nucleo64_stm32f411re

Operációs rendszer: **Ubuntu 16.04 LTS**
Fejlesztőkörnyezet: **Eclipse Mars.2 (4.5.2)**
Grafikus tervezőprogram: **STM32CubeMX 4.19.0 verzió**
Eszközök helye: **~/STM32Toolchain/**
Github repository: **nucleo64_stm32f411re**

Tartalomjegyzék

Tartalomjegyzék 1

Alapismeretek 3
C programozás 3
Számrendszerek C-ben 3
Változó kiírása adott számrendszerben 3
Értékadás különböző számrendszerekben 3
Bitműveletek 3
Bitműveletek alkalmazása 4
Bit set 4
Bit reset 5
Bit check 5

HAL library 7
Debug információk a Console-on 7
stm32f4xx_hal_gpio.h 7
GPIO_InitTypeDef 7
GPIO_PinState 7
HAL_GPIO_WritePin 8
Bit set/reset register 8

Órajel 9
HSE oszcillátor 9

Fejlesztőkörnyezet felépítése 10
GNU ARM Eclipse plugin 10
Használat 10
Blinky (blink a led) 10
GCC ARM tool-chain telepítése 11
ST Link driver az STM32Nucleo board-hoz 11

Firmware frissítése 11
STM32CubeMX 11
Telepítés előkészítése 12
Telepítés 12
Parancsikon létrehozása 12
Használat 13
Projekt generálása CubeMX-szel 13
STLink tool 14
Telepítés előkészülete 14
Telepítés 14
HEX fájl írása a board-ra: 14
STM32Cube-HAL importálása Eclipse-es projektbe 14
OpenOCD telepítése és konfigurálása 15
Telepítés 15
Ellenőrzés 16
Elérési útvonal beállítása 16
Fejlesztés 16
Teszt projekt: blinky 16
Új teszt projekt létrehozása Eclipse-ben 16
Teszt projekt kódrészlete 17
Teszt project módosítása a teszthez 17
Teszt project debuggolása 18
Teszt projekt: gombnyomásra LED villogtatás 20
Projekt elnevezése 22
Rendszerórajel beállítása 22
Gomb beállítása 23
LED beállítása 24
Projekt mentése 24
Kód generálása 24
Kód importálása Eclipse IDE-be 25
Kód megismerése 25
Kódolás 27
Tesztelés 27
Github repository 28

Alapismeretek

C programozás

Számrendszerek C-ben

Változó kiírása adott számrendszerben

```
int num = 29;
printf("Decimal: %d, octal: %o, hexadecimal: %x or %X\n", num, num, num, num);
```

- Egy változó értékét különböző számrendszerekben könnyedén ki tudjuk íratni:
    - %d - decimális formában
    - %o - oktális (8-as számrendszer) formában
    - %x - hexadecimális (16-os számrendszer) formában, a 9-nél nagyobb hexa számjegyek kisbetűvel pl d
    - %X - szintén hexa formátum, a 9-nél nagyobb hexa számjegyek nagybetűvel pl D

Értékadás különböző számrendszerekben

```
int num_bin = 0b11101; // decimal: 23
int num_oct = 035; // decimal: 23
int num_dec = 29; // decimal: the same
int num_hex = 0x1d; // or 0x1D, decimal: 23
printf("bin: %d, oct: %d, dec: %d, hex: %d", num_bin, num_oct, num_dec, num_hex);
```

- bináris szám: a **0b** előtag után tudjuk megadni a számot pl 0b0111 (decimális: 7)
- oktális szám: a **0** előtag után adjuk meg az oktális számjegyeket pl 012 (decimális 10)
- decimális szám: előtag nélkül, decimális számjegyekkel adható meg
- hexadecimális szám: **0x** előtag után kell megadni a hexa számjegyeket (mindegy, hogy kis vagy nagybetűvel írjuk a 9-nél nagyobb hexa számjegyeket) pl 0x14 (decimális 20)

Bitműveletek

```
int a = 0b10010; // decimal: 18
int b = 0b00011; // decimal: 3
printf("a = %d, b = %d\n", a, b); // a = 18, b = 3
int result = (a | b); // binary: 10011
printf("result = a | b: %d\n", result); // result = a | b: 19
result = (a & b); // binary: 00010
printf("result = a & b: %d\n", result); // "result = a & b: 2
result = (a ^ b); // binary: 10001
printf("result = a ^ b: %d\n", result); // result = a ^ b: 17
result = 0xFFFFFFF8; // 0b11111111111111111111111111111000
result = ~result; // 0b00000000000000000000000000000111 = 7
printf("result = ~result: %d\n", result); // result = ~result: 7
```

Bináris VAGY operátor: |

- VAGY kapcsolat igazságtáblája:

A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

Bináris ÉS operátor: &

- ÉS kapcsolat igazságtáblája:

A	B	A · B
0	0	0
0	1	0
1	0	0
1	1	1

Bináris XOR operátor: ^

- XOR (kizáró vagy) kapcsolat igazságtáblája:

A	B	A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

- Kizáró vagy kapcsolat visszavezetése ÉS illetve VAGY kapcsolatra: $A \oplus B = \overline{A}B + A\overline{B}$

Bitenkénti negálás: ~

- például 1000 (8) → 0111 (7)

Bitműveletek alkalmazása

Bit set

- Egy adott bit 1-be állítása: VÁLTOZÓ |= (1 << HELYIÉRTÉK)

```
int num = 0b0010; // decimal: 2
printf("%d\n", num); // 2
num |= (1 << 3); // set 4th bit → decimal: 10
printf("%d\n", num); // 10
```

- num |= (1 << 3) módszerrel a 4. bitet egybe lehet állítani.
- Ha az 1-et, mint számot (aminél ugyebár az LSB 1, a többi bit 0) shifteljük megfelelő számúszor balra, akkor csak azon a helyiértéken lesz 1, ahol be akarjuk állítani a bitet. Ezt VAGY kapcsolatba hozzuk a módosítandó számmal, akkor csak azon a helyiértéken lesz változás, ahova shifteltük az egyet. Így a többi helyiérték nem változik (csupa 0 van mindenhol máshol), a szükséges bit pedig be lett állítva

Bit reset

- Egy adott bit 0-ba állítása:

```
int num = 0b0110; // decimal: 6
printf("%d\n", num); // 6
num &= ~(1 << 1); // reset 2nd bit → decimal: 4
printf("%d\n", num); // 4
```

- num &= ~(1 << 1) módszer lényege, hogy az 1-et elshifteljük arra a helyiértékre, amit 0-ba állítanánk. Jelen esetben a 2. helyiértéket (2^1) szeretnénk 0-ba állítani. Ha az 1-et elshifteljük eggyel balra, akkor egy olyan számot kapunk, ahova a törlendő bit 1-es, az összes többi 0. Ezt negáljuk meg, így csak azon a helyiértéken lesz 0, amit törölni szeretnénk, a többi bit 1. Ezt ÉS kapcsolatba hozzuk a változtatandó számunkkal, így a kívánt helyiérték 0 lesz (0 ÉS BIT = 0), a többi pedig nem változik, hiszen 1 ÉS BIT = BIT.

