Az MDK-ARM bemutatása

SZERZŐ: FILEP GÁBOR



info@ccontrols.hu | www.ccontrols.hu

Az <u>MDK-ARM</u> 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

TARTALOMJEGYZÉK	3
FEjlesztőeszköz támogatás	4
Alkalmazások létrehozása	4
LED villogó	4
A Projekt beállítása	4
Beállítjuk az eszköz rendszer órajelét	7
Létrehozzuk a forráskód állományokat	8
Beállítjuk a fordítási opciókat	
Létrehozzuk az alkalmazást	20
Letöltjük az alkalmazást	20
Alkalmazások hibakeresése	21
Debugger kapcsolatok	21
Debugger használata	21
Hozzáadjuk a szükséges komponenseket a projekthez	21
Létrehozzuk a forráskód állományokat	22
Létrehozzuk a forráskód állományokat Létrehozzuk és letöltjük az alkalmazást	22
Létrehozzuk a forráskód állományokat Létrehozzuk és letöltjük az alkalmazást Használjuk a debuggert	22
Létrehozzuk a forráskód állományokat Létrehozzuk és letöltjük az alkalmazást Használjuk a debuggert Debug eszköztár	22 24 25 26
Létrehozzuk a forráskód állományokat Létrehozzuk és letöltjük az alkalmazást Használjuk a debuggert Debug eszköztár Command ablak	22
Létrehozzuk a forráskód állományokat Létrehozzuk és letöltjük az alkalmazást Használjuk a debuggert Debug eszköztár Command ablak Disassembly ablak	
Létrehozzuk a forráskód állományokat Létrehozzuk és letöltjük az alkalmazást Használjuk a debuggert Debug eszköztár Command ablak Disassembly ablak Töréspontok	
Létrehozzuk a forráskód állományokat Létrehozzuk és letöltjük az alkalmazást Használjuk a debuggert Debug eszköztár Command ablak Disassembly ablak Töréspontok Breakpoints ablak	
Létrehozzuk a forráskód állományokat Létrehozzuk és letöltjük az alkalmazást Használjuk a debuggert Debug eszköztár Command ablak Disassembly ablak Töréspontok Breakpoints ablak Watch ablak	
Létrehozzuk a forráskód állományokat Létrehozzuk és letöltjük az alkalmazást Használjuk a debuggert Debug eszköztár Command ablak Disassembly ablak Töréspontok Breakpoints ablak Watch ablak Call Stack és Locals ablak	
Létrehozzuk a forráskód állományokat Létrehozzuk és letöltjük az alkalmazást Használjuk a debuggert Debug eszköztár Command ablak Disassembly ablak Töréspontok Breakpoints ablak Watch ablak Call Stack és Locals ablak Register ablak	
Létrehozzuk a forráskód állományokat Létrehozzuk és letöltjük az alkalmazást Használjuk a debuggert Debug eszköztár Command ablak Disassembly ablak Disassembly ablak Breakpoints ablak Watch ablak Call Stack és Locals ablak Register ablak Memory ablak	
Létrehozzuk a forráskód állományokat Létrehozzuk és letöltjük az alkalmazást Használjuk a debuggert Debug eszköztár Command ablak Disassembly ablak Disassembly ablak Breakpoints ablak Kereakpoints ablak Call Stack és Locals ablak Register ablak Memory ablak Periféria Regiszterek	

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.

Add New Item to Group 'Source Gro	up 1'	x
C File (.c)	Create a new C source file and add it to the project.	
C++ File (.cpp)		
A Asm File (.s)		
h Header File (.h)		
Text File (.txt)		
Image File (.*)		
User Code Template		
	l	
Type: C File (.c)		
Name: main.c		
Location: D:\Projects\EFM3	2GG-STK3700\Blinky	
	Add Close He	lp

- 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.

	Split Window horizontally	
	Insert '#include file'	•
	Go to Headerfile	
•	Insert/Remove Breakpoint	F9
	Enable/Disable Breakpoint	Ctrl+F9
and the second		
5	Insert/Remove Bookmark	Ctrl+F2
5	Undo	Ctrl+Z
G,	Redo	Ctrl+Y
¥	Cut	Ctrl+X
þ	Сору	CtrI+C
2	Paste	Ctrl+V
	Select All	Ctrl+A
	Outlining	+
	Advanced	•
_		

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.

Options for Target 'Target 1'	×			
Device Target Output Listing User C/C++ Asm Linker Debug Utilities				
Silicon Labs EFM32GG990F1024				
<u>X</u> tal (MHz): 12.0 AR	M Compiler: Use default compiler version 5			
Operating system: None				
System Viewer File:	Use Cross-Module Optimization			
EFM32GG990F1024.svd	Use MicroLIB 🔽 Big Endian			
Use Custom File				
Read/Only Memory Areas Read	d/Write Memory Areas			
default off-chip Start Size Startup defau	It off-chip Start Size NoInit			
□ ROM1: □ □ □	RAM1:			
□ ROM2: □ □ □	RAM2:			
□ ROM3: □ □ □	RAM3:			
on-chip	on-chip			
IROM1: 0x0 0x100000 € ▼	IRAM1: 0x2000000 0x20000 □			
□ IROM2: □ □	IRAM2:			
OK Cancel	Defaults Help			

> 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.

Options for Target 'Target 1'	×
Device Target Output Listing User C/C++ Asm Linker Debug Utilities	
Select Folder for Objects Name of Executable: Blinky	
✓ Debug Information	Create Batch File
Create HEX File	
✓ Browse Information	
C Create Library: .\Objects\Blinky.lib	
OK Cancel Defaults	Help

> 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.

Options for Target 'Target 1'	
Device Target Output Listing User C/C++ Asm Linker Debug Utilities	
Select Folder for Listings Page Width: 79 + Page Length: 66 +	
Assembler Listing: .\Listings*.lst Cross Reference	
☐ <u>C</u> Compiler Listing: .\Listings*.txt ☐ C Preprocessor Listing: .\Listings*.i	
Linker Listing: .\Listings\Blinky.map	
Memory Map Symbols Size Info	
Callgraph Cross <u>R</u> eference <u>T</u> otals Info	
✓ Unuse <u>d</u> Sections Info	
✓ Veneers Info	
OK Cancel Defaults Help	

> 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.

Options for Target 'Target 1'				
Device Target Output Listing User C/C++ Asm Linker Debug Utilities				
Command Items	User Command	Stop on Exi	Spawn	
Before Compile C/C++ File				
		Not Specified		
🗌 🗌 Run #2		Not Specified		
Before Build/Rebuild				
	<i>iii</i>	Not Specified		
Run #2	<i>iii</i>	Not Specified		
□ After Build/Rebuild		I Not Constitute		
□ Kun #1		Not Specified		
1 Null #2		I not specified		
Run 'After-Build' Conditionally				
Reen When Complete	Start Debugging			
. <u>D</u> op miler complete	, <u>o</u> tart bobugging			
	OK Cancel	Defaults	Help	

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 <u>Silicon Labs</u> 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.

Options for Target 'Target 1'	×
Device Target Output Listing Us	er C/C++ Asm Linker Debug Utilities
Preprocessor Symbols	
Define:	
U <u>n</u> define:	
Language / Code Generation	
Execute-only Code	Strict <u>A</u> NSIC <u>W</u> amings: All Wamings ▼
Optimization: Level 0 (-00)	Enum Container always int 📃 Thumb Mode
Optimize <u>f</u> or Time	Plain Char is Signed No Auto Includes
Split Load and Store Multiple	Read-Only Position Independent 🔽 C99 Mode
✓ One ELF Section per Function	☐ <u>R</u> ead-Write Position Independent GNU extensions
Misc I	
Controls	
Compiler control string	ex-M3 -g -O0apcs=interworksplit_sections
	OK Cancel Defaults Help

> 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.

V Options for Target 'Target 1'
Device Target Output Listing User C/C++ Asm Linker Debug Utilities
Conditional Assembly Control Symbols
Define:
Undefine:
Language / Code Generation
Read-Qnly Position Independent Split Load and Store Multiple Read-Write Position Independent
Thumb Mode Execute-only Code
No Wamings No Auto Includes
Include Paths
Misc Controls
Assembler control string
OK Cancel Defaults Help

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.

Options for	Target 'Target 1'			x
Device Targe	t Output Listing User C/C++ Asm	Linker Debug	Utilities	
I Use Mem I Ma <u>k</u> e I Ma <u>k</u> e I Make I Do <u>n</u> t I Repo	ory Layout from Target Dialog RW Sections Position Independent RO Sections Position Independent Search Standard Libraries t 'might fail' Conditions as Errors	<u>X</u> /O Base: <u>R</u> /O Base: R/ <u>W</u> Base <u>d</u> isable Warnings:	0x0000000 0x20000000	
Scatter File			.	Edit
<u>M</u> isc controls				*
Linker control string	cpu Cortex-M3 *.o strictscatter ".\Objects\Blinky.sct"			Ţ
	ОК	Cancel Defa	aults	Help

> 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 porton keresztül.
- > Válassza ki a J-LINK / J-TRACE Cortex debug adaptert

Options for Target 'Target 1'	×
Device Target Output Listing User C/C++ Asm O Use Simulator with restrictions Settings Imit Speed to Real-Time Settings Settings	Linker Debug Utilities
Load Application at Startup Run to main() Initialization File: Edit	Load Application at Startup Run to main() Initialization File: Edit
Restore Debug Session Settings Image: Breakpoints Image: Toolbox Image: Watch Windows & Performance Analyzer Image: Memory Display Image: System Viewer	Restore Debug Session Settings Image: Breakpoints Image: Toolbox Image: Watch Windows Image: Memory Display Image: System Viewer
CPU DLL: Parameter: SARMCM3.DLL -MPU	Driver DLL: Parameter: SARMCM3.DLL -MPU
Dialog DLL: Parameter: DCM.DLL -pCM3	Dialog DLL: Parameter: TCM.DLL PCM3
Wam if outdated Executable is loaded Manage Component Vie	Wam if outdated Executable is loaded
OK Car	ncel Defaults Help

> 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.

JL2CM3	×
	The selected J-Link does not support JTAG. Switching to SWD.
	ОК

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.

Cortex JLink/JTrace Target Driver Setup	X
Debug Trace Flash Download	
J-Link / J-Trace Adapter SW D	Device
SN: 440112927	IDCODE Device Name Move
Device: Energy Micro EFM32 SV	WD Ox2BA01477 ARM CoreSight SW-DP
HW : V7.00 dll : V6.30h	Down
FW : Energy Micro EFM32 compiles	Automatic Detection ID CODE:
SW V 5 MHz V C M	Annual Configuration Device Name:
Auto Clk	dd Delete Update IR len:
Connect & Reset Ontions	Cache Ontions Download Ontions
Connect: Normal Reset: Normal	
✓ <u>R</u> eset after Connect	Cache Memory Download to Flash
_ Interface TCP //P	
USB O TCP/IP Network Settings -	Mited at a state of the state o
Scan IP-Address	Port (Auto: 0)
State: ready	Ping JLink Cmd

- > 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.

Build Output	X
Device: EFM32GG990F1024	*
VTarget = 3.328V	
State of Pins:	
TCK: 0, TDI: 0, TDO: 1, TMS: 1, TRES: 1, TRST: 1	
Hardware-Breakpoints: 6	
Software-Breakpoints: 8192	
Watchpoints: 4	
JTAG speed: 4000 kHz	
Erase Done.	-
Programming Done.	-
Verify OK.	=
Flash Load finished at 21:34:06	
	Ŧ
4 III II	

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

	5							
		Insert '#include file'		•	em_device.h	// Device header		
		Go to Headerfile			RTE_Components.h	// Component selection		
Manage Run-Time Environment		Insert/Remove Breakpoint		F9	bsp.h	// Board Support:CORE		
		Enable/Disable Breakpoint	Ct	rl+F9	bsp_trace.h	// Board Support:CORE		
Software Component	Sel.	Insert/Remove Bookmark	Ct	rl+F2	caplesense.h	// Board Support:Drivers:CapSense		
		🤊 Undo	¢	trl+Z	Board_LED.h	// Board Support:LED	. 125	
Board Support		🎽 Redo	9	Ctrl+Y	em_acmp.h	// Device:emlib:ACMP	pment Kit	
CORE	•	5 Cut		trl+X	em_cmu.h	// Device:emlib:CMU		
🗉 🚸 Buttons (API)		B Paste	0	tri+V	em_core.h	// Device:emlib:CORE		
Drivers		Select All	c	tri+A	em_emu.h	// Device:emlib:EMU		
VddCheck		Outlining		•	em_assert.h	// Device:emlib:Framework		
Vuucheck		Advanced		•	em_bus.h	// Device:emlib:Framework		
Udelay	; 	fig(SystemCoreClock/10	; (00		em_chip.h	// Device:emlib:Framework // Device:emlib:Framework		
TextDisplay					em_version.h	// Device:emlib:Framework		
SegmentLcd	V op				em_gpio.h	// Device:emlib:GPIO		
RetargetIo		/			em_lcd.h	// Device:emlib:LCD // Device:emlib:LESENSE		
NandElash	- Io	ggle(0); ggle(1);			em_msc.h	// Device:emlib:MSC		
		icks < 499):			em_system.h	// Device:emlib:SYSTEM		
DmaCtrl		0;		_	em_usart.h	// Device:emlib:USART		
Display		-		5.3.5	Display Driv	er		
CapSense		Low Energy		5.3.5	CapSense D	river (Low Energy)		
🖃 🚸 LED (API)				1.0.0	LED Interfac	<u>e</u>		
🗄 💠 CMSIS					Cortex Micr	ocontroller Software Inter	face Components	
😥 💠 CMSIS Driver					Unified Device Drivers compliant to CMSIS-Driver Specifications		MSIS-Driver Specifications	
🗄 💠 Compiler		ARM Compiler		1.4.0	.0 Compiler Extensions for ARM Compiler 5 and ARM Compiler 6		ler 5 and ARM Compiler 6	
🗄 💠 Device					Startup, System Setup			
😥 🚸 File System		MDK-Plus	•	6.10.0) File Access	on various storage device	5	
🗄 💠 Graphics		EFM32GG BSP	•	5.3.5	Silicon Labs	Graphics Library	-	
🖶 💠 Network		MDK-Plus	•	7.8.0	IPv4 Netwo	rking using Ethernet or Se	rial protocols	
🗄 🚸 USB		MDK-Plus	•	6.12.4	USB Comm	unication with various de	vice classes	
•			_					
			_					
Validation Output		Description						
Resolve Select Packs Details				OK	Cance		Help	

