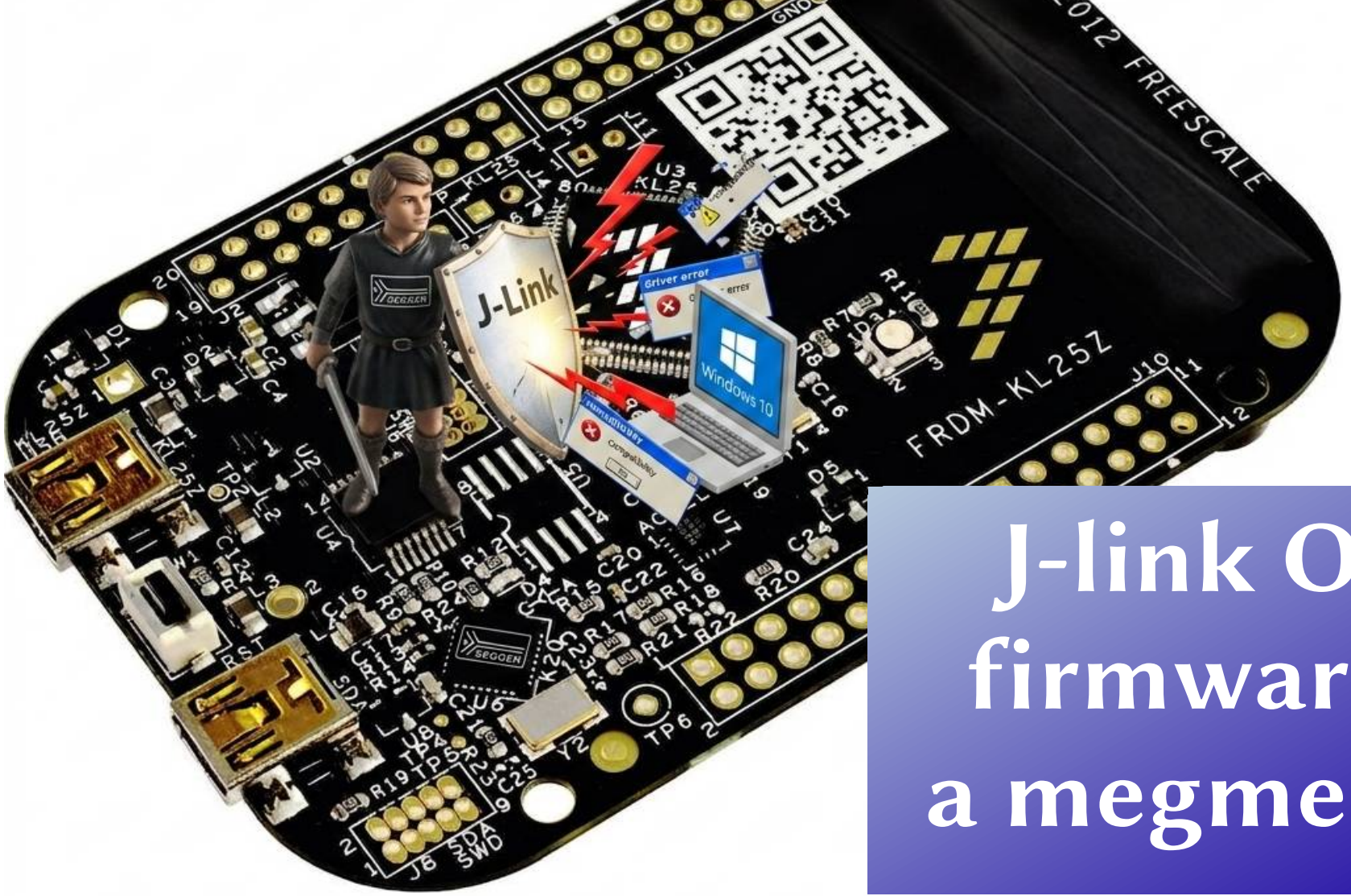


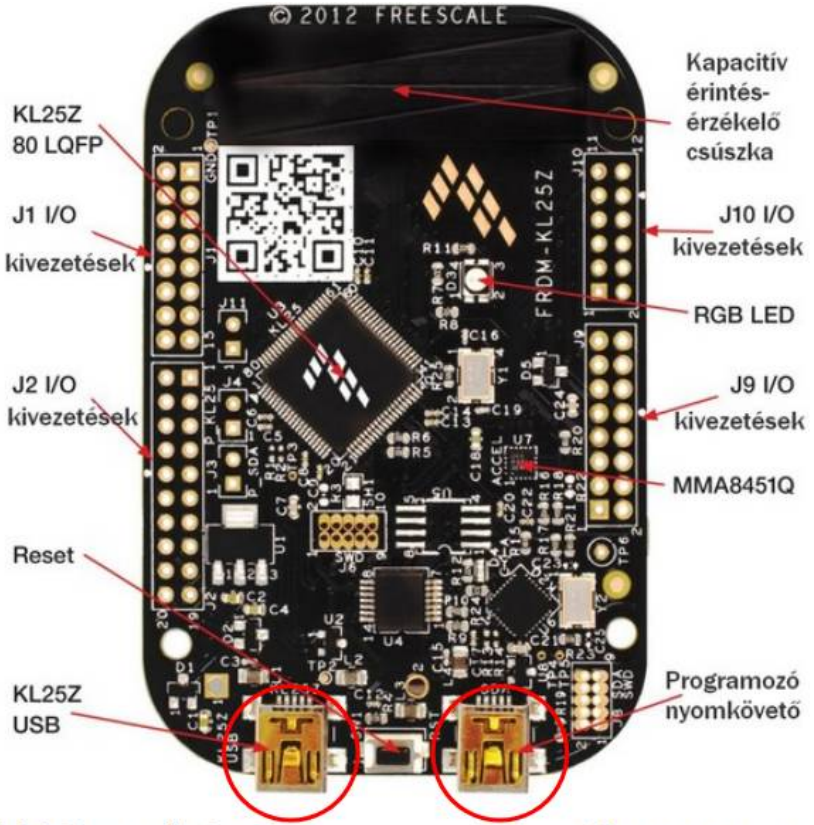
J-link OB firmware - a megmentő

Freescale FRDM-KL25Z kártya

- Freescale MKL25Z128VLK4 MCU
- Nagy teljesítményű ARM Cortex-M0+
- 48MHz, 16KB RAM, 128KB FLASH
- USB (Host/Device)
- SPI (2) / I2C (2) / UART (3)
- PWM (TPM)
- ADC (16 bit) / DAC (1x12bit)
- Érintésérzékelő
- GPIO (66)
- MMA8451Q - 3-tengelyű gyorsulásmérő
- Kapacitív érintésérzékelő csúszka

A kártya műszaki adatai:

- méret: 81mm x 53mm
- 5V USB vagy 4.5-9V külső tápellátás
- Beépített USB FLASH
- "fogd és vidd" programozás