Bit check

- A feladat a következő: meg szeretnénk tudni, hogy egy adott bit 0 vagy 1.
- Jelen példánkban a vizsgálandó bit a 4. bit (2^3 helyiérték).
- Kód:

```
int num = 0b1100; // decimal: 12
if(num & (1<<3))
{
   // code if bit is set
   printf("bit is set\n");
}
else
{
   // code if bit is reset
   printf("bit is reset\n");
}
```

- A megvalósítás a következő:
    - Tudjuk, hogy melyik bitet szeretnénk vizsgálni, az 1-et elshifteljük balra annyival, hogy a megfelelő helyiértéken legyen (jelen esetben 3-mal (1 << 3).
    - A vizsgálandó szám és az előbb shiftelt 1-es ÉS kapcsolata csak akkor lesz logikai igaz, ha a vizsgálandó szám megfelelő helyiértékén 1-es szerepel.
- Ha szükségünk van a vizsgált bit értékére, akkor érdemes ezt elmenteni egy változóba, például az alábbi módon:

```
int num = 0b1100; // decimal: 12
```

```c
int value = (num & (1<<3));
if(value)
{
    // code if bit is set
    printf("bit is set\n");
}
else
{
    // code if bit is not set
    printf("bit is not set\n");
}
```

- így a *value* változónk tartalmazza a vizsgált bit értékét.

HAL library

Debug információk a Console-on

A számítógépen keresztüli debuggolás során az egyik leghasznosabb funkció a Console-ra való kiírás. Ennek segítségével nyomon követhetjük például szenzoraink értékét, változóink állapotát stb.
Erre szolgál a trace_puts(const char *s) függvény.
Például a

```
trace_puts("button is pressed, LED is ON");
```

kód kiírja a Console-ra, amikor egy push button hatására éppen bekapcsoljuk a LED világítást.

stm32f4xx_hal_gpio.h

A GPIO használatához szükséges struktúrákat és függvényeket tartalmazza.

GPIO_InitTypeDef

Egy GPIO pin használatához be kell állítanunk, hogy az adott pin fizikailag hol helyezkedik el és milyen működést várunk tőle. Ezt a GPIO_InitTypeDef struktúrában tudjuk megadni.
GPIO_InitTypeDef struktúra:

Adattag	Jelentése
uint32_t Pin	fizikailag melyik pin a boardon
uint32_t Mode	milyen módban akarjuk használni ezt a pint
uint32_t Pull	pull-up vagy pull-down szeretnénk, hogy viselkedjen
uint32_t Speed	milyen sebességgel működjön
uint32_t Alternate	perifériát is kapcsolhatunk hozzá ezzel az opcióval

GPIO_PinState

Digitális GPIO esetén egy pin értéke 0 vagy 1. Ezt sokkal szemléletesebben is jelölhetjük, erre szolgál a GPIO_PinState enumeráció:

```
typedef enum
{
  GPIO_PIN_RESET = 0,
  GPIO_PIN_SET
}GPIO_PinState;
```

Tehát a logikai hamis (0) értékhez a GPIO_PIN_RESET jelölést, a logikai igaz (1) értékhez a GPIO_PIN_SET értéket használjuk.

HAL_GPIO_WritePin

Ezzel a függvénnyel tudjuk egy adott GPIO pin értékét 1-re vagy 0-ra állítani. Az 1 és 0 szemléletes jelölésére szolgál a GPIO_PinState enum.
Például

```
HAL_GPIO_WritePin(led_GPIO_Port, led_Pin, GPIO_PIN_SET);
```

függvényhívással a LED-ünkhöz tartozó pin értékét 1-be állítjuk, vagyis bekapcsoljuk a LED-et.

Bit set/reset register

Minden porthoz van egy bit set/reset regiszter, amivel a pineket egyesével, vagy az összeset kényelmesen, egy lépésben tudjuk állítani. A GPIOx struktúra tartalmazza ezt a regisztert, amire GPIOx->BSRR-ként tudunk hivatkozni.
Ez egy 32 bites regiszter, aminek az alsó 16 bitje a set-elni kívánt (1-esbe állítandó) pineket jelenti, a felső 16 bit a reset-elni (0-ba állítani) kívánt pineket tartalmazza.
Ha például be szeretnénk kapcsolni egy LED-et, ami a PA5-ös pinhez tartozik, akkor a GPIOA-nak a regiszterébe kell írnunk:

```
GPIOA->BSRR = (1 << 5);
```

Ha egy port minden pinjét set-elni akarjuk, akkor ilyen kényelmesen megtehető egyetlen értékadással:

```
GPIOA->BSRR = 0xFFFF;
```

Órajel

A rendszer órajelét a SYSCLK jelölést jelenti. Ennek forrása 3 különböző dolog lehet:

1. HSI (High Speed Internal Oscillator): 16 MHz
    1.1. vagyis nagysebességű belső oszcillátor biztosítja
2. HSE (High Speed External Oscillator): 4 - 26 MHz
    2.1. vagyis nagysebességű külső oszcillátor biztosítja
3. Main phase-locked loop (PLL):
    3.1. PLL: phased-locked loop → erről bővebben például ezen a linken lehet olvasni.
    3.2. bemeneti órajele 1 vagy 2 MHz kell, hogy legyen

Minden periféria órajele a rendszer órajeléből származik (vannak kivételek: USB OTG FS, RNG, SDIO, I2S.

HSE oszcillátor

A HSE a High Speed External Oscillator rövidítése, vagyis egy külső nagysebességű oszcillátor. Erre akkor van szükség, amikor nem érhető el belső órajel, hanem egy külső forrásból (külső oszcillátorból) érkezik a rendszerünk órajele. Például egy kristályoszcillátor van a board-ra forrasztva.
Ha HSE oszcillátort használunk, akkor azt a következő helyen kell beállítani: stm32f4xx.h.
Ebben a fájlban a HSE_VALUE változó értéke legyen a megfelelő órajelünk értéke.

Fejlesztőkörnyezet felépítése

GNU ARM Eclipse plugin

- verzió 3.2.1
- telepítés menete:
    - Help ---> Install New Software… ---> Add…
        - Name: GNU ARM Eclipse Plug-ins
        - Location: http://gnuarmeclipse.sourceforge.net/updates
    - OK
    - Next
    - útmutató követése
    - Eclipse újraindítása
- GNU ARM Eclipse plugin telepítése közben hiba:
    - received fatal alert: handshake_failure
    - Hiba orvoslása: https://gnuarmeclipse.github.io/blog/2017/01/29/plugins-install-issue

Használat

- Eclipse megnyitása
- Új projekt létrehozása: File ---> New ---> C Project
    - Project name: test_project
    - Location: ~/home/sebokb/workspace/test_project
    - Project type: STM32F4xx C/C++ project
    - Toolchains: Cross ARM GCC
    - Next
    - Chip family: STM32FX411xE
    - Flash size (kB): 512
    - External clock (Hz): 100000000
    - Content: Blinky (blink a led)
    - Next
    - Next
    - Next
    - Toolchain name: GNU Tools for ARM Embedded Processors (arm-none-eabi-gcc)
    - Toolchain path: /home/sebokb/STM32Toolchain/gnu-arm/bin
    - Finish

Blinky (blink a led)
include/BlinkLed.h:

```
#if defined(BOARD_OLIMEX_STM32_E407)
// STM32-E407 definitions (the GREEN led, C13, active low)
// Port numbers: 0=A, 1=B, 2=C, 3=D, 4=E, 5=F, 6=G, ...
```

```
#define BLINK_PORT_NUMBER         (2)
#define BLINK_PIN_NUMBER          (13)
#define BLINK_ACTIVE_LOW          (1)
```

src/_initilize_hardware.c:

```
SystemClock_Config();
```

GCC ARM tool-chain telepítése

- telepítéshez szükséges kliens: https://developer.arm.com/open-source/gnu-toolchain/gnu-rm
    - Linux 64-bit kliens: https://developer.arm.com/-/media/Files/downloads/gnu-rm/6-2016q4/gcc-arm-none-eabi-6_2-2016q4-20161216-linux.tar.bz2?product=GNU%20ARM%20Embedded%20Toolchain,64-bit,,Linux,6-2016-q4-major
- kicsomagolás helye: ~/STM32Toolchain/gnu-arm
- kicsomagolás és beállítás:
    - *tar xjf gcc-arm-none-eabi-6_2-2016q4-20161216-linux.tar.bz2*
    - *mv gcc-arm-none-eabi-6_2-2016q4/ gnu-arm*
    - Szükséges package:
        - libncurses: *sudo apt-get install libncurses5-dev*

ST Link driver az STM32Nucleo board-hoz

- Driver letöltése: http://www.st.com/content/st_com/en/products/embedded-software/development-tool-software/stsw-link007.html
- Get Software
- Bejelentkezés
- Letöltés
- Szükséges package:
    - libusb-1.0: *sudo apt-get install libusb-1.0*
- Kicsomagolás helye: ~/STM32Toolchain/stlink/

Firmware frissítése

1. *sudo java -jar STLinkUpgrade.jar*
2. Refresh device list
3. Open in update mode
4. Upgrade (ha szükséges)
5. Upgrade successful.

STM32CubeMX

A fejlesztés során ez az egyik leghasznosabb alkalmazás, ami segíteni fog minket. A projekt során szükséges elemek, perifériák beállítása egy kényelmes grafikus felületen megoldható, így sok doksi olvasást és kódolást megspórolhatunk.
A projekt keretein belül könnyedén el tudjuk menteni a jelenlegi beállításokat, amiket később bármikor visszaállíthatunk, szóval egy folyamatosan változó, fejlődő projektet is karban tudunk majd tartani, módosítani.
Telepítés folyamata:

- Kliens letöltése: http://www.st.com/content/st_com/en/products/development-tools/software-development-tools/stm32-software-development-tools/stm32-configurators-and-code-generators/stsw-stm32095.html
- Get Software

Telepítés előkészítése

- Futtatási jog: *sudo chmod +x SetupSTM32CubeMX-4.19.0.linux*
- 32 bites program futtatása 64 bites laptopon:
    - sudo dpkg --add-architecture i386
    - sudo apt-get update
    - sudo apt-get install libc6:i386 libncurses5:i386 libstdc++6:i386

Telepítés

1. *sudo ./SetupSTM32CubeMX-4.19.0.linux*
2. útmutató követése

vagy

1. auto install szkript: ~/STM32Toolchain/stm32cubemx/auto_install.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<AutomatedInstallation langpack="eng">
<com.st.microxplorer.install.MXHTMLHelloPanel id="readme"/>
<com.st.microxplorer.install.MXLicensePanel id="licence.panel"/>
<com.st.microxplorer.install.MXTargetPanel id="target.panel">
<installpath>/usr/local/STMicroelectronics/STM32Cube/STM32CubeMX</installpath>
</com.st.microxplorer.install.MXTargetPanel>
<com.st.microxplorer.install.MXShortcutPanel id="shortcut.panel"/>
<com.st.microxplorer.install.MXInstallPanel id="install.panel"/>
<com.st.microxplorer.install.MXFinishPanel id="finish.panel"/>
</AutomatedInstallation>
```

Parancsikon létrehozása

A parancsikon létrehozására szükségünk van a dconf-editorra, melyet a következő paranccsal telepíthetünk:

```
sudo apt install dconf-editor
```

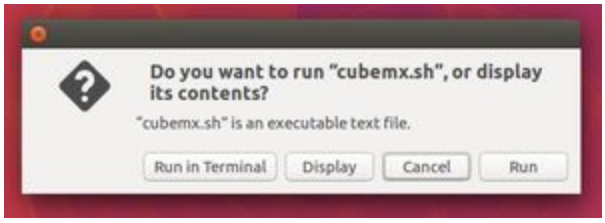
A parancsikon szerepét egy shell script formájában szeretném megvalósítani.
Hozzunk létre egy shell scriptet például az asztalon (*~/Desktop*), hogy kényelmesen elérhessük.
A script:

```
cd /usr/local/STMicroelectronics/STM32Cube/STM32CubeMX
./STM32CubeMX
```

cubemx.sh névre mentsük el a scriptet.
Tegyük futtathatóvá a scriptet a következő paranccsal:

```
chmod u+x cubemx.sh
```

A kényelmes használat miatt csináljuk meg, hogy dupla kattintásra is lefusson a script.
Ehhez nyissuk meg a dconf-editort, például a terminálba való begépelésével ($ dconf-editor).
Itt keressük meg a következőt: org > gnome > nautilus > preferences.
Itt az executable-text-activation opciót állítsuk ask-ra, ennek jelentése: ha duplán kattintunk gegy shell scriptre, akkor felugri egy ablak, hogy jelenítse meg a tartalmát / futtassa le a terminálban / futtassa le. Nekünk ezek közü az utolsó kell.
Bezárhatjuk a dconf-editort, végeztünk.
A “parancsikon” használata: dupla kattintás a scriptre, majd Run:



Használat

- kliens indítása:
  - *cd /usr/local/STMicroelectronics/STM32Cube/STM32CubeMX*
  - *java -jar STM32CubeMX*
- New Project
- Board Selector
  - rendezés Reference szerint
  - NUCLEO-F411RE
  - Periherals/Connectors Selector
    - Button: Nb = 1
    - LED: Nb = 1
- Load Project: read-only ---> OK
- Project > Generate Code
  - Project Name: pelda_kod
  - Project Location: /home/sebokb/STM32Toolchain/cubemx_test/
  - Code Generation ---> Open Folder

Projekt generálása CubeMX-szel

A teszt projektek között van egy részletes útmutató ehhez, ahol egy példán mutatom be a használatát. Ide kattintva oda tudsz ugrani.

STLink tool

Telepítés előkészülete

1. CMake telepítése
   1.1. forráskód letöltése: https://cmake.org/download
   1.2. Unix/Linux source: https://cmake.org/files/v3.8/cmake-3.8.0-rc1.tar.gz
   1.3. build: *./bootstrap && make && make install*

Telepítés

1. forráskód letöltése: https://github.com/texane/stlink
2. repo klónozása: *git clone https://github.com/texane/stlink.git*
3. klónozás helye: *~/STM32Toolchain/stlink/*
4. buildelés:
   4.1. cd stlink
   4.2. make
   4.3. cd build/Release && make install DESTDIR=_install
5. st-flash másolása:
   5.1. cd flash
   5.2. sudo cp st-flash /usr/bin

HEX fájl írása a board-ra:

./st-flash --format ihex write ~/workspace/test1/Debug/test1.hex

~~STM32Cube-HAL importálása Eclipse-es projektbe~~

- GNU-ARM plugin generált egy csontváz projektet
  - ebben benne van a CMSIS *(Cortex Microcontroller Software Interface Standard)*
  - gyártó független absztrakciós réteg a Cortex-M processzorokhoz
  - gyártó specifikus HAL (Hardware Abstraction Layer) szükséges ezen kívül: STM32Cube keretrendszer
- File ---> New ---> C Project
  - Hello World ARM Cortex-M C/C++ Project
  - Processor core: M4 (F4-hez)
  - Clock (Hz): 100000000 (100 MHz)
  - Flash size (KB): 512
  - SRAM (KB): 128
  - Trace output: None (no trace output)
  - Next
  - Vendor CMSIS name: DEVICE ---> stm32f4xx
  - Next

    - Next
    - Toolchain name: GNU Tools for ARM Embedded Processors (arm-none-eabi-gcc)
    - Toolchain path: /home/sebokb/STM32Toolchain/gnu-arm/bin
    - Finish
- Projekt felépítése
    - /src és /include mappákban a fő programrészek
    - main.c csak egy csontváz
    - /system mappában az ARM CMSIS csomag
    - HAL másolása a következő mappákba:
        - /system/include/stm32f4xx
        - /system/src/stm32f4xx
- *jelenleg nem szükséges*

OpenOCD telepítése és konfigurálása

Telepítés

- szükséges források és információk
- github-ról a forráskód letöltése
    - 64 bites Ubuntu-hoz: gnuarmeclipse-openocd-debian64-0.10.0-201701241841.tgz
    - telepítés és kicsomagolás:

```
sudo mkdir -p
/home/sebokb/STM32Toolchain/gnuarmeclipse/openocd/0.10.0-201701241841/
$ cd /home/sebokb/STM32Toolchain/gnuarmeclipse/openocd/0.10.0-201701241841/
$ sudo tar xvf ~/Downloads/gnuarmeclipse-openocd-debian64-0.10.0-201701241841.tgz
```

    - telepítés sikerességét ellenőrizhetjük a parancssorból:

```
$ /home/sebokb/STM32Toolchain/gnuarmeclipse/openocd/0.10.0-201701241841/bin$
./openocd --version
```

    - siker esetén az output:

```
GNU ARM Eclipse 64-bits Open On-Chip Debugger 0.10.0-00113-g0f83948
(2017-01-24-19:18)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
```

- UDEV alrendszer beállítása:

```
$ sudo cp
/home/sebokb/STM32Toolchain/gnuarmeclipse/openocd/0.10.0-201701241841/contrib/9
9-openocd.rules \
  /etc/udev/rules.d/
$ sudo udevadm control --reload-rules
```

    - az USB eszközök kezelése miatt szükséges ez

- USB hozzáférési jogok biztosítása:

```
sudo usermod -aG plugdev $FELHASZNÁLÓ
```

Ellenőrzés

- teszteljük le, hogy elérhető-e a boardunk config fájlja:

```
/home/sebokb/STM32Toolchain/gnuarmeclipse/openocd/0.10.0-201701241841/bin$
openocd -f board/st_nucleo_f4.cfg
```

  - mivel a boardunk egy STM32 Nucleo 64 (F411RE), ezért az *st_nucleo_f4.cfg* fájl szükséges a debugolás során
  - sikeres telepítés és jogok biztosítása esetén a kimenet a következő:

```
Open On-Chip Debugger 0.9.0 (2015-09-02-10:42)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
Info : The selected transport took over low-level target control. The results might differ
compared to plain JTAG/SWD
adapter speed: 2000 kHz
adapter_nsrst_delay: 100
none separate
srst_only separate srst_nogate srst_open_drain connect_deassert_srst
Info : Unable to match requested speed 2000 kHz, using 1800 kHz
Info : Unable to match requested speed 2000 kHz, using 1800 kHz
Info : clock speed 1800 kHz
Error: open failed
in procedure 'init'
in procedure 'ocd_bouncer'
```

Elérési útvonal beállítása

- Eclipse-en belül: Window fül -> Preferences -> Run/Debug -> OpenOCD
- Folder:
  */home/**sebokb**/STM32Toolchain/gnuarmeclipse/openocd/0.10.0-201701241841/bin*
    - ahol *sebokb* a felhasználónév

Fejlesztés

Teszt projekt: blinky

Új teszt projekt létrehozása Eclipse-ben

1. File -> New -> C project vagy C++ project
2. Project name: *led_villogas*
3. Project type: STM32F4xx C/C++ Project
4. Next
5. Chip family: STM32F411xE

6. Flash size (kB): 512
7. External clock (Hz): 8000000 (8 MHz)
8. Content: Blinky (blink a led)
9. Use system calls: Freestanding (no POSIX system calls)
10. Trace output: Semihosting DEBUG channel
11. pipák:
    - 11.1. Check some warnings
    - 11.2. Use -Og on debug
    - 11.3. Use newlib nano
    - 11.4. Exclude unused
12. Next
13. Next
14. Next
15. Toolchain name: GNU Tools for ARM Embedded Processors (arm-none-eabi-gcc)
16. Toolchain path: /home/**sebokb**/STM32Toolchain/gnu-arm/bin
    - 16.1. ahol *sebokb* a felhasználónév
17. Finish

Teszt projekt kódrészlete

- a tesztelés során egy LED villogtatását töltjük a board-ra
- ennek a fő része a következő:

```
while (1)
  {
   blink_led_on();
   timer_sleep(seconds == 0 ? TIMER_FREQUENCY_HZ : BLINK_ON_TICKS);

   blink_led_off();
   timer_sleep(BLINK_OFF_TICKS);

   ++seconds;
   // Count seconds on the trace device.
   trace_printf("Second %u\n", seconds);
  }
```

- amint fentebb látható, a LED be- és kikapcsolását végzi a kód, ezek között némi késleltetéssel
- illetve számolja az eltelt másodperceket, amiket a debug konzolon ki is írat folyamatosan

Teszt project módosítása a teszthez

- az Eclipse által generált kód nem a mi Nucleo 64 boardunkhoz tartozó kódot készít, ezért a user led port és pin számát be kell állítani, hogy tudja villogtatni
- az adatlap 23. oldala alapján a mi boardunkon a User LD2 a PA5-ön van
- erre kell módosítani a generált kódban található user led port és pin adatokat
    - Eclipse-en belül a Project explorer-ben nyissuk le az include mappát
    - a BlinkLed.h-ban módosítsuk a következő részt:

```
// Port numbers: 0=A, 1=B, 2=C, 3=D, 4=E, 5=F, 6=G, ...
#define BLINK_PORT_NUMBER         (SZÁM)
#define BLINK_PIN_NUMBER          (SZÁM)
#define BLINK_ACTIVE_LOW          (SZÁM)
```

- erre:

```
// Port numbers: 0=A, 1=B, 2=C, 3=D, 4=E, 5=F, 6=G, …
#define BLINK_PORT_NUMBER          (0)
#define BLINK_PIN_NUMBER           (5)
#define BLINK_ACTIVE_LOW           (0)
```

- mert az A porthoz tartozó szám a 0, az 5-ös pinhez természetesen az 5-ös számot kell módosítani, illetve

Teszt project debuggolása

1. forráskód fordítása (Build)
   1.1. kis barna kalapács (GUI sávon) vagy Project -> Build Project
2. debuggolás beállítása:
   2.1. kis bogár ikon (GUI sávon) vagy Run -> Debug Configurations
   2.2. GBD openOCD Debugging
   2.3. ikon: New launch configuration
   2.4. Project és C/C++ Application a jelenlegi projektünket jelentse
   2.5. Debugger fülön:
      2.5.1. Config options: *-f board/st_nucleo_f4.cfg*
      (mivel STM32F411RE boardunk van)
3. debuggolás indítása:
   3.1. kis bogár ikon -> 1 led_villogas Debug (led_villogas Debug névre kereszteltem a Debug Config-ban ezt a beállítást)
   3.2. Resume (F8)
   3.3. sikeres beállítások esetén a kimenet a debug konzolon:

```
Hello ARM World!
System clock: 84000000 Hz
Second 1
Second 2
Second 3
Second 4
Second 5
Second 6
Second 7
Second 8
Second 9
```

   3.3.1. a “Hello ARM World!” felirat a példa kódban szerepel, ahogyan a rendszer órajel frekijének kiírása is:

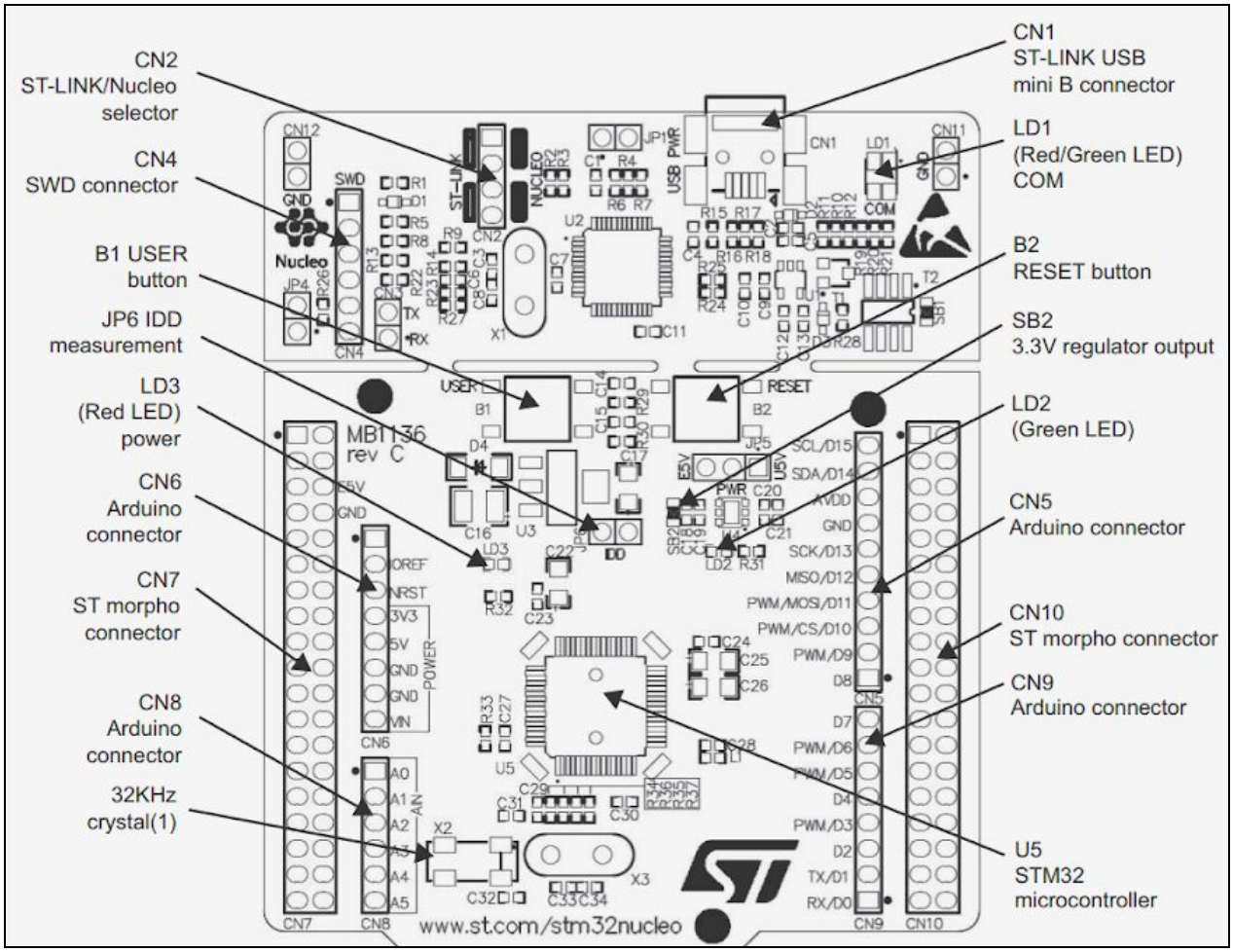
```
// Send a greeting to the trace device (skipped on Release).
trace_puts("Hello ARM World!");
// At this stage the system clock should have already been configured
// at high speed.
trace_printf("System clock: %u Hz\n", SystemCoreClock);
```

3.3.2. a másodpercek számlálása és kiírása már fentebb említve volt:

```
++seconds;
// Count seconds on the trace device.
trace_printf("Second %u\n", seconds);
```

Teszt projekt: gombnyomásra LED villogtatás

A Nucleo64 board felső layout-ja:



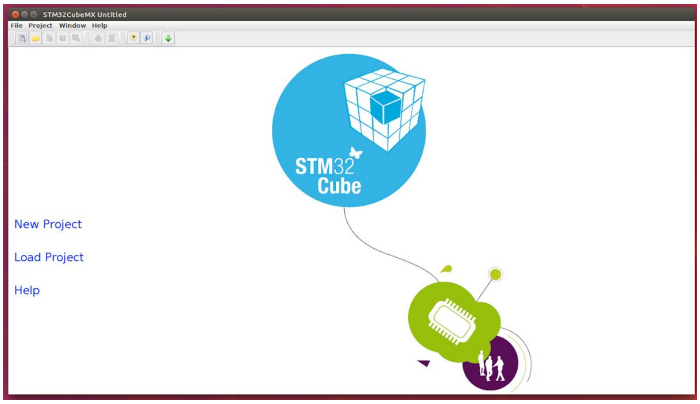
Az ezen látható B1 USER button egy kék nyomógomb, melyhez tetszőleges funkciót írhatunk szoftveresen.

Jelen projektben a gomb nyomva tartása alatt egy LED fog világítani, amely az LD2 (Green LED) felhasználói LED, amit szintén szoftveres megoldással használhatunk saját céljainkra.

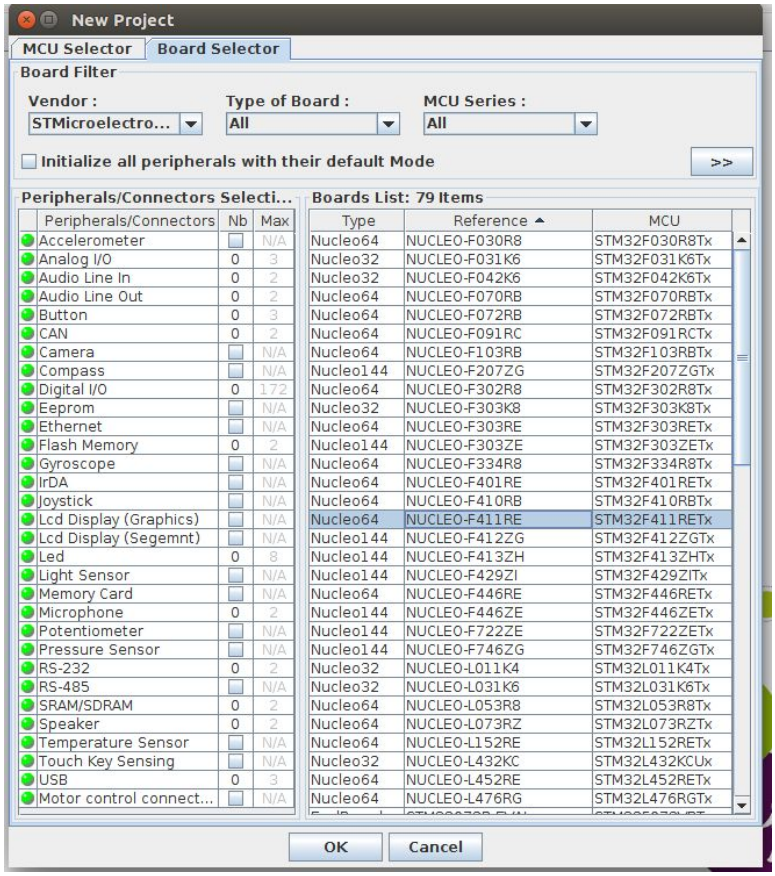
Az X3 oszcillátor nincsen a boardon, ez lenne az onboard oscillator, vagyis ami a rendszer órajelét adja.

Ehelyett a programozó boardon lévő X1 külső oszcillátort kell használni, ami egy 8 MHz-es kristályoszcillátor.

Indítsunk el az STM32CubeMX-et, ezután a következő kezdőképernyő fogad minket:



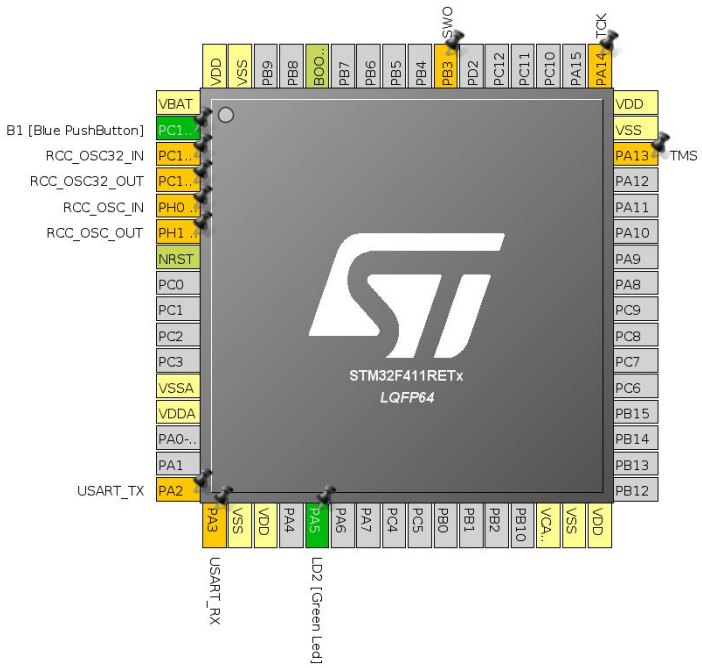
Itt a bal oldali New Project-re vagy a a File menüpont New Project (billentyűkombináció: CTRL-N) opcióra kattintva hozzunk létre új projektet.
Itt válasszuk ki a saját board-unkat.
Fent kattintsunk az MCU Selector-ra.
A Lines közül menjünk az STM32F411-re. A szűrt MCU-k közül keressük ki a miénket, ez az MCU oszlopban szereplő STM32F411RETx.
Ezután fent válasszuk ki a Board Selectort



.

A Type of Board a mi esetünkben: Nucleo64.
Az MCU Series a nekünk: STM32F4.

Majd a maradt lehetőségek közül válasszuk ki a miénket, a Reference oszlopban a NUCLEO-F411RE-et. Vigyázzunk, itt csak egyszer kattintsunk rá, különben hiányozni fog néhány további beállítás.
A bal oldali részen a Button mellett az Nb-t írjuk át 0-ról 1-re. A Led mellett szintén 1-e szerepeljen.
Alul kattintsunk az OK-ra, ami legenerálja a szükséges részeket az iménti beállításoknak megfelelően.
Ezután bejön a következő rész:



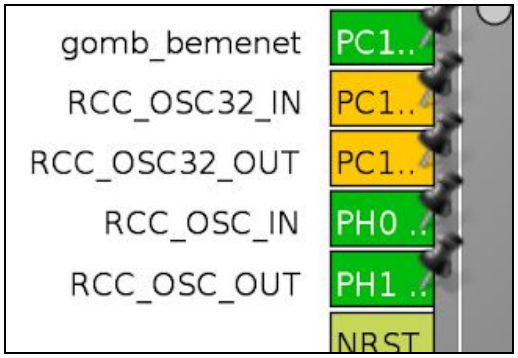
Itt a zölddel jelzett részek azok, amiket az előbb aktívra állítottunk és használni szeretnénk a projekt során.

Projekt elnevezése

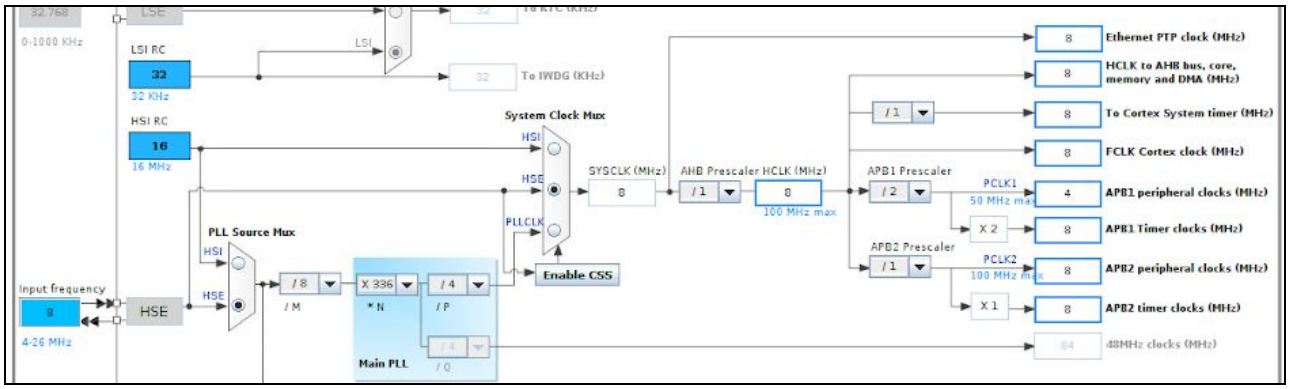
A Project menüben a Settings opcióra kattintva adjunk meg egy nevet a projektünknek, ezt a Project Name mezőben tehetjük meg.
Ez a lépés a kód generálása és a projekt mapparendszere miatt fontos.
Ha ezt nem tesszük meg, a kód generálása során felugrik ez az ablak és ott kell beállítani a nevet.

Rendszerórajel beállítása

A Pinout fülön a Configuration listában keressük meg az RCC feliratot.
Az RCC a Reset and Clock Control rövidítése, vagyis a rendszerünk resetelésének és órajelének a beállítása tartozik ide.
A High Speed Clock (HSE) menüjében válasszuk a Crystal/Ceramic Resonator opciót, hiszen egy 8 MHz-es külső kristályoszcillátorunk adja az órajelet.
Ez megjelenik a grafikus felületen is:



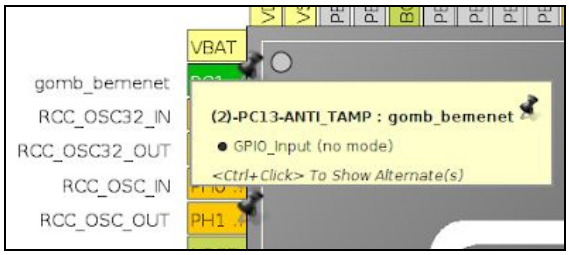
A Pinout melletti, Clock Configuration fülön tudjuk az órajellel kapcsolatos értékeket megadni. Ehhez a következő beállításokat kell választani:



Alul az Input frequency 8 Mzh, hiszen ennyi az X1 frekveciája.
A PLL Source Mux-nál válasszuk a HSE gombot.
Az M-es osztás értéke: 8.
A System Clock Mux-nál szintén a HSE-t válasszuk, hogy a rendszerünk megkapja a külső oszcillátor órajelét.

Gomb beállítása

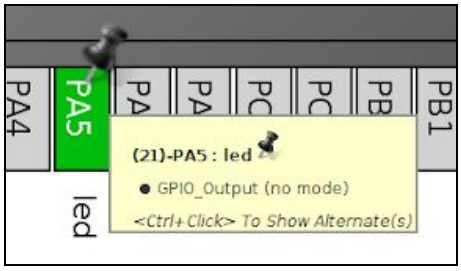
Látható, hogy a PC13 GPIO tartozik a kék user push buttonhoz, vagyis ezen a pinen keresztül érjük el a gombunkat.
Kattintsunk a zöld mezőre és válasszuk a GPIO_Input opciót, hiszen mi a gomb nyomkodásával egy bemeneti jelet küldünk majd a rendszerhez.
A zöld mezőre jobb klikkel kattintva át tudjuk nevezni a pint a Enter User Label opcióval. Ide írjuk be például a *gomb_bemenet* feliratot.
A gomb beállítása után ezt fogjuk látni:



Itt látható a pin neve: *gomb_bemenet*. A sárga részen pedig a pin típusa, vagyis a GPIO_Input.

LED beállítása

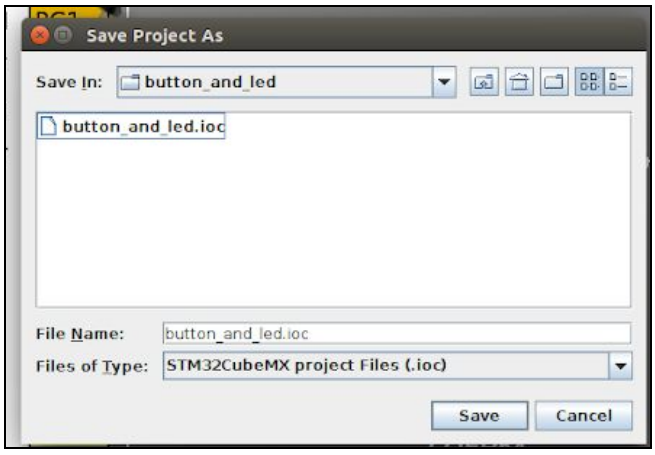
Az ábrán alul látható, hogy a felhasználói LED-hez a PA5 pin tartozik.
A zöld mezőre bal klikkel kattintva válassuk ki a GPIO_Output lehetőséget, hiszen a LED vezérlésére (be- vagy kikapcsolásához) egy kimeneti jelre van szükségünk.
Szintén ajánlott átnevezni, például *led*-re.
A végeredmény:



GPIO típusa: GPIO_Output (kimenet)
Pin neve: *led*.

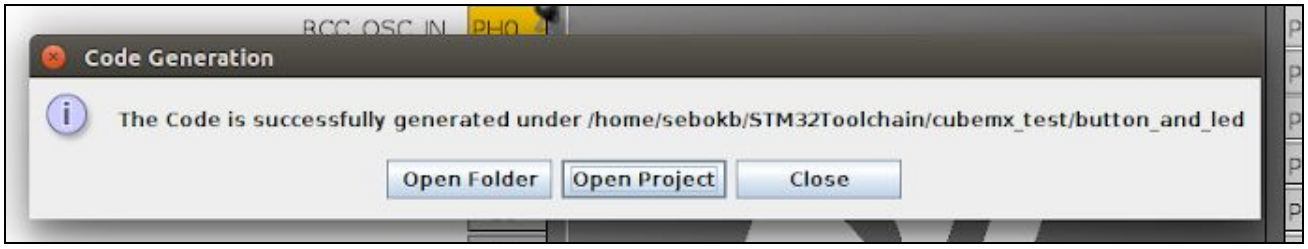
Projekt mentése

Ha később is szeretnénk használni, módosítani a projektet, akkor ajánlott elmenteni. Ezt a File menü Save Project As .. menüpontjával tudjuk megtenni. Itt tudunk nevet adni a projekt fájlnak, jén a *button_and_led* nevet választottam. Érdemes a projektnek egy új, saját mappát létrehozni, hogy elkülönüljön a többi projekt fájltól. A projekt fájl formátuma: *.ioc*, így a későbbi betöltés során majd a *button_and_led/button_and_led.ioc* fájlt kell keresnünk.
A mentés ablak:



Kód generálása

Elkészültünk a szükséges részek (push button, led) beállításával, így a CubeMX segítségével generáljuk le az ehhez szükséges kódot.
A menü Project pontján belül válasszuk a Generate Code lehetőséget.
Siker esetén a következő ablakot látjuk:

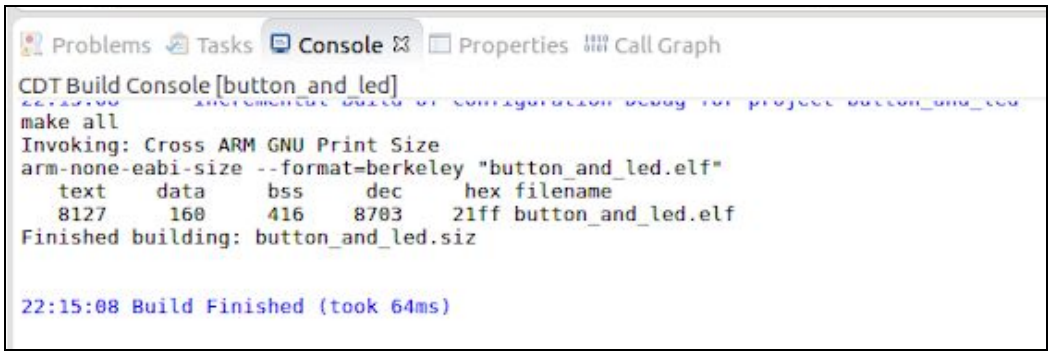


Kód importálása Eclipse IDE-be

Indítsuk el az Eclipse Mars fejlesztői környezetet.
Hozzunk létre egy új projektet a board-unknak megfelelően:

1. File → New → C Project
2. Projekt beállítása:
   2.1. Project name: button_and_led
   2.2. