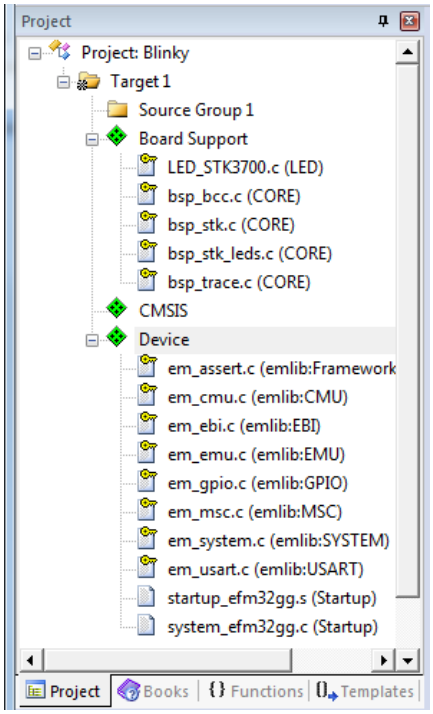
- Kattintson a **Resolve**-ra.



Ez feloldja az összes függőséget és engedélyezi a többi szükséges szoftver komponenst.

- Kattintson az **OK**-ra.

A projekt tartalmazni fogja a kiválasztott szoftver komponenseket a startup állományokkal, a CMSIS rendszer állományokkal és a fejlesztőeszközt támogató állományokkal együtt. A **Project** ablakban láthatók a kiválasztott szoftver komponensek a kapcsolódó fájlokkal együtt. Egy fájlon való dupla kattintás megnyitja azt a szerkesztőben.



BEÁLLÍTJUK AZ ESZKÖZ RENDSZER ÓRAJELÉT

A rendszer és a processzor órajelét a *system_<device>.c* fájl határozza meg.

Megjegyzés: Néhány eszköznél a rendszer beállítás a main függvény része és/vagy egy szoftver keretrendszert használ ennek a beállítására egy külső alkalmazással.

Az órajel konfiguráció egy alkalmazásnál számos dologtól függ, úgymint az órajel forrástól (XTAL vagy integrált oszcillátor) továbbá a memóriáknak és perifériáknak a szükségleteitől. A gyártók rendelkezésre bocsátanak egy eszköz-specifikus állományt *system_<device>.c*, ennek megfelelő használatához elengedhetetlen elolvasni a kapcsolódó dokumentumokat.

Tipp: Nyissa meg a referencia kézikönyvet a **Books** ablakból, hogy a mikrovezérlő órajel rendszeréről részletes információkat kapjon.

Az STK3700 fejlesztői eszköz a belső magas frekvenciás oszcillátorról (HFRCO) fut 14MHz-en a tápfeszültségre helyezést követően. Ez az alapértelmezett gyári beállítás, egyelőre módosításra nincs szükség. Azonban, ha változtatna a beállításokon az egyedi fejlesztéséhez a *system_efm32gg.c* fájlt kell módosítani. A rendszer induláskor a *SystemInit()* függvényt hívja meg a *main()* függvény hívása előtt. A rendszer órajel módosításához ebbe a függvénybe kell a megfelelő módosításokat elvégezni. Új projekt létrehozásakor ez a függvény üres.

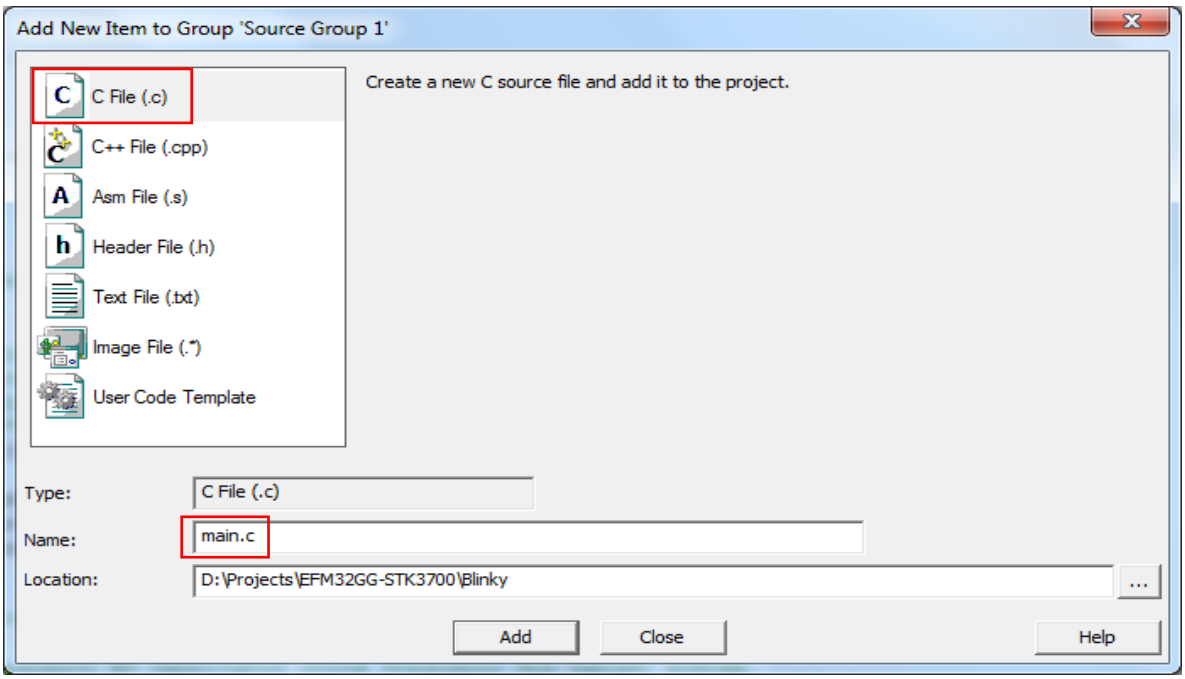
A *system_efm32gg.c* fájl módosításához nyissa ki a **Device** csoportot a **Project** ablakban és kattintson kettőt a fájl nevén, majd módosítsa a fájlt a lentebb látható példa alapján.

Órajel beállítás a *system_efm32gg.c* fájlban

```
#include „em_cmu.h”
void SystemInit(void)
{
  CMU_OscillatorEnable(cmuOsc_HFXO, 1, 1);
  CMU_ClockSelectSet(cmuClock_HF, cmuSelect_HFXO);
}
```

LÉTREHOZZUK A FORRÁSKÓD ÁLLOMÁNYOKAT

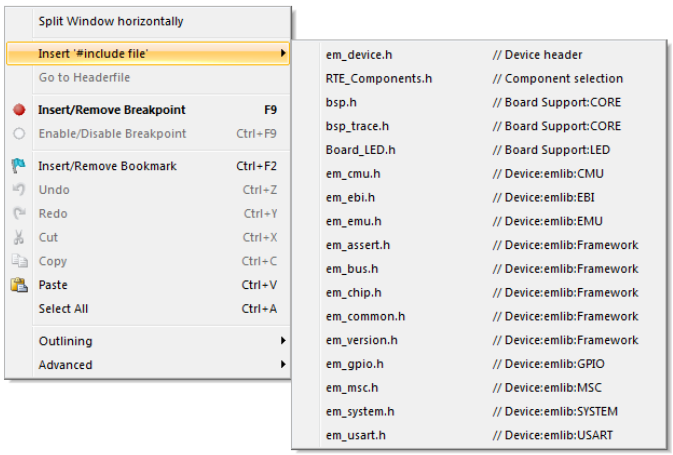
A projekt még nem tartalmaz *main()* függvényt, ezért létrehozunk egy üres C állományt és hozzá adjuk az alkalmazás kódokat. Az EFM32GG-STK3700 Starter kit 2 darab felhasználó által használható LED-et tartalmaz. Ezeket a LED-eket fogjuk felváltva villogtatni.



- A **Project** ablakban kattintson jobb gombbal a **Source Group 1** elemen majd válassza az **Add New Item to Group** lehetőséget.
- Kattintson a **C File (.c)** elemre és adja meg az állomány nevét *main.c*.

Ezzel hozzáadtuk a *main.c* állományt a **Source Group 1** csoporthoz. Most hozzáadhatjuk az alkalmazás specifikus kód részeket, kezdve a szükséges include állományokkal.

- Kattintson jobb gombbal a szerkesztő ablak üres területén és a megjelenő helyi menüben az **Insert '#include file'** almenüből válassza ki az **em_device.h**, **bsp.h,** és végül a **Board_LED.h** elemeket.



Megjegyzés: az **Insert '#include file'** almenüben csak azoknak a szoftver komponenseknek a header fájljai jelennek meg amik a **Manage Run-Time Environment** párbeszéd ablakban engedélyezve lettek korábban.

- Adja hozzá a következő kód részletet a *main.c* állományhoz az #include-ok után.

```
int32_t volatile msTicks = 0;


void SysTick_Handler (void) {
  msTicks++;
}


int main( void ) {


  LED_Initialize();
  LED_On(0);
  SysTick_Config(SystemCoreClock/1000);


  while(1) {
    BSP_LedToggle(0);
    BSP_LedToggle(1);


    while(msTicks < 499);
    msTicks = 0;
  }
}
```

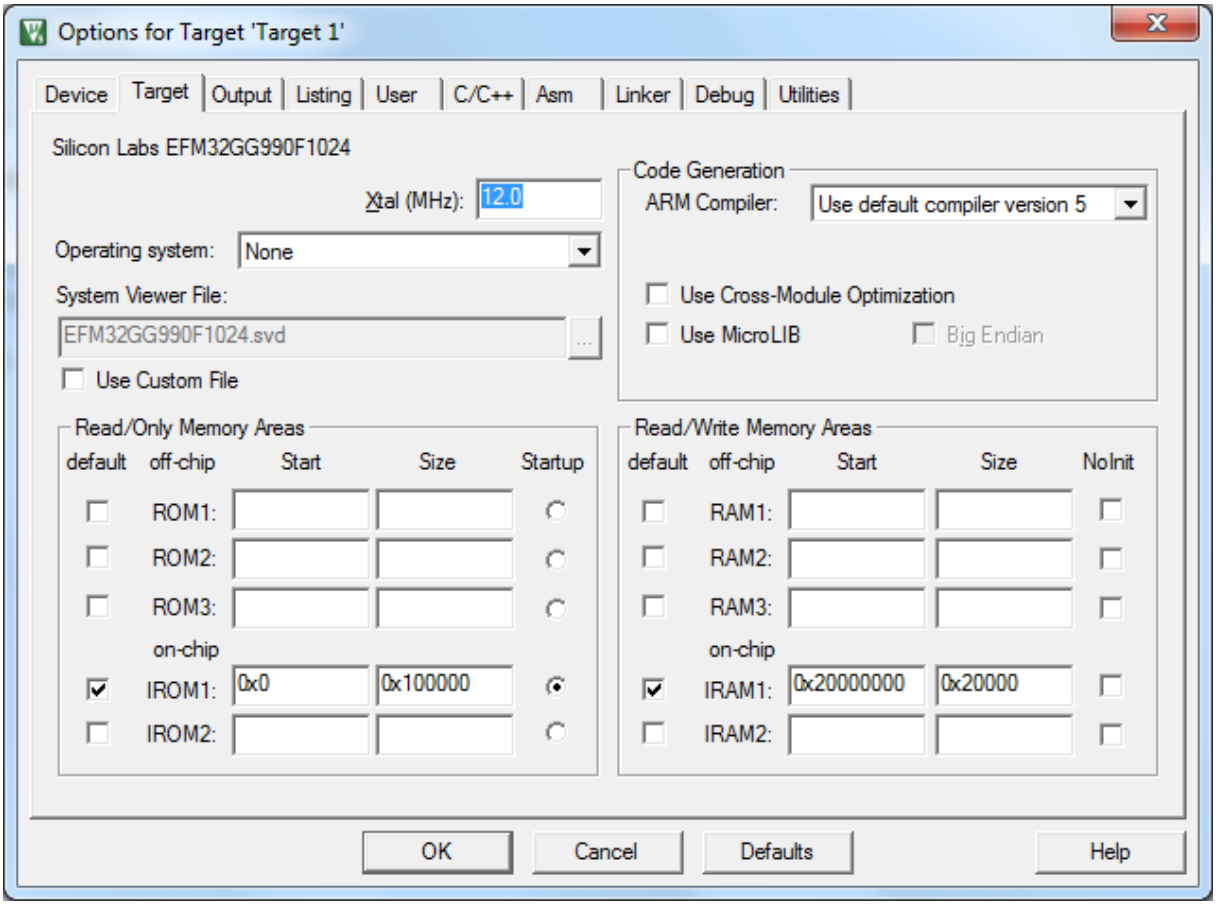
BEÁLLÍTJUK A FORDÍTÁSI OPCIÓKAT

A µVision menüsorból válassza ki a **Project – Options for Target 'Target 1'…** vagy kattintson az ikonsoron az **Options for Target** ikonra.

Itt megadhatjuk az eszközünk külső XTAL frekvenciáját. Az itt megadott értéket a debugger és a flash programozó használja, az alkalmazás működésére nincs hatással. Kiválaszthatjuk a projektben használt fordító verzióját, amennyiben a fejlesztő eszközhöz rendelkezésre áll több is. A µVision 5.25 két fordítót tartalmaz az 5.06 és az újként megjelenő 6.9-es verziót. Szükség esetén módosíthatjuk a ROM és RAM memória tartományokat. Alapértelmezésben ezek a tartományok lefedik a kiválasztott eszközben felhasználható ROM és RAM területeket.

Az STK3700 fejlesztő eszköz tartalmaz egy külső 48MHz-es kvarckristályt. A ROM és RAM tartományokat egyelőre változatlanul hagyjuk.

- Állítsa be az **Xtal (MHz)** értékét **48**-ra.



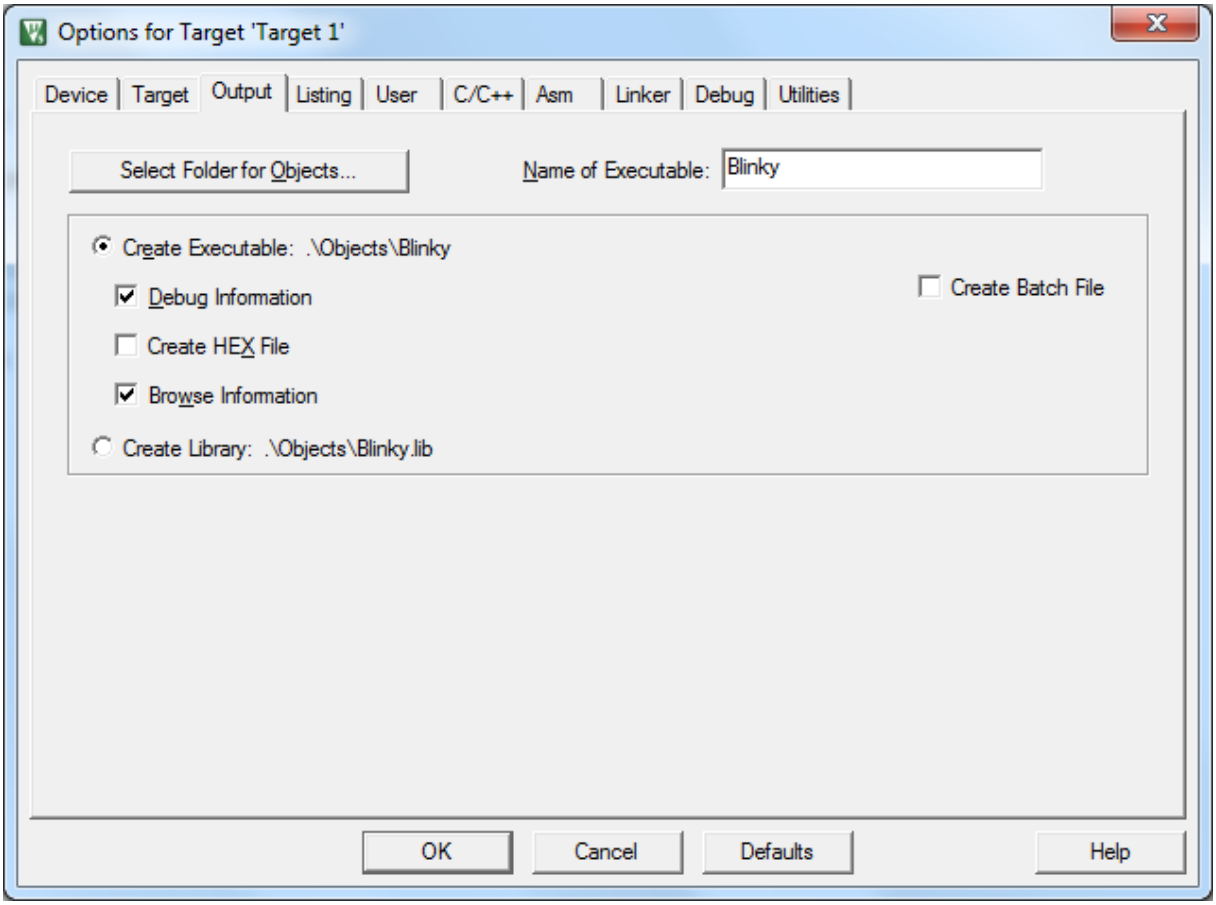
- Váltson át az **Output** fülre.

Ezen a fülön megadhatjuk a kimeneti állományok elérési útvonalát valamint azok nevét. Továbbá eldönthetjük, hogy futtatható kódot szeretnénk fordítani vagy egy **Library**-t akarunk létrehozni egyéb projektekhez.

Futtatható kódok fordítása esetén dönthetünk, hogy szükségünk van-e hibakereséshez **Debug Information** adatokra, sorozatgyártáshoz alkalmas intel hex formátumú állományra, valamint a **Source Browser**-hez szükséges adatokra.

Lehetőség van az adott projekthez létrehozni a fordítási parancsfájlt, ami lehetőséget nyújt arra, hogy a µVision elindítása nélkül tudjuk lefordítani újra a forrásállományokat.

Az **Output** fülön az alapértelmezett beállítások megfelelőek, nincs szükség módosításra.

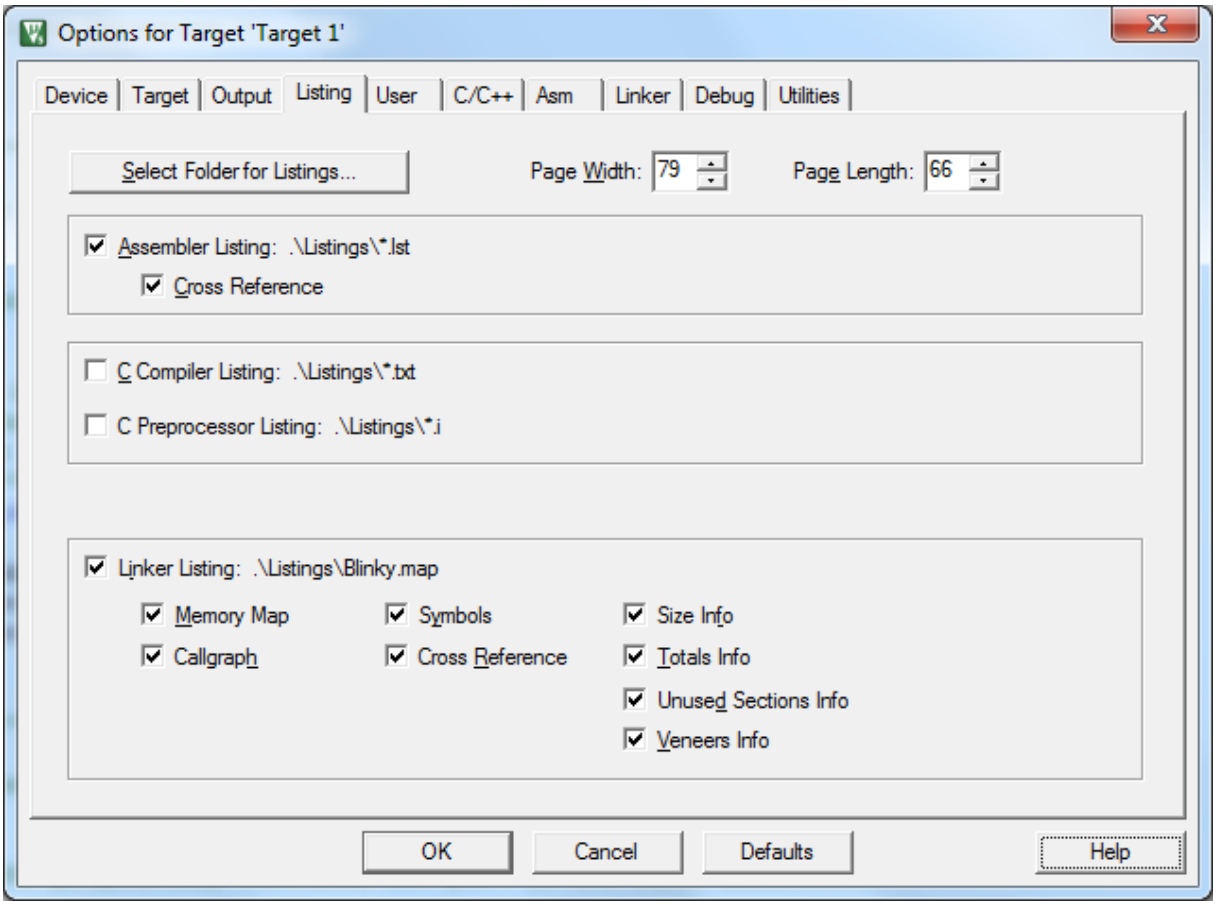


- Kattintson a **Listing** fülre

Ezen a fülön megadhatjuk, hogy a fordítás során, melyik fázisokról készítsen a fordító leíró állományt. Ezeken az állományokon keresztül nyomon követhető, hogy az adott forráskódból a fordító milyen assembler kódot fordít, anélkül, hogy Debug módba kellene vinni a rendszert.