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 ) {</pre>
      while( slider pos < slide pos last ) {</pre>
        SegmentLCD ARing( slide pos last--, 0 );
      }
    }
  }
  Adja hozzá a következő kódrészletet az Init
                                         szekcióhoz
  SegmentLCD Init(false);
```

CAPLESENSE_setupCallbacks(CapSenseScanCallback, OUL); 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.

D:\Projects\EFM3	2GG-STK3700\Blinky	ABlinky.uvprojx - µVision	
File Edit View	Project Flash De	ebug Peripherals Tools SVCS Window Help	
	XBRIO	은 (_	
	÷ 0 0 0 0		
Registers	д 💽	Disassembly 4	
Register	Value	0x00009B8 0002 DCW 0x0002	<u> </u>
E Core R0 R1 R2 R3	0x20000070 0x20000C70 0x20000C70 0x20000C70	0x00009BA 001A DCW 0x001A 0x00009BC 0003 DCW 0x0003 0x00009BE 001C DCW 0x001C 12: LED_Initialize(); ©0x00009C0 F7FFFE84 BL.W LED_Initialize (0x00006CC)	
R4 R5 R6 R7 R8	0x0000000 0x20000010 0x00000000 0x00000000 0x00000000	<pre>13: LED_On(0); 0x000009C4 2000 MOVS r0,#0x00 0x000009C6 F7FFE86 BL.W LED_On (0x000006D6) 14: SysTick_Config(SystemCoreClock/1000); 15:</pre>	4
R9	0x20000228		
Image: Right of the second	C0000000 0x2000050 0x2000050 0x2000050 0x20000167 0x00000167 0x00000167 0x0000000 0x21000000	<pre>imank Gatup_ems2gg: // Device header finclude "bm_device.h" // Keil.EFM32GG-STK3700::Board Support:CORE finclude "board_LED.h" // Keil.EFM32GG-STK3700::Board Support:CORE finclude "board_LED.h" // ::Board Support:LED fints2_t volatile msTicks = 0; fint main(void) { If Dint main(void) { LED_Initialize(); LED_Initialize(); LED_On(0); SysTick_Config(SystemCoreClock/1000); Set = 10000; Mile(1) { MST_LedToggle(0); BST_LedToggle(1); msTicks = 0; 20 while(msTicks < 499); msTicks = 0; 22 . } 23 } </pre>	
🖭 Project 🧱 Regi	isters	•	
Command		P Watch 1 P	
JTAG speed: 40 Load "D:\\Pro; WS 1, `System(>	000 kHz jects\\EFM32GG- CoreClock,0x0A	-STK3700\\Blinky\\Objects\\Blinky	
ASSIGN BreakDi	isable BreakEna	able BreakKill BreakList BreakSet 🛛 🖓 Call Stack + Locals Watch 1 🔲 Memory 1	
		J-LINK / J-TRACE Cortex t1: 0.00023290 sec	:

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.

Command	
BS READ msTicks	*
WS 1,msTicks	
Parancssor Dinamikus para	incs lista
	-
4	- F
ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess COVERA	AGE COVTOFILE DEFINE

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.

47: LED_Initia	lize(); BL.W	LED Initialize (0x00001470)
48: LED On(0);		
O0x00001FD8 2000	MOVS	r0,#0x00
0x00001FDA F7FFFA4E	BL.W	LED_On (0x0000147A)
49: SysTick_Co	nfig(Syste	emCoreClock/1000);
50:		
Ox00001FDE 4822	LDR	r0,[pc,#136] ; @0x00002068
0x00001FE0 6800	LDR	r0,[r0,#0x00]
0x00001FE2 F44F727A	MOV	r2,#0x3E8
0x00001FE6 FBB0F1F2	UDIV	r1,r0,r2
1828: if ((ticks	- 1UL) >	SysTick LOAD RELOAD Msk)
1829: {		
0x00001FEA 1E48	SUBS	r0,r1,#1
0x00001FEC F1B07F80	CMP	r0,#0x1000000
⇒0x00001FF0 D300	BCC	0x00001FF4
1830 · return (1111.1 •	
] <		
D main.c* D startup e	fm32aa.s	core cm3.b
42 -		
43 int main(vo	1d) {	
44 - /********	********	* * * *

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.

D 🕅	Projects	EFM32	GG-STK3700	\Advanced\	Blinky.uvprojx	- µVisio	on			_						-	_ 0	x
<u>F</u> ile	<u>E</u> dit	<u>V</u> iew _	Project Fl <u>a</u>	ih <u>D</u> ebug	Peripherals	Tools	<u>S</u> VCS	<u>W</u> indow	<u>H</u> elp									
	🞽 🔒	1	X 🕩 🛍	90	← ⇒ 🆗	12.1	8 B.		//=// _R	2	slide_pos_last	-	🗟 🥐 🛛 🕰 🗸	۲	0 (ջ 🚓	-	2
٢	*	🧼 🕶		Target 1		. 🔊	📥 🔁	🔶 🔶	(1)									

- 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 törésponton 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.

Breakpoints	X
Current Breakpoints:	
☑ 00: (A read 0x20000000 len=4), 'msTicks',	
✓ 01: (E) 0x00001FDE, "\\Blinky\main.c\49",	
U2: (E) 0x00001FD8, "\\Blinky\main.c\48",	
	,
	Access
Expression:	∏ <u>R</u> ead ∏ <u>W</u> rite
Count: 1	Size:
Command:	
,	
Define Kill Selected Kill All Ch	asa Hala
	Heip

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.

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

Watch 1		
Name	Value	Туре
SystemCoreClock	14000000	unsigned int
msTicks	0x00001B3	int
🖶 🛠 SysTick	0xE000E010	pointer
🗸 CTRL	0x00010007	unsigned int
🐓 LOAD	0x000036AF	unsigned int
VAL	0x00003692	unsigned int
CALIB	0x400036B0	unsigned int
<pre>Enter expression></pre>		
Call Stack + Locals Watch 1	Memory 1	

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.

Call Stack + Locals		E
Name	Location/Value	Туре
CAPLESENSE_getSliderPosit	0x00000478	int f()
🖗 i	<not in="" scope=""></not>	auto - int
💮 🔗 minPos	<not in="" scope=""></not>	auto - int
💮 🔗 minVal	<not in="" scope=""></not>	auto - unsigned int
🗈 🔗 interpol	0x200010B4	auto - unsigned int[6]
🗈 🔗 channelPattern	0x200010A0	auto - unsigned int[5]
position	<not in="" scope=""></not>	auto - int
CapSenseScanCallback	0x00000F42	void f()
🗄 🔍 🔮 LESENSE_IRQHandler	0x00001662	void f()
📖 🖗 main	0x0000000	int f()
Call Stack + Locals Watch 1	Memory 1	

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.

Memory 1						
Address:	&msTic	ks				Â
0x20000	000:	00000000	00000000	00000000	00000001	
0x20000	010:	00000001	0000001	00000001	0000001	
0x20000	020:	00000001	0000001	00000001	00000001	
0x20000	030:	00000001	00000001	00000001	00000001	
0x20000	040:	00000001	00000001	00000001	00000000	
0x20000	050:	00000000	00000100	00000000	00000000	
0x20000	060:	02DC6C00	0008000	00D59F80	00000000	
0x20000	070:	00000000	00000000	00000000	00000000	
0x20000	080:	00000000	00000000	00000000	00000000	
0x20000	090:	00000000	00000000	00000000	00000000	
0x20000	0A0:	00000000	00000000	00000000	00000000	Ŧ
Call St	ack + L	ocals Watch	1 Memor	y 1		

- 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.





Az ULINKplus debuggerrel egyszerre optimalizálhatjuk szoftverünket fogyasztáskritikus alkalmazásokban, és készíthetünk automata teszt algoritmusokat az extra IO lábak szkriptelésével.

<u>TUDJON MEG TÖBBET ></u>