USB eszköz
Közvetlen csatlakozás a mikrovezérlőhöz

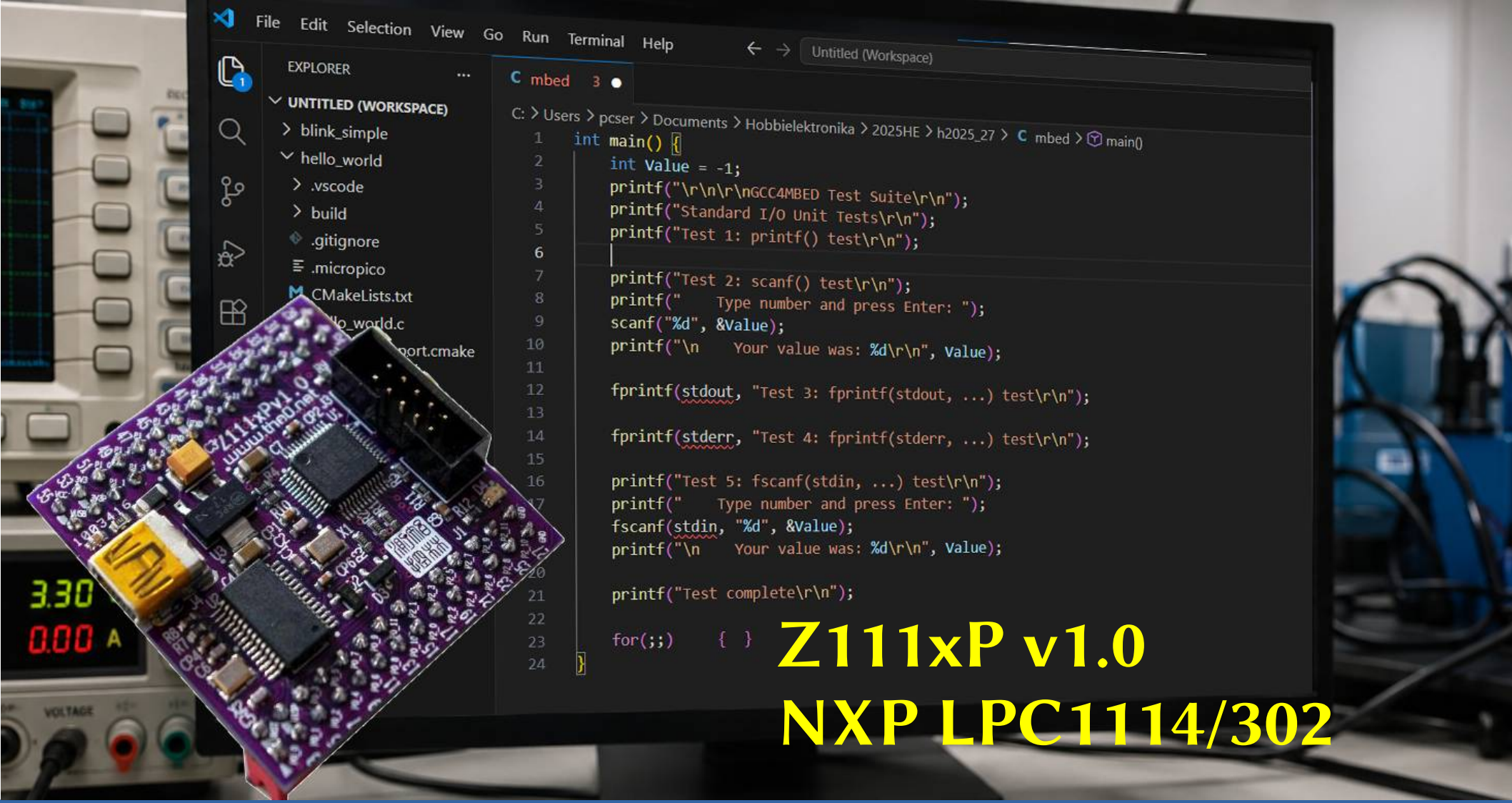
Programozó eszköz
Soros porton kommunikál a mikrovezérlővel (*Serial class*)

A FRDM-KL25Z kártya újraélesztése – 1. lépés

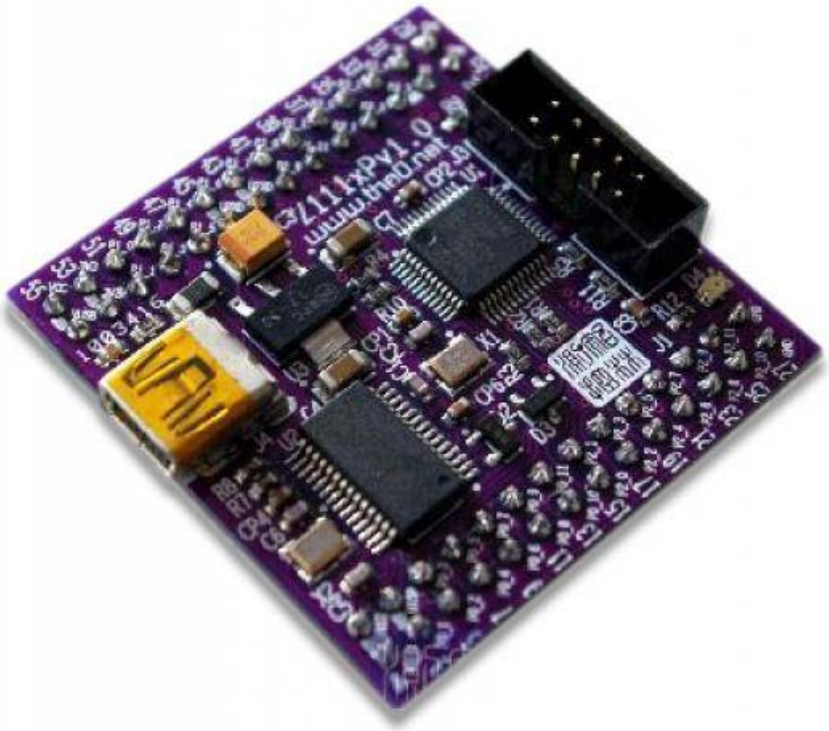
- Ismert probléma, hogy a Windows 10 „elrontja” az OpenSDA bootloadert, a programletöltés nem működik, a kártya használhatatlanná válik
- **Az újraélesztés menete (csak Windows 7 alatt működik stabilan):**
  - Az https://www.pemicro.com/opensda/ oldalról le kell tölteni az OpenSDA Firmware (MSD & Debug) csomagot (ehhez regisztrálni kell)
    A letöltött fájl: Pemicro_OpenSDA_Debug_MSD_Update_Apps_2023_12_12.zip
    Ebből csak a kétszeresen zip-pelt **BOOTUPDATEAPP_Pemicro_v111.SDA** kell
  - A https://os.mbed.com/handbook/Firmware-FRDM-KL25Z oldalról le kell tölteni a kártyához való mbed firmware-t (20 140 530_k20dx128_kl25z_if_opensda).
    A zip fájlból csomagoljuk ki a firware-t: **kl26z_opensda.s19**
  - Helyezzük a FRDM-KL25Z kártyát bootloader (programbetöltő) módba, majd a megjelenő BOOTLOADER meghajtóba másoljuk be a **BOOTUPDATEAPP_Pemicro_v111.SDA** és a **kl26z_opensda.s19** állományokat!
- **A fenti módon újtaélesztett kártyát csak Windows 7 alatt használható!**

Segger J-Link OB firmware – a megmentő

- A SEGGER olyan firmware-t kínál, amely az **NXP OpenSDA** platformon fut, azt **J-Link LITE**-tal kompatibilissé, valamint Windows 10 alatt is használhatóvá teszi
- **A telepítés lépései:**
  - Telepítsük a J-Link Software and Documentation Pack-et a PC-re. Ez felteszi a szükséges J-Link USB-drivert és a későbbi használathoz szükséges komponenseket
  - Ugyanazon a letöltési oldalon lejjebb görgetve a **J-Link OpenSDA - Board-Specific Firmwares** szekcióban keressük meg és töltsük le a **FRDM-KL25Z** kártyához tartozó **12_OpenSDA_FRDM-KL25Z.bin** firmware-t
  - Helyezzük a kártyát boot módba (a reset gomb lenyomott állapotában csatlakoztassuk), majd az új firmware-t drag-and-drop módszerrel másoljuk a megjelent meghajtóra. A másolás után áramtalanítsuk / újra csatlakoztassuk a kártyát. A rendszer ekkor már **J-Link** eszközként ismeri fel
- A **J-Link OB** támogatja az **SWD** programletöltést/nyomkövetést és a virtuális soros porti funkciót. Opcionálisan használható a drag-and-drop programozás is. A „mass storage” funkció be- és kikapcsolása a parancssori **Jlink.exe**-ben az **MSDenable** illetve **MSDdisable** parancsokkal történik


File Edit Selection View Go Run Terminal Help
Untitled (Workspace)
EXPLORER
UNTITLED (WORKSPACE)
blink_simple
hello_world
.vscode
build
.gitignore
.micropico
CMakeLists.txt
mbed 3
C: > Users > pcser > Documents > Hobbielektronika > 2025HE > h2025_27 > mbed > main()
int main() {
int Value = -1;
printf("\r\n\r\nGCC4MBED Test Suite\r\n");
printf("Standard I/O Unit Tests\r\n");
printf("Test 1: printf() test\r\n");
printf("Test 2: scanf() test\r\n");
printf(" Type number and press Enter: ");
scanf("%d", &Value);
printf("\n Your value was: %d\r\n", Value);
fprintf(stdout, "Test 3: fprintf(stdout, ...) test\r\n");
fprintf(stderr, "Test 4: fprintf(stderr, ...) test\r\n");
printf("Test 5: fscanf(stdin, ...) test\r\n");
printf(" Type number and press Enter: ");
fscanf(stdin, "%d", &Value);
printf("\n Your value was: %d\r\n", Value);
printf("Test complete\r\n");
for(;;) { }
}
Z111xP v1.0
NXP LPC1114/302

Z111xP v1.0 kártya (NXP LPC1114/302)



A kártya **ARM Cortex-M0** magú, 32 bites, NXP **LPC1114** mikrovezérlőt tartalmaz, melynek főbb jellemzői:

- 32KB Flash/8 KB RAM,
- max. 50 MHz CPU frekvencia
- 1 UART, 2 SPI, 1 I2C
- 2 db 32-bites időzítő/számláló
- 2 db 16-bites időzítő/számláló
- 8 csatornás 10-bites ADC
- Real-time óra
- Tápfeszültség/jelszint: 3,3V

A kártya a mikrovezérlőn kívül egy USB/UART átalakítót tartalmaz, amely kommunikációra és a programok letöltésre (a mikrovezérlőbe gyárilag betöltött bootloaderrel) használható. A panel egy 10 pólusó mini JTAG (2 mm-es osztású) csatlakozón keresztül szabványos JTAG programozó és hibavadász készülékhez is csatlakoztatható (pl. JLink).

A kártya elsősorban tanuláshoz, vagy szerényebb igényű alkalmazások fejlesztéséhez javasolt.

Ajánlott tankönyv: Joseph Yiu: The Definitive Guide to the ARM Cortex-M0

GCC4Mbed projekt: HelloWorld

main.cpp

```
// A beépített LED1 villogtatása

#include "mbed.h"

DigitalOut myled(LED1); // P1_5

int main()
{
    while(1)
    {
        myled = 1;
        wait(0.2);
        myled = 0;
        wait(0.2);
    }
}
```

Makefile

```
PROJECT          := HelloWorld
DEVICES          := LPC1114
GCC4MBED_DIR     := ../..
GCC4MBED_TYPE    := Release
MBED_OS_ENABLE   := 0
NEWLIB_NANO      := 1
NO_FLOAT_SCANF   := 1
NO_FLOAT_PRINTF  := 1

include $(GCC4MBED_DIR)/build/gcc4mbed.mk
```

Programletöltés:

Segger **Jflash.exe** programmal (ha van Jlink)

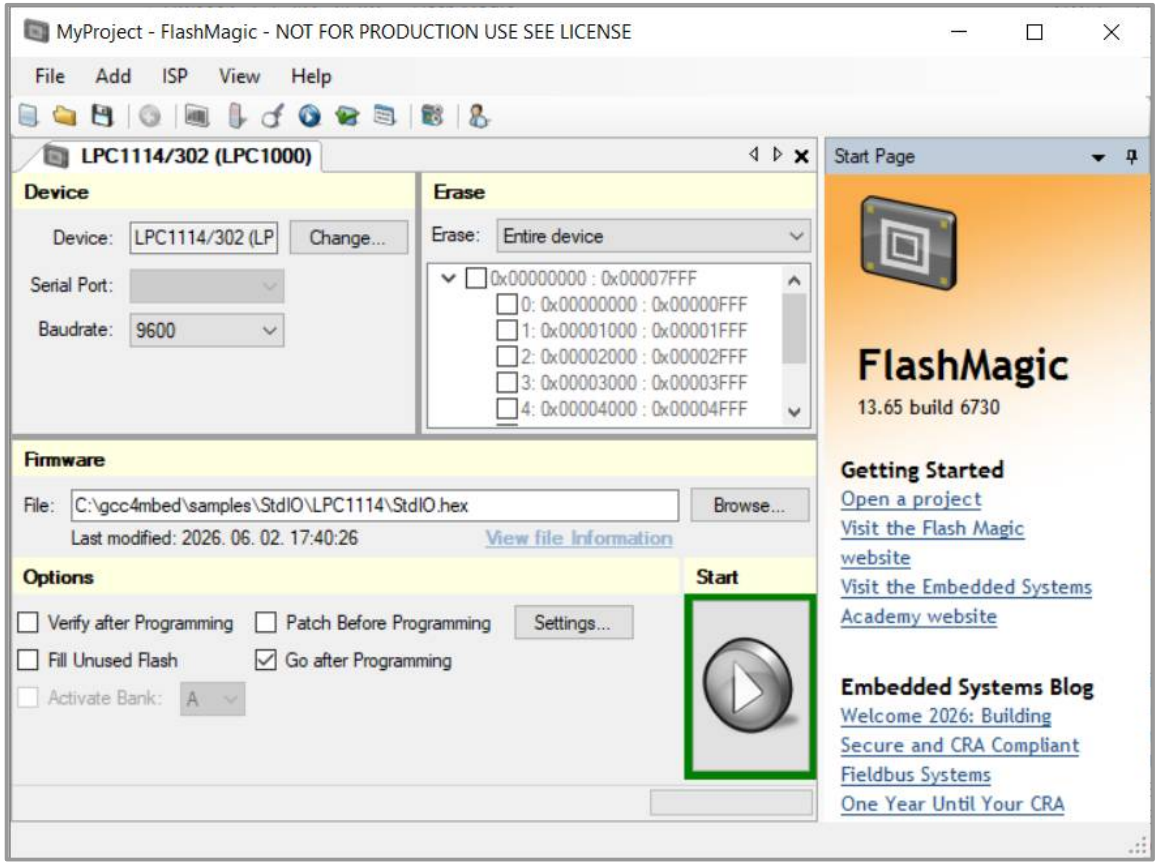
FlashMagic programmal (bootloader módban)

Programletöltés

A Flash Magic program az **LPC1114** mikrovezérlő gyárilag beépített ISP bootloadert használja, amely UART-on keresztül fogadja a programozási parancsokat

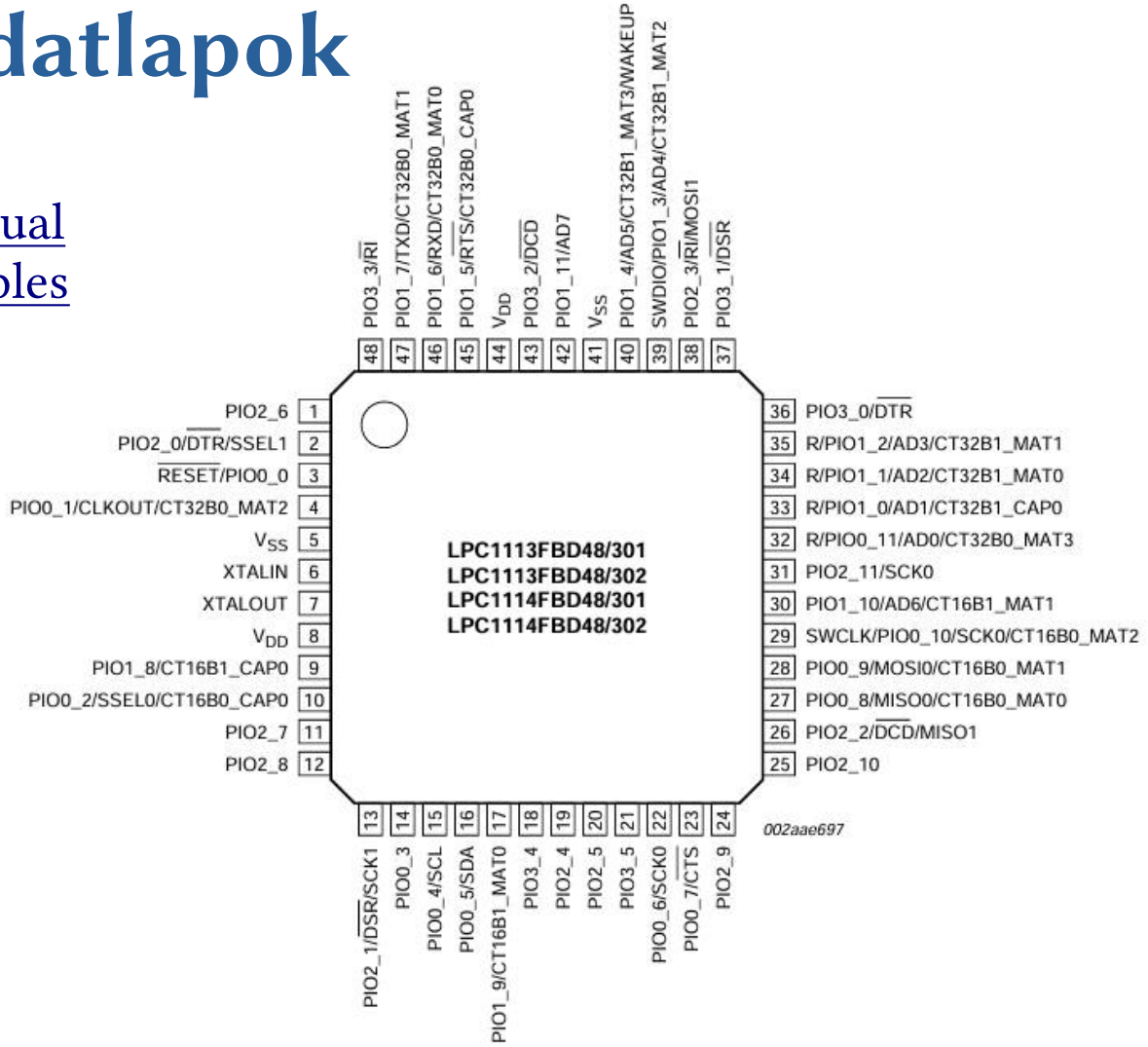
A **Z111xP v1.0** kártyán egy **PL2303** USB–soros átalakító biztosítja a PC és az MCU közti kommunikációt, és a hardver úgy van kialakítva, hogy a **PL2303** vezérelni tudja a mikrovezérlő RESET és ISP lábait is

Ennek köszönhetően a Flash Magic automatikusan bootloader módba tudja kapcsolniaz MCU-t, majd az RX/TX vonalakon keresztül rátölti a kiválasztott fájl tartalmát

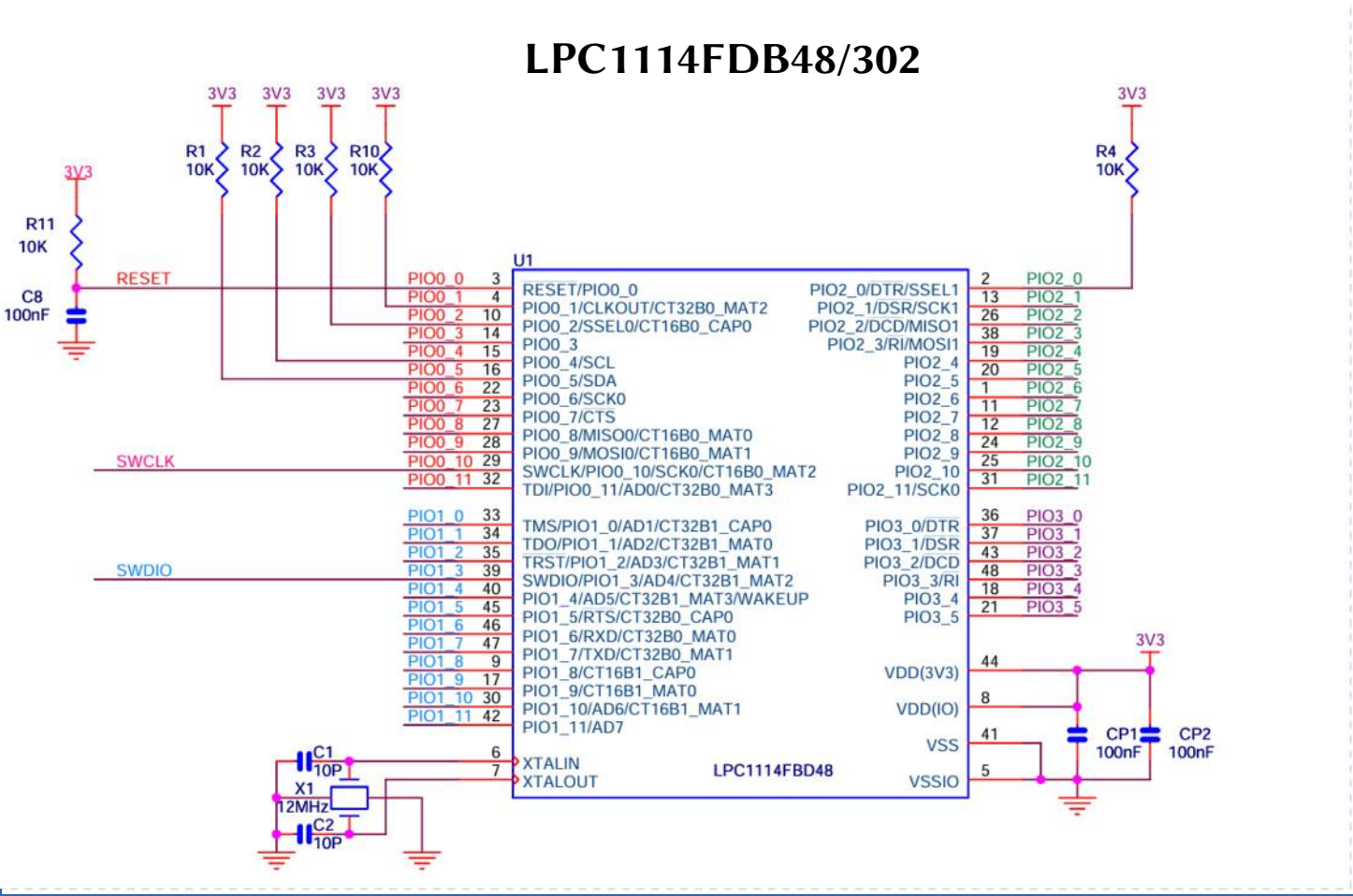


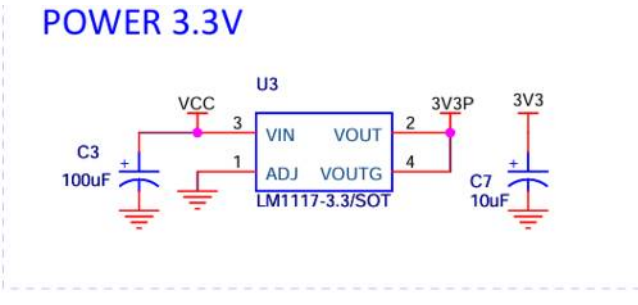
Adatlapok

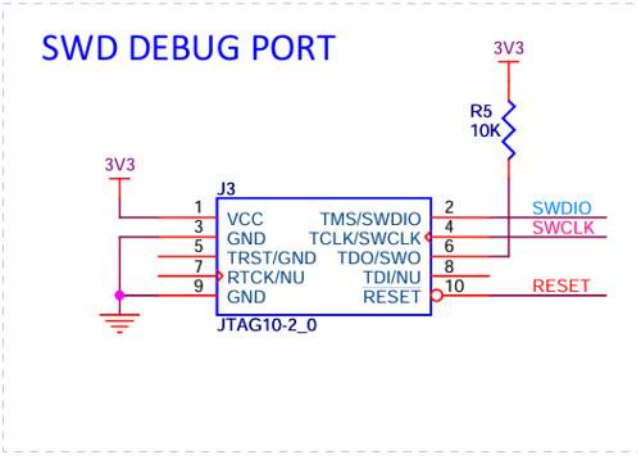
- NXP: LPC111x adatlap
- NXP: LPC111x/LPC11Cxx User manual
- NXP: LPCOpen Libraries and Examples
- Adam Green: gcc4mbed



Z111xP v1.0 kártya





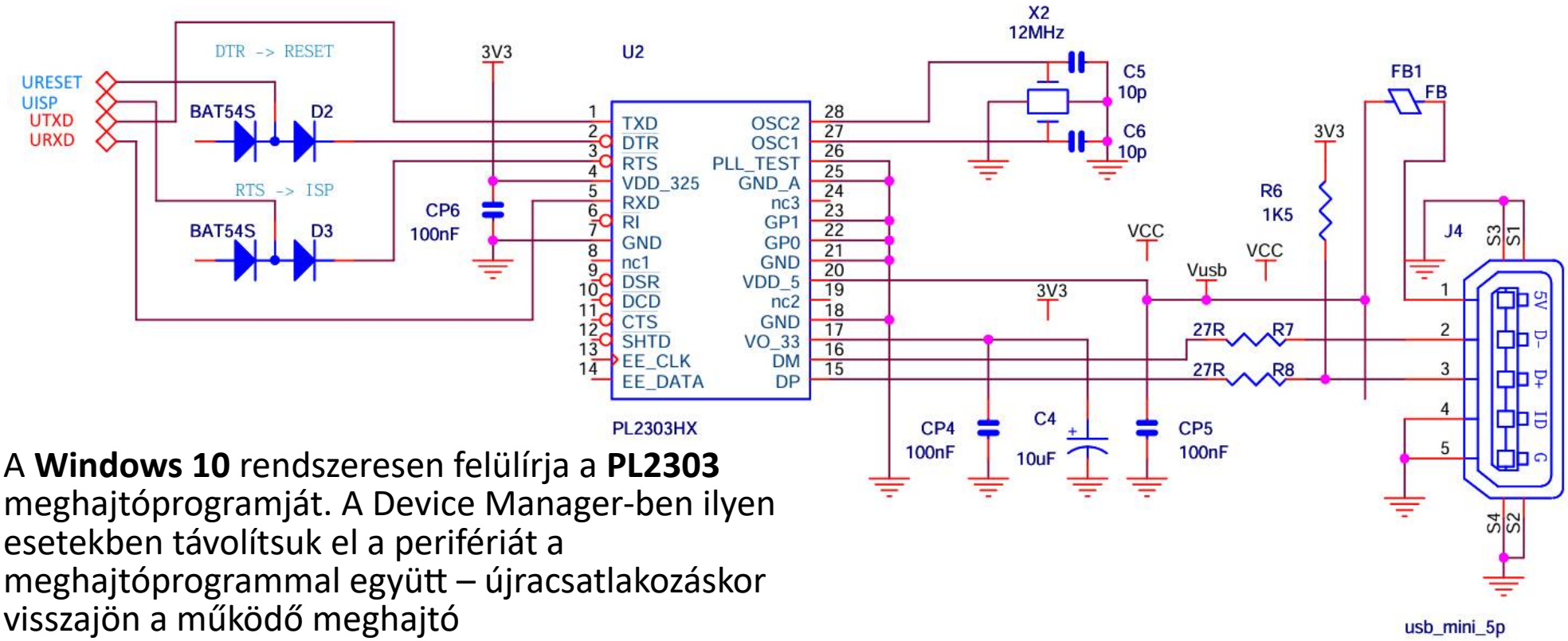


Title Z111xP

Size A4 | Document Number LPC1114

Z111xP v1.0 kártya

UART-USB átalakító és programletöltő

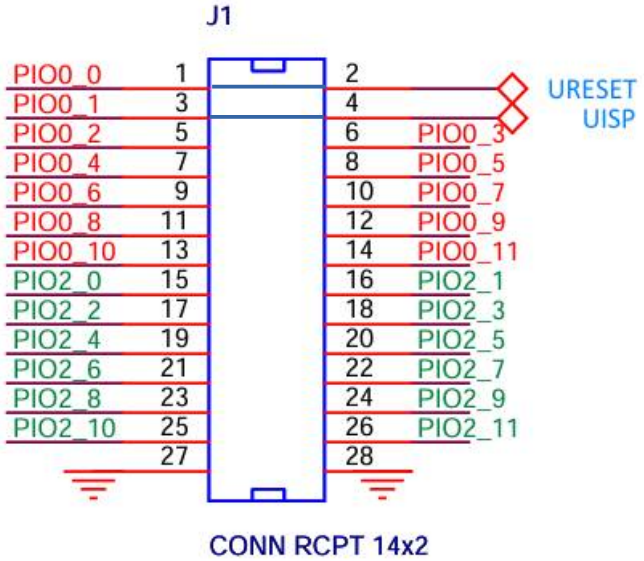


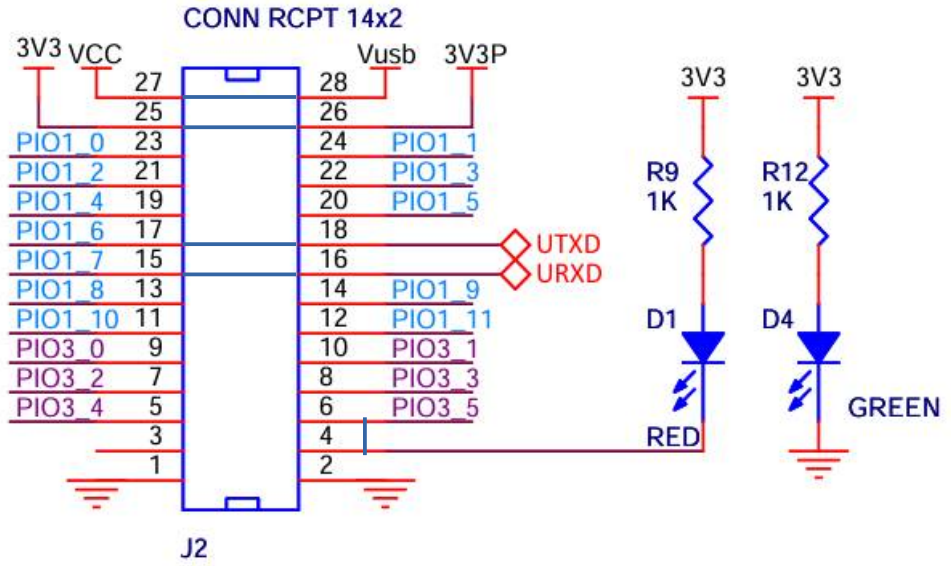
A **Windows 10** rendszeresen felülírja a **PL2303** meghajtóprogramját. A Device Manager-ben ilyen esetekben távolítsuk el a perifériát a meghajtóprogrammal együtt – újracsatlakozáskor visszajön a működő meghajtó

Z111xP v1.0 kártya

Kivezetések és átkötések

- **Tapasztalat:** A biztonságos indítás érdekében programletöltés után a **PIO0_0** és **PIO0_1** lábakról le kell venni az átkötést és a **PIO0_0** láb ideiglenes földre húzásával hardveresen resetelni kell a mikrovezérlőt

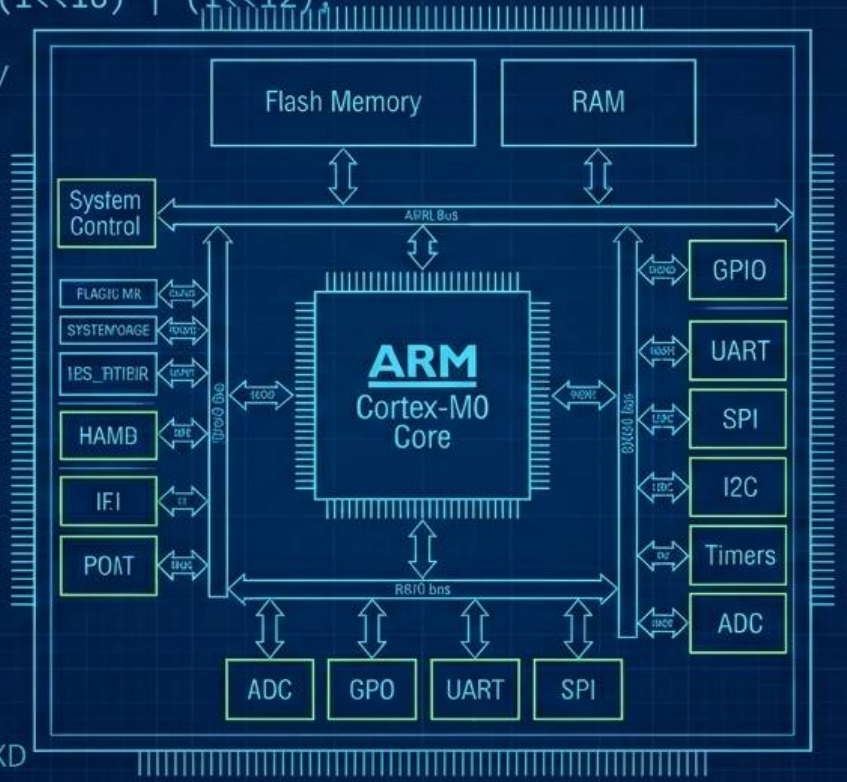




NXP LPC1114 BARE-METAL PROGRAMMING ARCHITECTURE (GCC + MAKE)

```
/* Initialize System Clock */
LPC_SYSCON->SYSAHBCLKCTRL |= (1<<16) | (1<<12);

/* Configure GPIO0 as Output */
LPC_GPIO0->DIR |= (1<<7);
LPC_GPIO0->DATA |= (1<<7);
```




```
/* UART pin settings */
LPC_IOCON->PIO1_6 &= ~0x07;
LPC_IOCON->PIO1_6 |= 0x01; //RXD

LPC_IOCON->PIO1_7 &= ~0x07;
LPC_IOCON->PIO1_7 |= 0x01; //TXD
```

Bare-Metal programozás a gcc4mbed infrastruktúrájának felhasználásával

Bare metal programozás a gcc4mbed infrastruktúrájával

- **Bare Metal megközelítés:** Közvetlen hardverkezelés operációs rendszer és szoftveres absztrakciós rétegek nélkül
- Az infrastruktúra lényege: Egy kész, offline fejlesztői környezet (gcc4mbed), amely a háttérben biztosítja a teljes fordítási láncot
- A környezet tartópillérei:
  - **GCC ARM (arm-none-eabi):** A professzionális keresztfordító rendszer
  - **GNU Make:** a fordítási és linkelési folyamatok Makefile-alapú automatizálása
  - **CMSIS Core:** A Cortex-M0 MCU szabványos, hardverközeli hozzáférési rétege
  - **LPC11xx eszközfájlok:** regiszterdefiníciós fejléc (LPC11xx.h) és a startup kód
- **A módszer előnyei:**
  - Nincs felesleges szoftver overhead (nincs elpazarolt Flash és RAM)
  - Determinisztikus futásidő és közvetlen megszakítás-kezelés
  - Teljes kontroll a mikrokontroller belső perifériái és órajelei felett

A könyvtárszerkezet kialakítása

```
c:\gcc4mbed\baremetal\
├── cmsis/
│   ├── core_cm0.h              (Cortex-M0 mag specifikus definíciók)
│   ├── core_cmFunc.h           (Mag belső függvényei)
│   └── core_cmInstr.h          (Speciális mag utasítások)
├── targets/
│   └── TARGET_LPC11XX/
│       ├── LPC11xx.h           (Gyári regiszter- és periféria definíciók)
│       ├── bitfields.h         (Regisztermaszkok és bitmezők)
│       ├── LPC1114.ld          (Linker szkript: Flash/RAM memória térkép)
│       ├── startup_LPC11xx.S   (Vektor tábla és hardveres reset kód)
│       ├── system_LPC11xx.c    (Rendszer- és órajel inicializálás)
│       └── system_LPC11xx.h
└── projects/
    ├── Blinky/
    │   ├── main.c              (Egyszerű LED villogtató forráskód)
    │   └── Makefile            (Lokális fordítási szabályok)
    ├── UART_demo/
    │   ├── main.c              (Soros porti „Hello world!” mintapélda)
    │   └── Makefile
    └── UART_test/
        ├── main.c              (Teszt kód az UART működésének ellenőrzéséhez)
        └── Makefile
```

Blinky/main.c

```c
#include "LPC11xx.h"
#include "bitfields.h"
#define LED1_MASK  (1 << 5)                                  // P3_5 maszkja (D1 LED)
static volatile uint32_t msTicks = 0;

void SysTick_Handler(void) {
    MsTicks++;                                               // SysTick megszakítás kiszolgálása
}

void delay_ms(uint32_t ms) {
    uint32_t currentTicks = msTicks;
    while ((msTicks - currentTicks) < ms);
}

int main(void) {
    SystemCoreClockUpdate();                                 // Rendszeróra változók frissítése
    SysTick_Config(SystemCoreClock / 1000);                  // 1 kHz-es SysTick időzítő
    LPC_SYSCON->SYSAHBCLKCTRL |= SYSAHBCLKCTRL_GPIO;         // GPIO blokk engedélyezése
    LPC_GPIO3->DIR |= LED1_MASK;                             // PIO3_5 kimenet (D1 katód)
    while (1) {
        LPC_GPIO3->MASKED_ACCESS[LED1_MASK] = 0;             // LED1 be
        delay_ms(250);
        LPC_GPIO3->MASKED_ACCESS[LED1_MASK] = LED1_MASK;     // LED1 ki
        delay_ms(250);
    }
}
```

Blinky/Makefile – 1. rész

```
TARGET    = blinky
BUILD_DIR = build

WORKSPACE_DIR = ../..
CMSIS_DIR     = $(WORKSPACE_DIR)/cmsis
TARGET_DIR    = $(WORKSPACE_DIR)/targets/TARGET_LPC11XX

SRCS = main.c $(TARGET_DIR)/system_LPC11xx.c $(TARGET_DIR)/startup_LPC11xx.S
OBJS = $(addprefix $(BUILD_DIR)/, $(addsuffix .o, $(basename $(notdir $(SRCS)))))

CC   = arm-none-eabi-gcc
SIZE = arm-none-eabi-size

CPU      = -mcpu=cortex-m0 -mthumb
INCLUDES = -I. -I$(CMSIS_DIR) -I$(TARGET_DIR)

CFLAGS  = $(CPU) -c -g -O3 -Wall -ffunction-sections -fdata-sections $(INCLUDES)
ASFLAGS = $(CPU) -c -g

LDSCRIPT = $(TARGET_DIR)/LPC1114.ld
LDFLAGS  = $(CPU) -nostartfiles -Wl,--gc-sections -T$(LDSCRIPT) --specs=nosys.specs
```

Blinky/Makefile – 2. rész

```
all: $(BUILD_DIR) $(BUILD_DIR)/$(TARGET).elf

$(BUILD_DIR):
	@if not exist $(BUILD_DIR) mkdir $(BUILD_DIR)

# Szabály a helyi C fájlokhoz (main.c)
$(BUILD_DIR)/%.o: %.c
	$(CC) $(CFLAGS) $< -o $@
# Szabály a távoli targets C fájlokhoz (system_LPC11xx.c)
$(BUILD_DIR)/%.o: $(TARGET_DIR)/%.c
	$(CC) $(CFLAGS) $< -o $@
# Szabály a távoli targets Assembly fájlokhoz (startup_LPC11xx.S)
$(BUILD_DIR)/%.o: $(TARGET_DIR)/%.S
	$(CC) $(ASFLAGS) $< -o $@

# ELF fájl linkelése és méretének kiíratása
$(BUILD_DIR)/$(TARGET).elf: $(OBJS)
	$(CC) $(LDFLAGS) $(OBJS) -o $@
	$(SIZE) $@
clean:
	@if exist $(BUILD_DIR) rmdir /s /q $(BUILD_DIR)

.PHONY: all clean
```

Problémák és megoldások

Problémák és megoldások az átállásnál

- **Az Mbed online fordító környezete** kényelmes volt, de nehéz és korlátozó szoftveres rétegeket hagyott maga után:
  - Az eredeti Linker Script (.ld) fenntart egy üres blokkot a RAM elején
    Megoldás: A címtartományokat vissza kell írni a nyers fizikai határokra
  - A startup kód alapértelmezetten az Mbed saját belső inicializáló függvényét (mbed_main) hívja a standard C main() helyett
    Megoldás: A startup fájlban át kell írni a ugrási parancsot szabványos main-re
- **Az NXP mikrovezérlők hardveres csapdái:**
  - **A 7. vektor (0x1C) ellenőrzése:** Indításkor a Boot ROM összeadja az első 8 vektort; az összegnek nullának kell lennie. Hibás checksum esetén a chip nem indítja el a programot, hanem azonnal ISP (letöltő) módba ugrik – tehát gondoskodnunk kell a helyes cheksum érték kiszámításáról és az 0x1C címen történő elhelyezéséről
  - **A CRP (Code Read Protection) csapdája:** A kódvédelem regiszterébe (0x2FC címtől kezdődő 4 bájt) tévedésből beírt rossz érték véglegesen lezárhatja a Flash-t (JTAG/ISP letiltás). Az LPC1114 esetén ez a kódterületre esik, tehát „ki kell kerülni”

LPC1114.ld releváns részletei

```
ENTRY(Reset_Handler)
MEMORY {    FLASH (rx)  : ORIGIN = 0x00000000, LENGTH = 32K
            RAM   (rwx) : ORIGIN = 0x10000000, LENGTH = 4K
}
PROVIDE(__valid_user_code_checksum = 0 - ( _vStackTop + (Reset_Handler + 1) + (NMI_Handler + 1) +
                                        (HardFault_Handler + 1) )
);
_vStackTop = ORIGIN(RAM) + LENGTH(RAM); /* A Stack pointer fixen a RAM legteteje */
SECTIONS {
    .text :
    {
        _text = .;
        KEEP(*(.isr_vector))
        *(.after_vectors*)              /* CRP elé kerülő dolgok */
        . = 0x000002FC;                 /* Feltöltés a CRP fix hardveres címéig */
        KEEP(*(.crp))
        *(.text*)                       /* Programkód és konstansok */
        *(.rodata*)
        . = ALIGN(4);
        _etext = .;
    } > FLASH
```

Cheksum számítás a 7. vektorhoz

LPC11xx.S releváns részlete

```
.syntax unified
    .arch armv6-m
    .weak    __valid_user_code_checksum
    .section .isr_vector,"a",%progbits
    .align 2
    .globl __isr_vector
__isr_vector:
    /* Cortex-M0 Core megszakítások */
    .long    _vStackTop
    .long    Reset_Handler
    .long    NMI_Handler
    .long    HardFault_Handler
    .long    0
    .long    0
    .long    0
    .long    __valid_user_code_checksum

    .long    0
    .long    0
    .long    0
    .long    SVC_Handler
     ...
```

.weak __valid_user_code_checksum

Gyenge (felülbírálható) hivatkozás. Azt mondja a fordítónak: ha a C kód vagy a Linker Script nem definiálja ezt a szimbólumot, ne dobjon hibát, hanem vegye 0-nak. Ha viszont a Linker Script kiszámolja (ahogy a mi **PROVIDE** parancsunk teszi), akkor az a valós érték lép a helyére

.section .isr_vector, "a", %progbits

Létrehoz egy egyedi memóriaszekciót a megszakítási táblának az alábbi tulajdonságokkal:

.isr_vector: A szekció neve (a linker a Flash elejére teszi)

"a" (allocatable): Helyet kell neki foglalni a memóriában (bekerül a végleges binárisba)

%progbits: A szekció tényleges programadatot (kódot vagy fix konstansokat) tartalmaz, nem üres helyőrző

Blinky (C++ projekt)

- Ez a mintaprojekt azt szemlélteti, hogy az LPC1114 mikrovezérlőhöz készített bare metal környezetben akár C++ projekteket is készíthetünk:
- **Globális statikus példányosítás:** A **Led led(LPC_GPIO3, 5);** objektumpéldány globális térben jön létre, így a memóriafoglalás már fordítási időben eldől, és a mikrokontroller indulásakor azonnal lefut a hardvert inicializáló konstruktor
- **Hardver-absztrakció és biztonság:** A **Led** osztály belül az **LPC1114** maszkolt portelérését (MASKED_ACCESS) használja. Ez hardveres szinten biztosítja az atomi (írási/olvasási konfliktusoktól mentes) bitmanipulációt az **on(), off()** és **toggle()** metódusokban
- **A StateProxy és az operátor kiterjesztés:** A beágyazott segédosztály transzparens átjárót képez a hardver és a szoftver között. Az = és a **bool()** operátorok felülbírálásával eléri, hogy a fizikai LED-re ne csak függvényként, hanem egyszerű logikai változóként is hivatkozhassunk
- **A működés tesztelése:** A **main** függvény a SysTick időzítőre támaszkodva, 250 ms-os lépésekben, szekvenciálisan teszteli az osztály összes képességét: a klasszikus metódusokat (**on/off, toggle**), a proxy-n keresztüli közvetlen értékadást (**led.state = true**), valamint a fizikai hardver állapotának intelligens visszaolvasását (**if (led.state == false)**)

Blink4 (C++ projekt)

- Ez a C++ mintapélda, ami a LPC1114 mikrokontrolleren két LED-et villogtat, a **dinamikus memóriakezelést** szemlélteti:
- **Dinamikus helyfoglalás**: A main függvényben a **Led** objektumok a **new** operátorral, dinamikusan jönnek létre. Ez a megoldás a háttérben a **cpp_runtime.h**-ban felüldefiniált, memóriatakarékos **new** operátorunkat használja, így elkerülhető a szabványos C++ könyvtár mikrokontrolleres környezetben pazarló heap-kezelése.
- **Objektumorientált hardver-absztrakció:** A Led osztály konstruktora a dinamikus példányosításkor automatikusan engedélyezi a GPIO órajelet és beállítja a kimeneti irányt. A hardverkezelés az LPC1114 maszkolt portelérését (MASKED_ACCESS) használja az atomi írás/olvasás érdekében.
- **StateProxy segédosztály:** az operátorok felülbírálásával lehetővé teszi, hogy a LED-ek állapotát egyszerű változóként (pl. led.state = true;) lehessen írni és olvasni.
- **Időzítés és működés:** A SysTick megszakítás által hajtott **delay_ms()** segítségével a végtelen ciklus a dinamikusan lefoglalt led1 és led2 objektumokat (mutatókon keresztül elérve: **->on(), ->off()**) 500 ms-os ütemben, ellenütemben villogtatja