A **Linker Listing** opciónál részletesen kiválaszthatjuk, hogy a linkelés egyes fázisairól keletkezzen-e feljegyzés a *.map* állományban.

A **Listing** fülön az alapértelmezett beállítások megfelelőek.

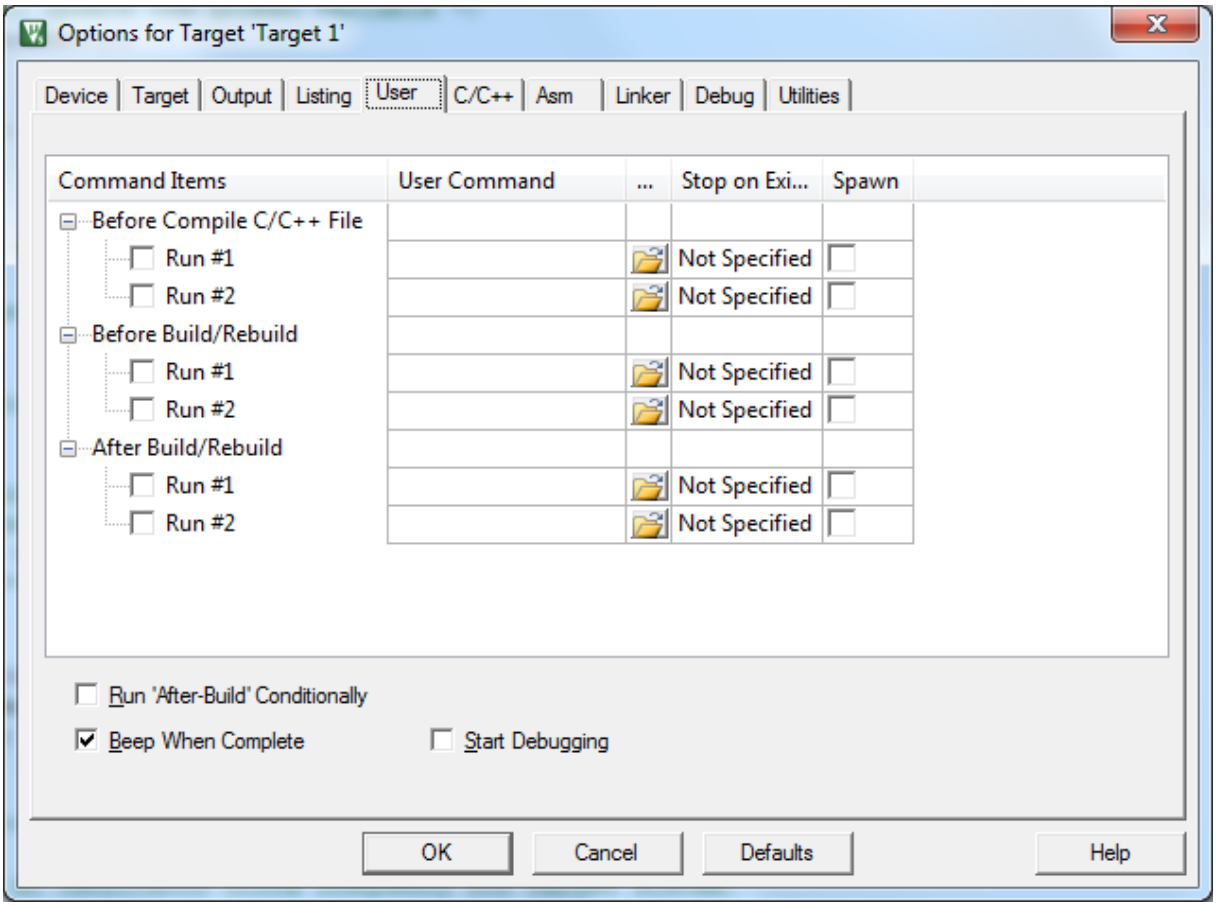


- Kattintson a **User** fülre

Felmerülhetnek olyan helyzetek, amikor a fordítás elvégzése előtt vagy után külső programok használata szükséges. Ezen programok automatikus lefuttatásának beállítására alkalmas a **User** fül.

Lehetőségünk van a C állományok lefordítása előtt, az alkalmazás fordítási folyamata előtt és után beállítani a végrehajtandó utasításokat.

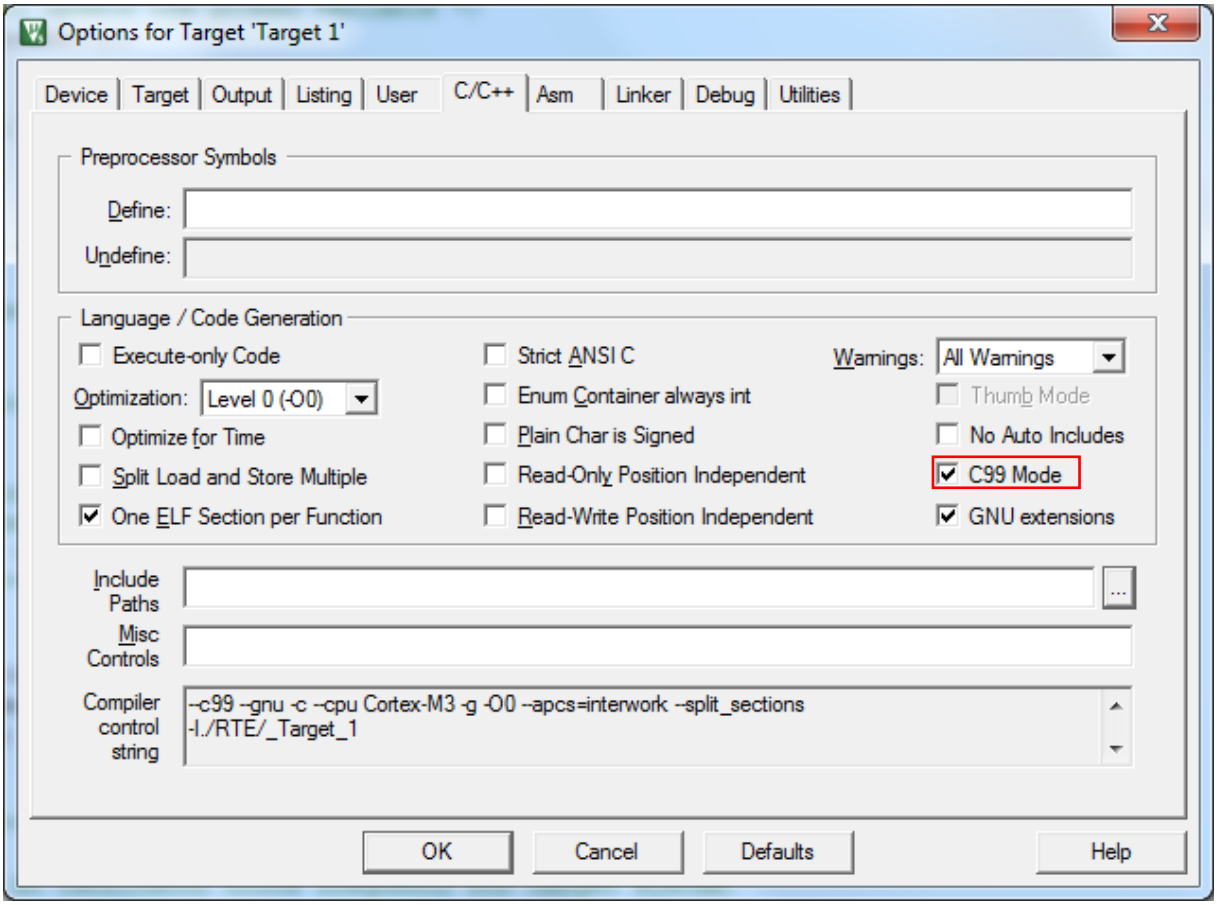
A **User** fülön módosításra nincs szükség.



- Kattintson a **C/C++** fülre

Ezen a fülön adhatjuk meg a teljes projektre vonatkozólag a **C/C++ fordító** alapértelmezett beállításait. Többek között a teljes projektre vonatkozólag adhatunk meg definíciókat. Továbbá a C/C++ fordító számos opcióját állíthatjuk be. Speciális elérési útvonalakat adhatunk meg. Továbbá van lehetőség a fent nem szereplő opciókon túl is befolyásolni a fordító futását. Legalul a projekt és a fent beállított opcióknak megfelelő sztring olvasható le.

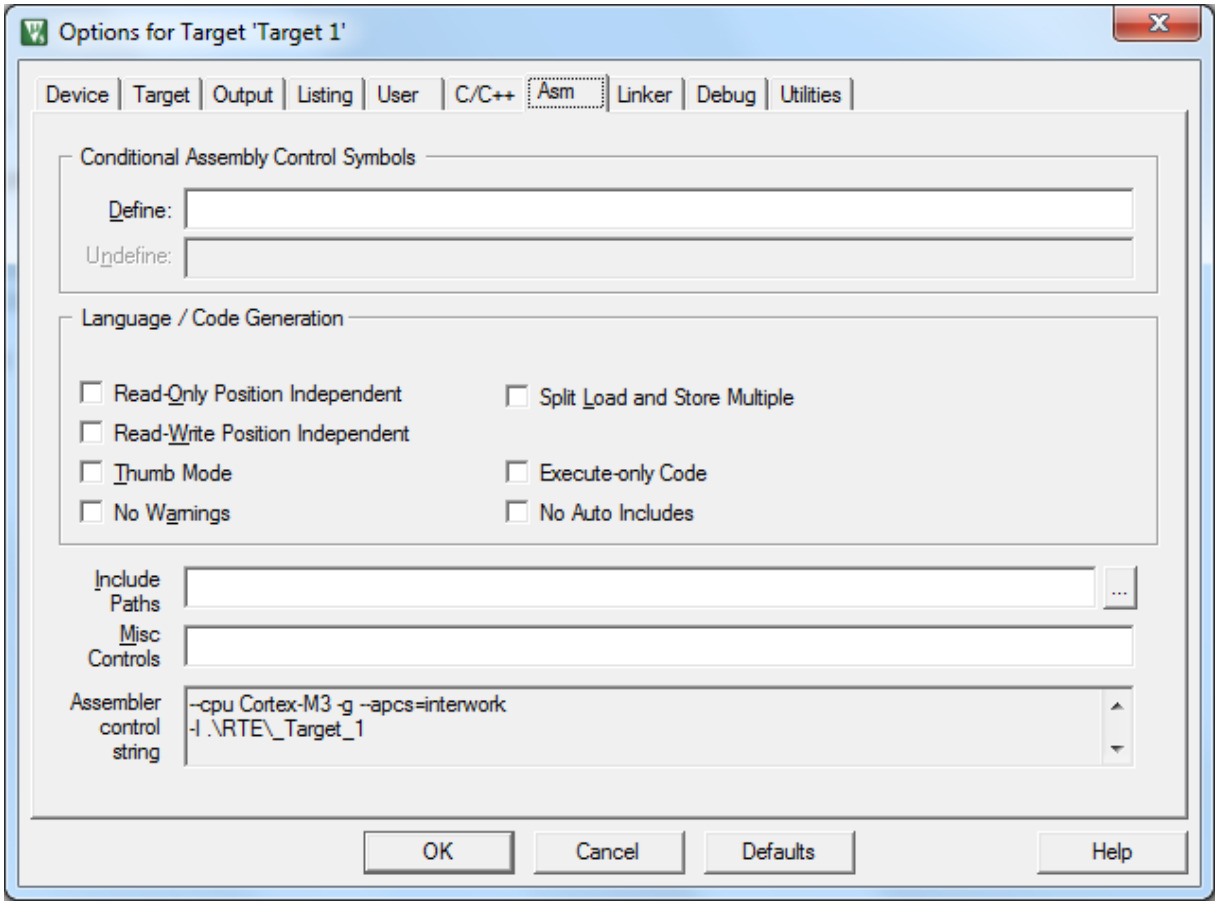
A Silicon Labs által elkészített **Board Support** csomagok a C99 szabvány lehetőségeit kihasználva készültek, ezért engedélyezni kell a **C99 Mode** opciót a sikeres fordítás érdekében. A többi beállítást egyelőre változatlanul hagyjuk.



- Kattintson az **Asm** fülre

Ezen a fülön - hasonlóan az előzőhöz, - a teljes projektre vonatkozólag állíthatjuk be az **Assembler** alapértelmezett beállításait. Ha a projektjeinket csak C állományokból építjük fel, akkor egyáltalán nem kell foglalkoznunk ezekkel a beállításokkal.

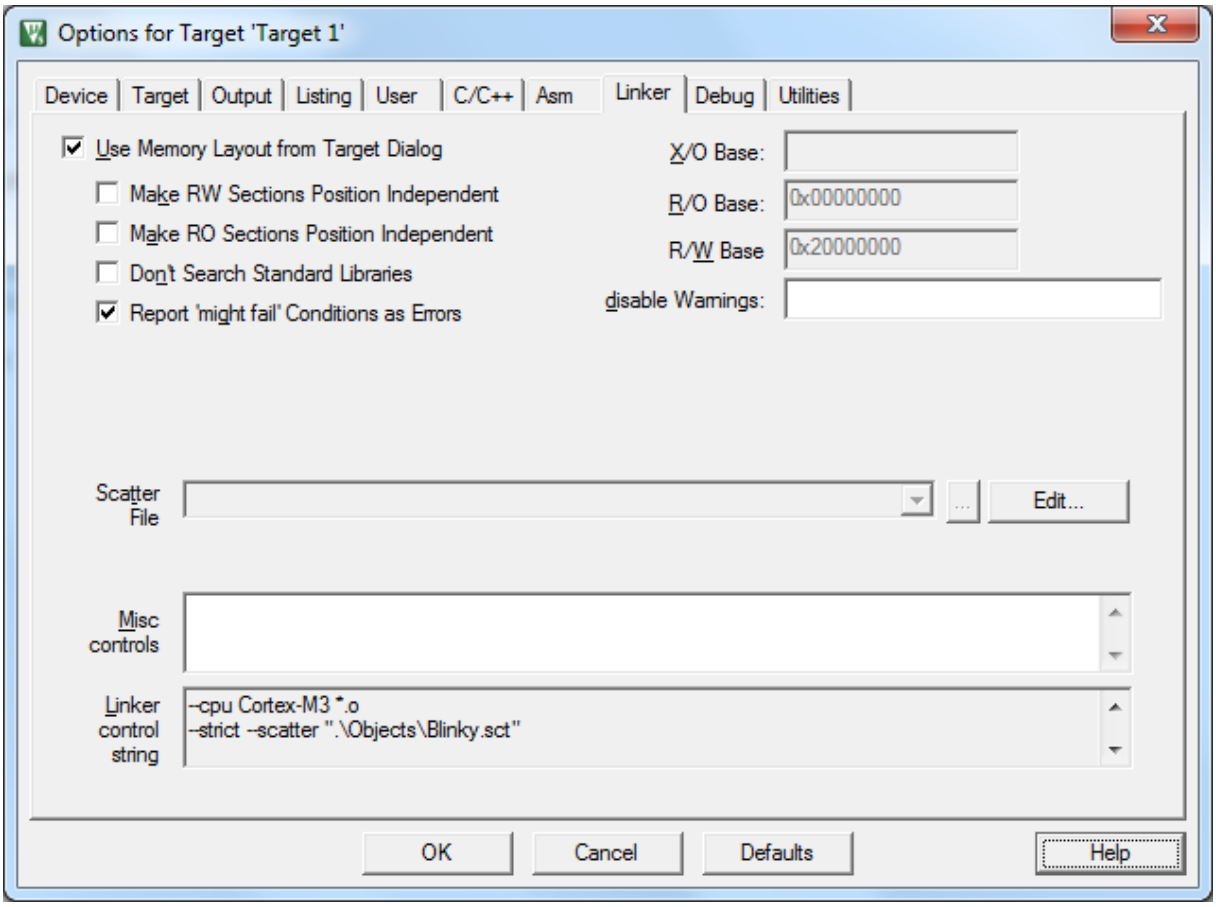
Jelenlegi projektben nincs szükség a beállítások módosítására



- Kattintson a **Linker** fülre

Ezen a fülön a **Linker** beállításait adhatjuk meg. Alapértelmezésben a **Use Memory Layout from Target Dialog** aktív, ami azt eredményezi, hogy a **Linker** a **Target** fülön megadott memória paraméterek alapján hozza létre a Scatter fájlt. Amennyiben ettől nem akarunk eltérni, akkor ezen a fülön további beállításra nincs szükség.

Összetettebb alkalmazásoknál szükség lehet olyan fokú módosításra a memória térképen, amit a **Target** fülön már nem lehet megadni. Ekkor ki kell kapcsolni a **Use Memory Layout from Target Dialog** opciót és kézzel létrehozni a szükséges Scatter fájlt.

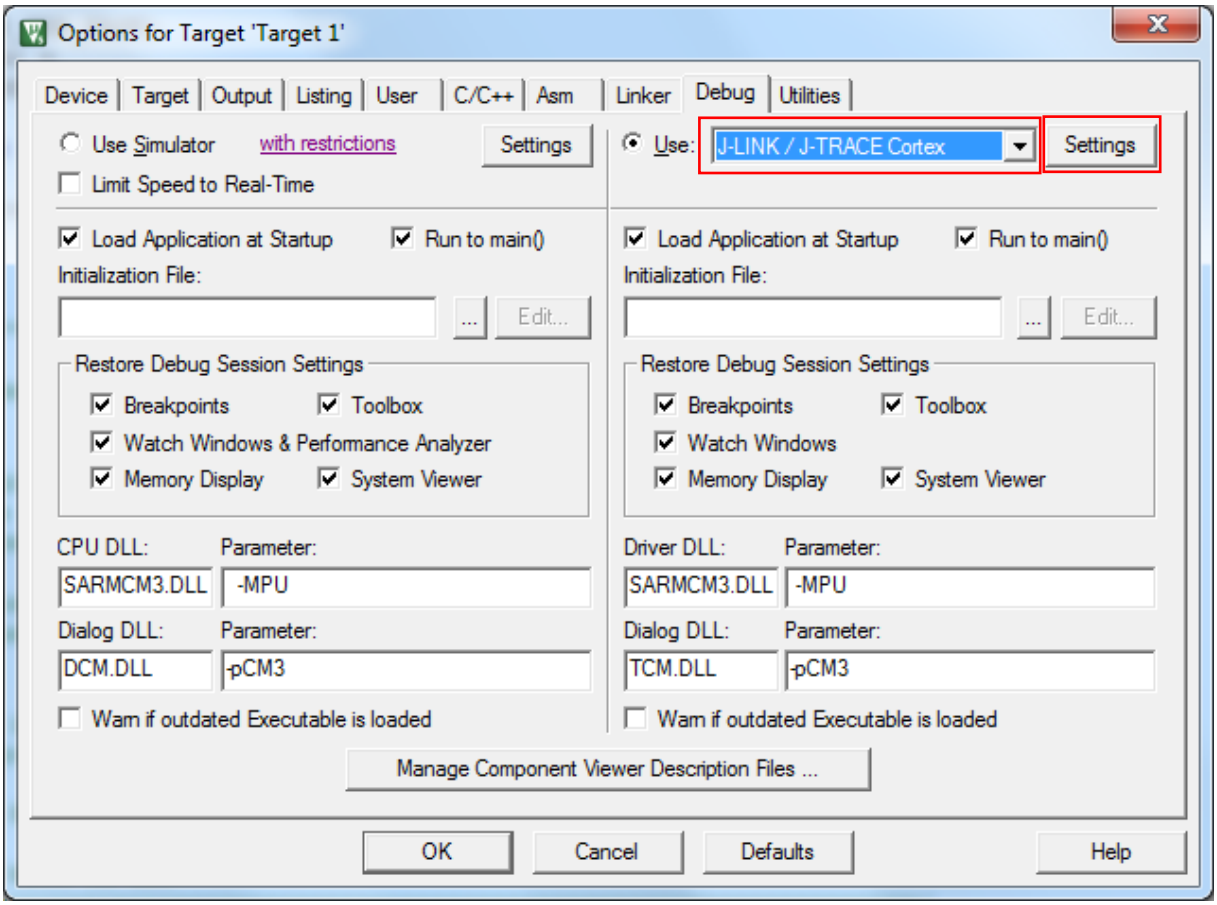


- Kattintson a **Debug** fülre

Ezen a fülön elsősorban eldönthetjük, hogy valós eszközön Debug adapter használatával vagy szimulátoron használatával akarjuk az alkalmazásunkat futtatni. A szimulátor használata korlátozásokkal lehetséges csak. Számos gyártó rendelkezésre bocsátja az adott eszközéhez a specifikus állományokat, de ha ezek nem állnak rendelkezésre, akkor ez a funkció csak általános leírókkal használhatók, amik nem tartalmazzák többek között az adott eszköz periféria címeit. Emiatt mondhatni a legegyszerűbb alkalmazást sem lehet futtatni. Viszont a szimulátor tökéletesen alkalmas akkor, ha valamilyen algoritmusfejlesztésről van szó, ami az adott eszköz perifériáihoz nem akar hozzáférni, csak bemeneti és kimeneti adatai vannak.

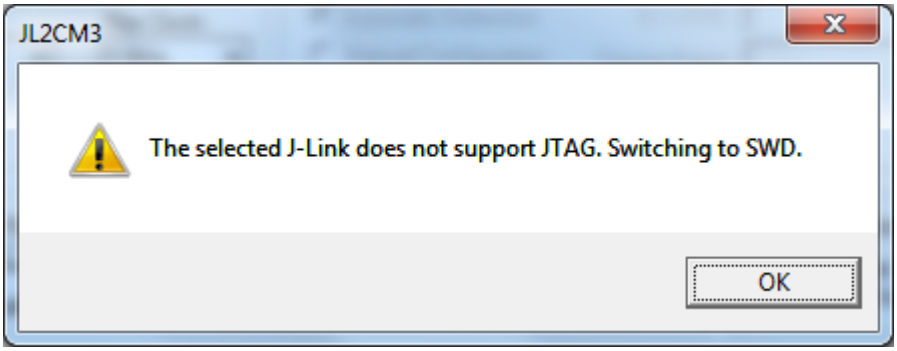
Jelen alkalmazásban rendelkezésre áll a fejlesztőkörnyezet ezért azt fogjuk használni. Az EFM32GG-STK3700 fejlesztőkörnyezeten integrálva van egy korlátozott funkcionalitású Segger J-Link debug adapter, ezért a lenyíló menüből a **J-LINK / J-TRACE Cortex** beállítást kell kiválasztani. Ez az adapter csak a **Serial Wire Debug** interfészt támogatja, a **JTAG** nem.

- Csatlakoztassa az EFM32GG-STK3700 fejlesztőkörnyezetet a PC-hez a **DBG USB port**on keresztül.
- Válassza ki a **J-LINK / J-TRACE Cortex** debug adaptert



- Kattintson a **Settings** gombra

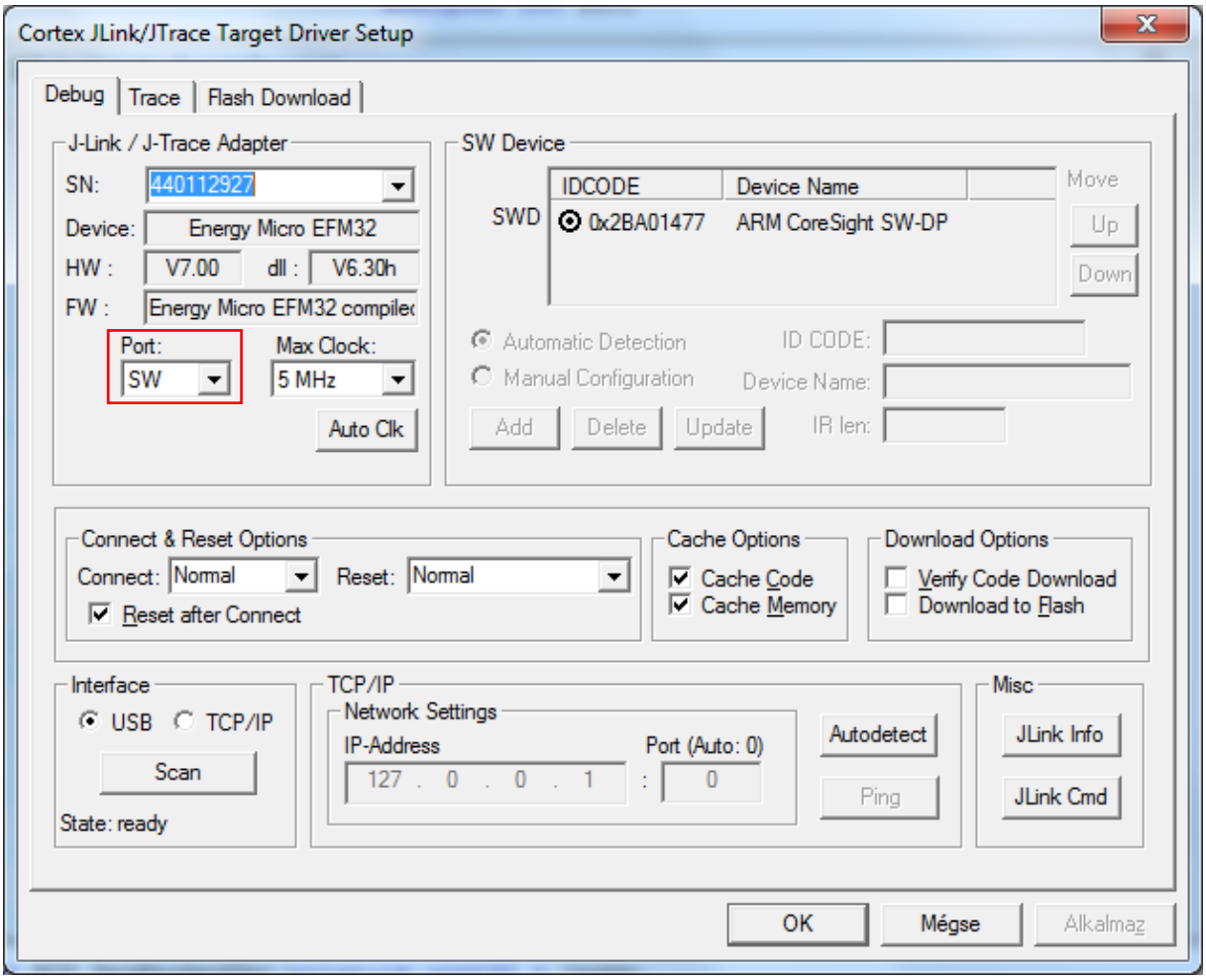
A beállítások ablak megjelenésekor a program rögtön figyelmeztet, hogy a csatlakoztatott debug adapter nem támogatja a JTAG módot ezért átvált SWD módra.



- Kattintson az **OK** gombra.

A **Debug** fülön a **J-Link / J-Trace Adapter** szekcióban láthatjuk a csatlakoztatott debug adapter paramétereit, valamint a kiválasztott interfészt és a sebességét. Az **SW Device** szekcióban pedig látható, hogy a Debug adapterre egy eszköz csatlakozik, ami az EFM32GG-STK3700 eszközön található EFM32GG990F1024 mikrovezérlő.

Jelen projekthez további beállításokra nincs szükség.



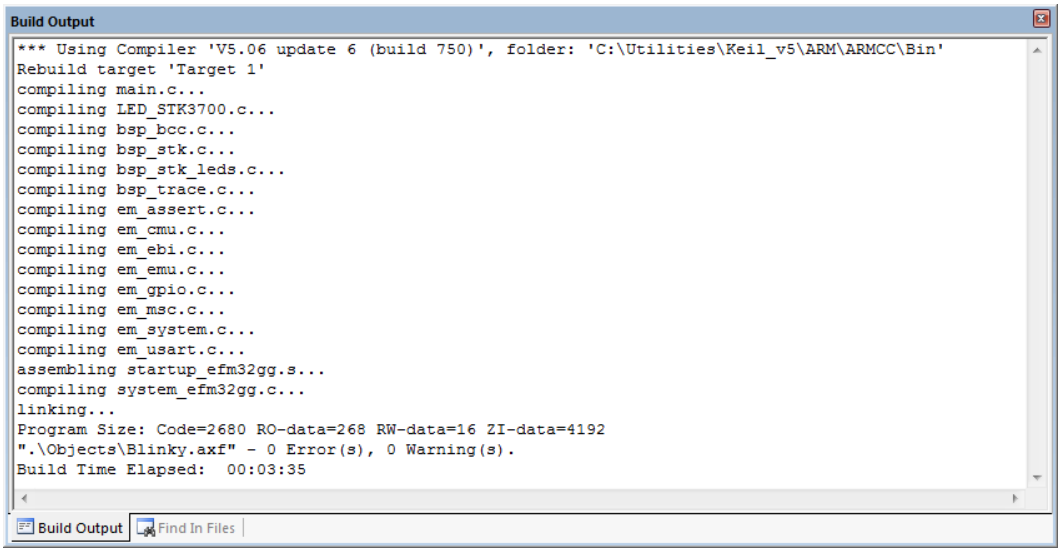
- Kattintson az **OK** gombra
- Majd az **Options for Target** ablakban is kattintson az **OK** gombra.

Ezzel el is végeztük a szükséges beállításokat

LÉTREHOZZUK AZ ALKALMAZÁST

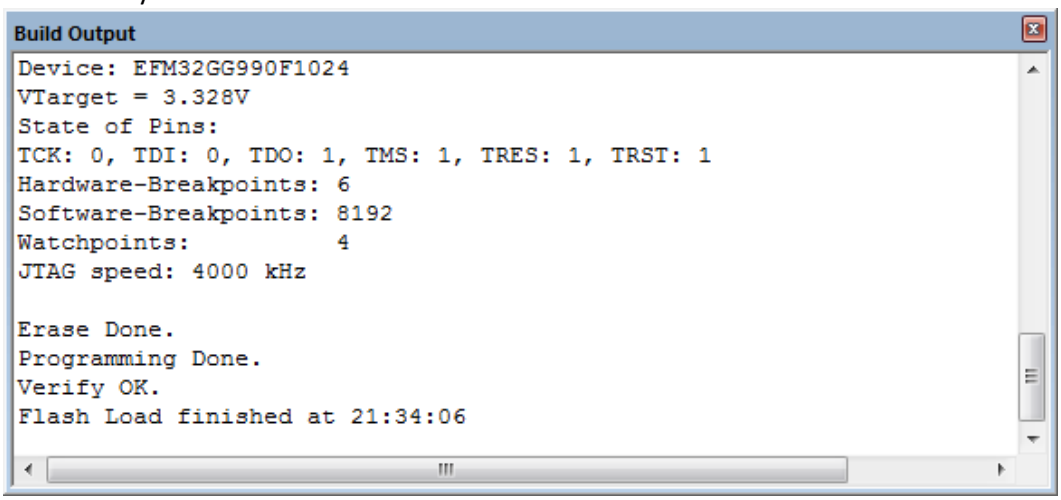
- Kattintson a **Rebuild**  ikonra.

Ez lefordítja és linkeli az összes forrás állományt. A **Bulid Output** ablakban információt kapunk a létrehozási folyamatról. Az ablakban a program mérete, nulla hiba és nulla figyelmeztetés látszik amennyiben a folyamat hibamentes zajlott le.



LETÖLTJÜK AZ ALKALMAZÁST

Az eszköztáron kattintson a **Download** ikonra. A **Build Output** ablakban nyomon követhetjük a letöltési folyamatot.



Az alkalmazás sikeres letöltését követően a mikrovezérlőt újra kell indítani, hogy az alkalmazás elinduljon.

- Nyomja meg a fejlesztőeszközön a **RESET** gombot

A LED-ek elkezdenek felváltva villogni 1 másodperces frekvenciával.

ALKALMAZÁSOK HIBAKERESÉSE

Az ARM CoreSight™ technológia integrálva van az ARM Cortex-M processzor alapú eszközökbe, ami sokoldalú debug és trace lehetőségeket nyújt. Ez lehetőséget biztosít a legtöbb eszközben a következőkre: program elindítása és leállítása, töréspontok kezelése, memória hozzáférés, és Flash programozás. További jellemzői a mintavétel, adat nyomkövetés, kivételkezelés, és utasítás nyomkövetés.

DEBUGGER KAPCSOLATOK

Az MDK magában foglalja a µVision Debugger-t, ami számos debug/trace adapterhez biztosítja a kapcsolatot, és lehetővé teszi a Flash memória programozását. Támogatja a megszokott funkciókat, úgymint egyszerű és feltételes töréspontok, watch ablak, végrehajtás vezérlés. A trace használatával további funkciók használhatók, mint az esemény / kivétel figyelés, logikai analizátor és forráskód felfedés.

A ULINK*plus* és ULINK2 debug adapterek alkalmasak a JTAG/SWD debug csatlakozásra és támogatják a trace funkciót a Serial Wire Output (SWO) csatornán keresztül. A ULINKpro debug/trace adapter szintén az ETM trace csatlakozón keresztül kapcsolódik és a folyamatos trace technológiát használja, ahhoz, hogy teljes utasítás nyomonkövetést tárolni tudja a forráskód felfedéshez.

CMSIS-DAP alapú USB JTAG/SWD debug felület általában a fejlesztői kártyák és starter kitek részét képezi. Az MDK számos szabadalmaztatott csatoló felületet támogat, amik hasonló technológiákat kínálnak.

Az MDK támogatja a harmadik féltől származó debug megoldásokat is, úgymint Segger J-Link vagy J-Trace. Néhány starter kit kártya J-Link Lite technológiát nyújt, mint beágyazott megoldás.

DEBUGGER HASZNÁLATA

A következőkben az előző fejezetben létrehozott LED villogó alkalmazást fogjuk először kiegészíteni. Üzembe helyezzük az LCD kijelzőt és a kapacitív érzékelőt, majd ezen az alkalmazáson bemutatásra kerülnek a debugger főbb funkciói.

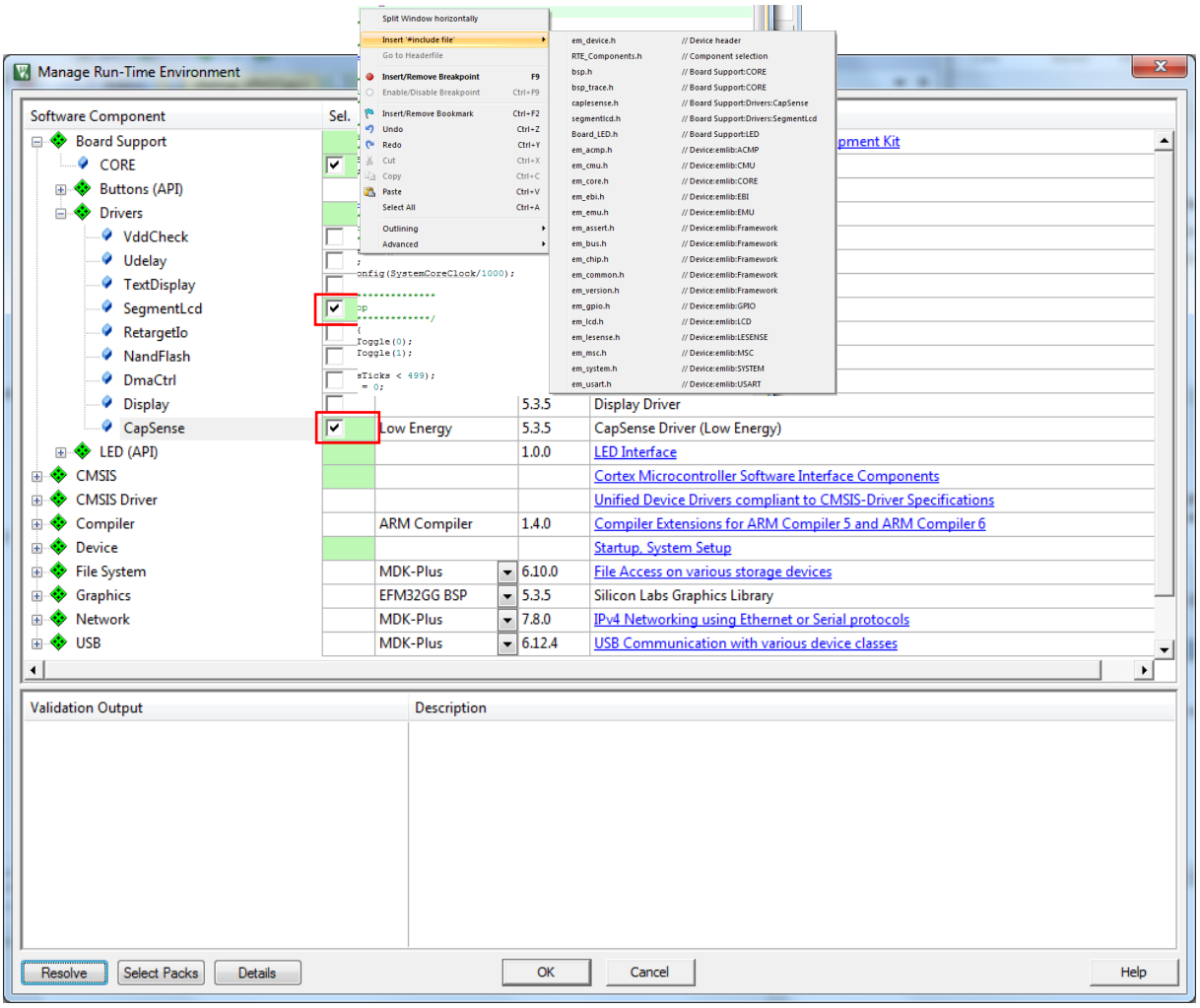
- **Hozzáadjuk a szükséges komponenseket a projekthez**
- **Létrehozzuk a forráskód állományokat**.
- **Létrehozzuk és letöltjük az alkalmazást**
- **Használjuk a debuggert**

HOZZÁADJUK A SZÜKSÉGES KOMPONENSEKET A PROJEKTHEZ

 Kattintson a **Manage Run-Time Environment** ikonra az eszköztáron

Megnyílik az ablak, amelyben korábban kiválasztottuk a LED villogó alkalmazáshoz a szükséges komponenseket.

- Nyissa ki **::Board Support:Drivers** és engedélyezze a **SegmentLCD** és **CapSense** elemeket.
- Kattintson a **ReSolve**-ra



- Kattintson az **OK**-ra

Ezzel hozzáadtuk azokat a gyártói támogató kódokat, amelyek gyorsítják és egyszerűsítik az alkalmazások létrehozását.

LÉTREHOZZUK A FORRÁSKÓD ÁLLOMÁNYOKAT

Ha nem a *main.c* állomány van megnyitva a szerkesztőben, akkor a **Project** ablakban a *main.c* állomány dupla kattintással megnyithatjuk.

- Kattintson duplán a *main.c* állományon
- Kattintson jobb gombbal a szerkesztő ablak **Includes** üres területén és a megjelenő helyi menüben az **Insert '#include file'** almenüből válassza ki az **caplesense.h** és **segmentlcd.h** elemeket.
- Adja hozzá a következő változó deklarációkat a **Variables** szekcióhoz

```
uint32_t slide_pos_last;
```

- Adja hozzá a következő kódrészletet a **Private functions** szekcióhoz

```
static void CapSenseScanCallback(void) {
  uint32_t slider_pos;
  slider_pos = CAPLESENSE_getSliderPosition();
  if( slider_pos != -1U ){
    slider_pos /= (0x30/7);

    if( slider_pos > slide_pos_last ) {
      while( slider_pos > slide_pos_last ) {
        SegmentLCD_ARing( slide_pos_last++, 1 );
      }
    } else if( slider_pos < slide_pos_last ) {
      while( slider_pos < slide_pos_last ) {
        SegmentLCD_ARing( slide_pos_last--, 0 );
      }
    }
  }
}
```

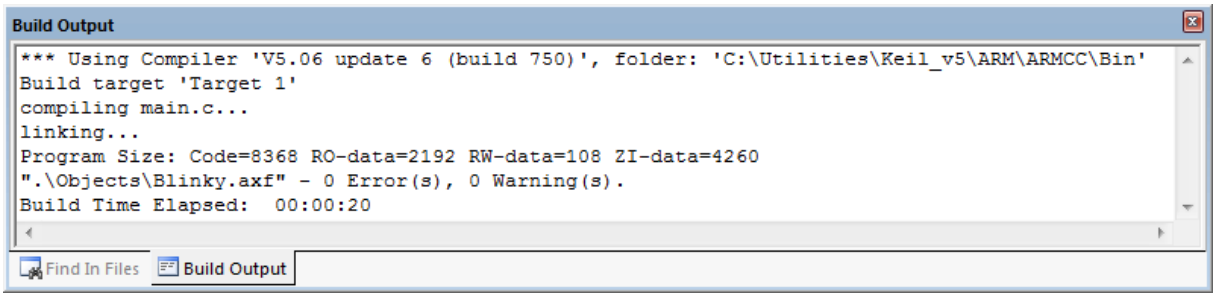
- Adja hozzá a következő kódrészletet az **Init** szekcióhoz

```
  SegmentLCD_Init(false);
  CAPLESENSE_setupCallbacks(CapSenseScanCallback, 0UL);
  CAPLESENSE_Init(false);
  SegmentLCD_Write("Control");
```

LÉTREHOZZUK ÉS LETÖLTJÜK AZ ALKALMAZÁST

- Kattintson a **Build**  ikonra.

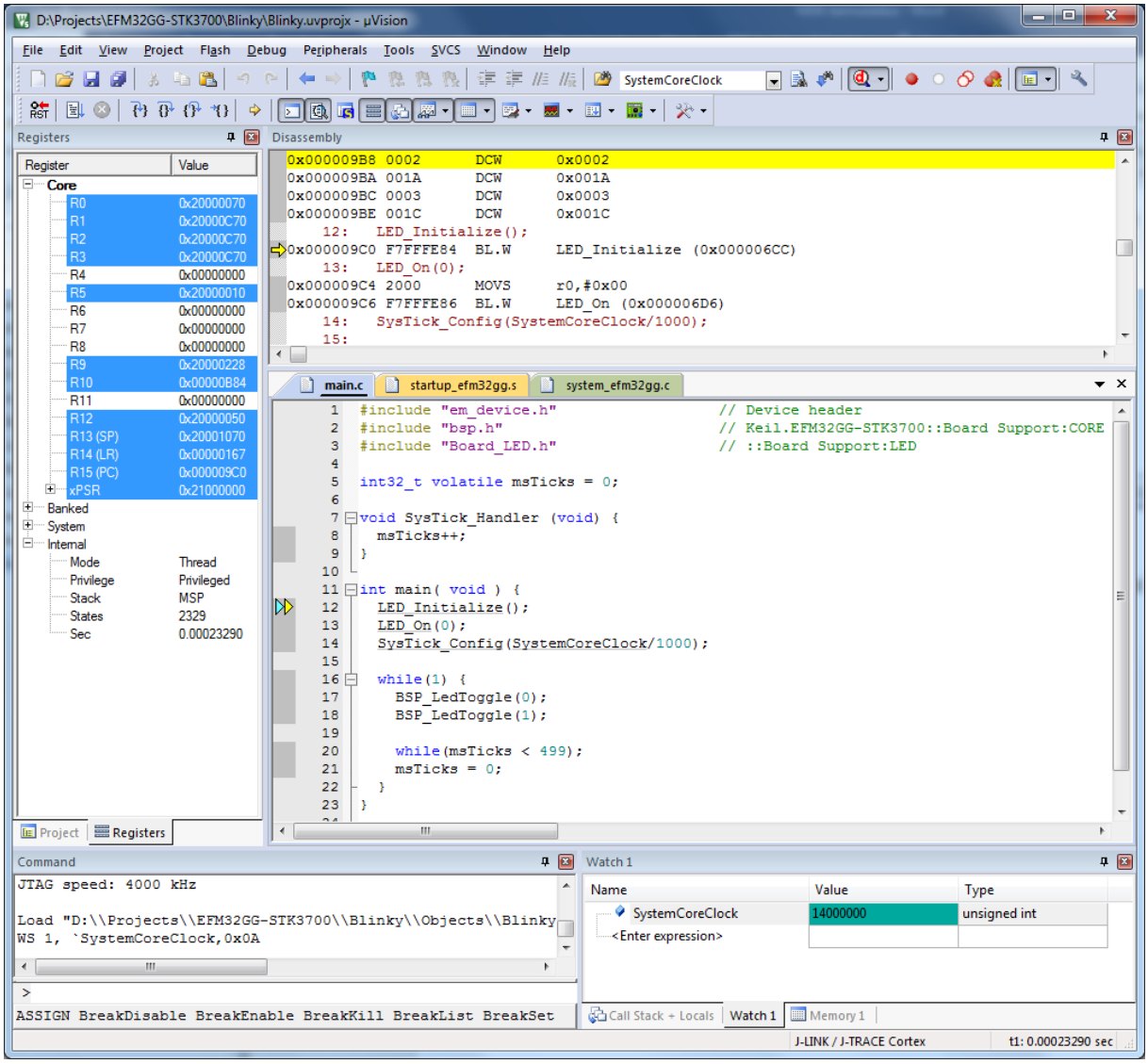
Ekkor a fordító környezet megvizsgálja, hogy mely forrásállományok változtak meg és csak azokat fordítja le újra, ezzel időt spórolva a fejlesztőnek.



- Kattintson a **Download**  ikonra.

HASZNÁLJUK A DEBUGGERT

 Indítsa el a hibakeresést a hardveren. Az eszköztáron kattintson a **Start/Stop Debug Session** ikonra.



A debugger elindulása közben a µVision betölti az alkalmazást, lefuttatja a startup kódot, és megáll a main függvényen.

A debugger indításakor szükség esetén a µVision újra letölti automatikusan az alkalmazást, ha az utolsó letöltés óta az megváltozott. A lefordított alkalmazást ellenőrzi nem pedig a forráskód állományokat, azaz nem fordítja le automatikusan a forráskódállományok változásait, erről a fejlesztőnek kell gondoskodni.

 Kattintson a **Run** ikonra az eszköztáron. A LED-ek elkezdenek felváltva villogni 1 másodperces frekvenciával, valamint a kijelzőn megjelenik a „Control” felirat. A kapacitív érzékelő megérintésével a kijelzőn lévő cikkekre osztott körgyűrű kitöltése állítható.

DEBUG ESZKÖZTÁR

A Debug eszköztár gyors hozzáférést biztosít a legfontosabb hibakeresési utasításokhoz, úgymint:

Step lépésenként (soronként) hajtja végre a programot, a függvényhívásokba is belépve.

Step Over lépésenként (soronként) hajtja végre a programot, a függvényhívásokat egy utasításként kezelve.

Step Out futtatja a programot az aktuális függvényhívás kilépésig.

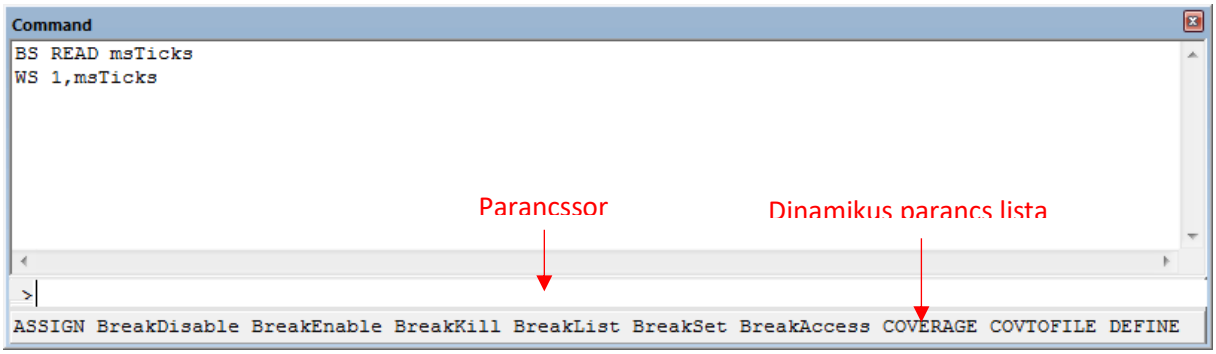
Stop megállítja a program végrehajtást.

Reset végrehajt egy CPU újraindítást.

Show a forráskódban a következő utasításra ugrik (aktuális PC regiszter).

COMMAND ABLAK

A **Command** ablak lehetővé teszi a debug utasítások parancssoros bevitelét, valamint információk kiírását futás időben.



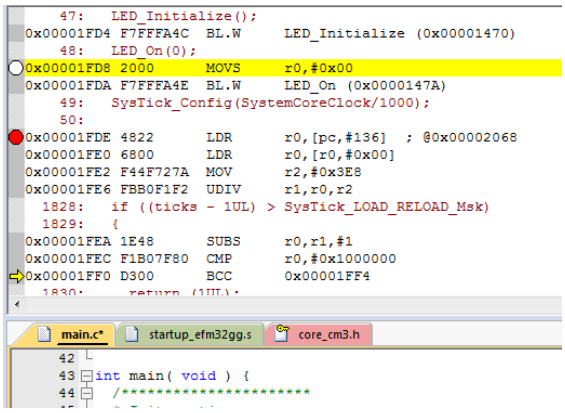
Az **F1** gomb megnyomásával részletes információkat kaphat a parancssor működéséről és az utasításokról.

A dinamikus parancs lista folyamatosan igazodik a parancssorba beírt betűkhöz, utasításokhoz. Érvényes utasítás beírását követően az utasításhoz tartozó következő paraméterről ad információt.

DISASSEMBLY ABLAK

A **Disassembly** ablakban látható a program végrehajtás assembly utasításai kiegészítve a forráskóddal (ahol ez lehetséges). Amikor ez az ablak aktív, a hibakeresési utasítások assembly szinten működnek.

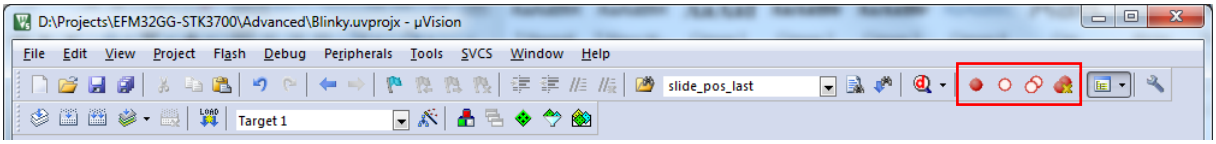
Az ablak margóján láthatók a töréspont, könyvjelző és a következő utasítást jelző szimbólumok.



TÖRÉSPONTOK

Töréspontokat állíthat be

- a forráskód állományok készítése és szerkesztése közben. Kattintson a szerkesztő vagy a **Disassembly** ablakban a szürke margóra a töréspont beállításhoz.
- a töréspont gombok használatával az eszköztárban.



- Debug:Breakpoints menü használatával.
- utasítás megadásával a **Command** ablakban.
- a Disassembly vagy szerkesztő ablakban a helyi legördülő menü használatával.
- az **F9** funkció billentyű használatával.

BREAKPOINTS ABLAK

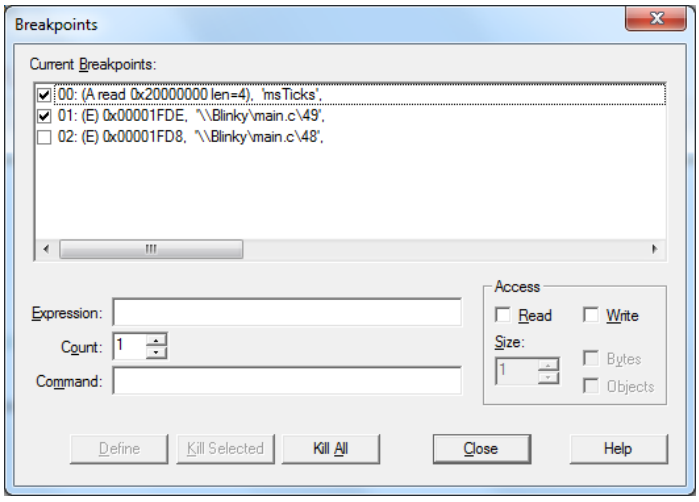
A **Breakpoints** ablak használatával összetett, feltételes töréspontokat definiálhat.

➢ Kattintson a **Debug:Breakpoints** menüpontra a megnyitáshoz

A **Current Breakpoints** ablakban a checkbox használatával engedélyezheti vagy letilthatja a töréspontokat. Egy meglévő töréspontonon duplán kattintva módosíthatja azt.

Egy új töréspont létrehozásához adjon meg egy **Expression**. A kifejezéstől függően az alábbi töréspont típusok jöhetnek létre:

- **Execution Breakpoint (E):** keletkezik amikor a kifejezés egy kód címet határoz meg és aktiválódik amikor a kód címet elérte.
- **Access Breakpoint (A):** keletkezik amikor a kifejezés egy memória hozzáférést határoz meg (olvasás, írás, mindkettő) és a memória cím hozzáférése aktiválja. Feltételes operátorok (==) használatával megadott értékkel összehasonlítva kerül kiértékelésre.



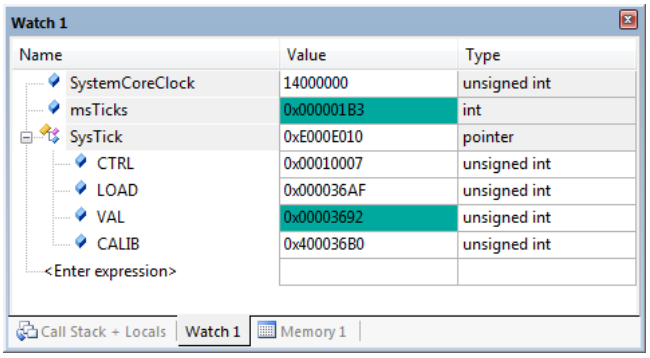
A **Command** mezőben megadott utasítás hatására, a µVision végrehajtja a megadott utasítást és folytatja a végrehajtást.

A **Count** érték meghatározza a töréspont kifejezés bekövetkezésének számát és csak ezután állítja le a program végrehajtását.

WATCH ABLAK

A **Watch** ablak lehetővé teszi, hogy megfigyeljen program szimbólumokat, regiszterek, memória területeket és kifejezéseket.

 Nyisson meg egy **Watch** ablakot az eszköztárról vagy használja a **View:Watch Windows** menüpontot.



A következő módokon adhat hozzá a **Watch** ablakhoz változókat:

- Kattintson a <Enter expression> mezőre, majd dupla kattintással vagy az **F2** funkcióbillentyűvel.
- A szerkesztőben állítsa a kurzort egy változóra, majd használja a helyi menüt és válassza ki a **Add *<változó név>* to…** menüt.
- Fogja meg és húzza a **Watch** ablakra a változót.
- **Command** ablakban használja a **WatchSet** utasítást.

Az ablak tartalma frissítésre kerül, amikor a program végrehajtás leáll, vagy program végrehajtás közben, ha a **View:Periodic Windows Update** engedélyezve van. A frissítés sebessége erősen függ a használt debugger típusától és sebességétől.

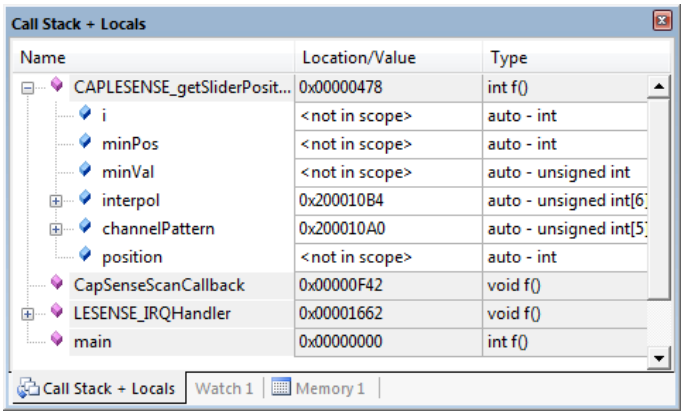
A **Value** mezőn dupla kattintással bármikor, akár futás közben is módosíthatók a változók értékei.

A **Watch** ablakból változót törölni csak a program végrehajtás leállítását követően lehet.

CALL STACK ÉS LOCALS ABLAK

A **Call Stack + Locals** ablak megmutatja a funkciók egymásba ágyazódását, és a változókat az adott programponton.

 Nyissa meg a **Call Stack + Locals** ablakot az eszköztárról vagy a **View:Call Stack Window** menüponttal.



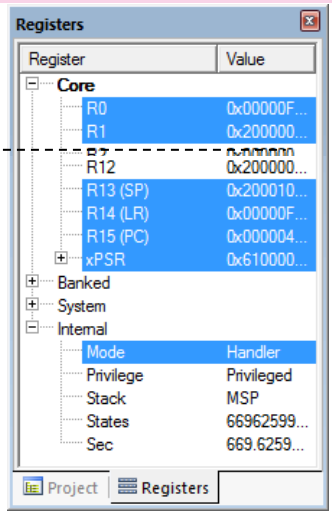
Amikor a program végrehajtás leáll, a **Call Stack + Locals** ablak automatikusan megmutatja az aktuális funkció egymásba ágyazását a helyi változókkal együtt.

Az egyes funkciók helyi menüjéből egyszerűen végig követhető, hogy az adott funkciót a program melyik pontjáról hívták (**Show Caller Code**) vagy az adott funkción belül hol történt a következő funkció hívása (**Show Callee Code**).

Ha CMSIS_RTOS RTX-et használ, akkor az egyes szálak változói meg vannak jelölve.

REGISTER ABLAK

A **Register** ablak megmutatja a mikrovezérlő regisztereinek tartalmát.



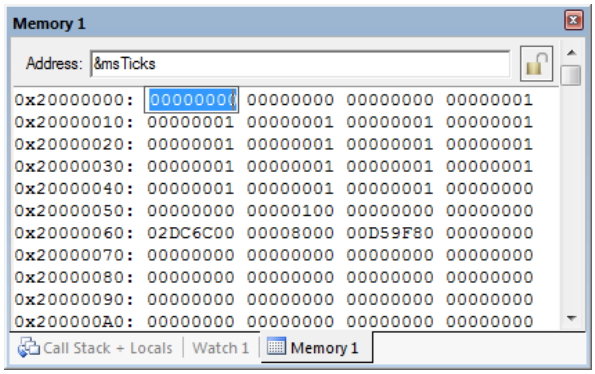
Nyissa meg a **Register** ablakot az eszköztárból vagy a **View:Register Windows** menüponttal.

A regiszter értékén dupla kattintással vagy az **F2** gomb megnyomásával bármikor, akár futás közben is, módosíthatja egy regiszter tartalmát. Az éppen megváltozott regiszterek kék színnel vannak kiemelve. Az ablakban látható értékek a program végrehajtás leállításakor frissülnek.

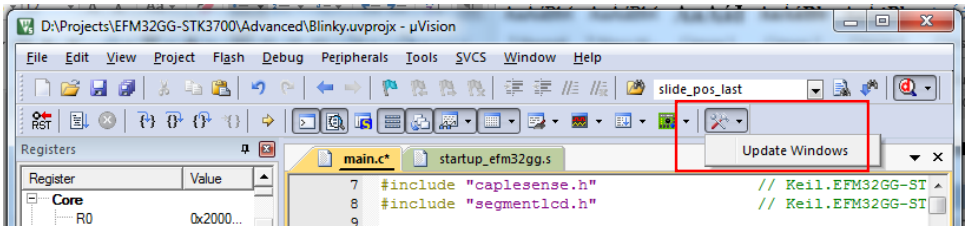
MEMORY ABLAK

Memória területek tartalmát figyelheti meg a **Memory Windows** használatával.


Nyisson meg egy **Memory** ablakot az eszköztárból vagy a **View:Memory Windows** menüponttal.



- Adjon meg egy kifejezést az **Address** mezőben, hogy figyelemmel tudja kísérni a memória területet.
- A memória tartalom megváltoztatásához használja a **Modify Memory at …** utasítást a **Memory** ablak helyi menüjéből vagy kattintson duplán az értékre vagy használja az **F2** billentyűt.
- A **Helyi Menü** lehetőséget biztosít a memória tartalom megjelenítési formátumának kiválasztására.
- A **Memory ablak** periodikus frissítéséhez engedélyezze a **View:Periodic Windows Update**. Az ablakok kézi frissítéséhez használja a **Toolbox:Update Windows** parancsot az eszköztáron.




Megállíthatja a **Memory** ablak frissítését a **Lock** gombra kattintással. Ezt a funkciót használhatja akár arra, hogy összehasonlítsa ugyanazon a memória címen lévő értékeket azáltal, hogy ugyanazt a memória szekciót jeleníti meg egy második **Memory** ablakban.

PERIFÉRIA REGISZTEREK

A periféria regiszterek memória címre leképzett regiszterek, amiket a processzor írhat és olvashat, hogy vezéreljen egy perifériát. A **Peripherals** menü biztosítja a **Core Peripherals** regiszterekhez való hozzáférést, úgymint Nested Vector Interrupt Controller (NVIC) vagy a System Tick Timer. Továbbá hozzáférhet az eszköz periféria regisztereihez a **System Viewer** használatával.

*Megjegyzés: a **Peripherals** menü tartalma a kiválasztott mikrovezérlőtől függ.*

SYSTEM VIEWER

A System Viewer ablakokban információt kaphat az eszköz periféria regisztereiről.

 Nyissa meg a GPIO periféria regisztert az eszköztárról vagy a **Peripheral:System Viewer** menüből.

A **System Viewer**-rel a következőket teheti:

- Megnézheti a periféria regiszter tulajdonságait és értékeit. Az értékek automatikusan frissülnek, amikor a **View:Periodically Windows Update** engedélyezve van.
- Megváltoztathatja a hozzátartozó értékeket hibakeresés közben.
- A kereső mezőben speciális tulajdonságokra kereshet a **TR1 Regular Expression** használatával. A *µVision User's Guide* melléklete ismerteti a reguláris kifejezések szintaxisát.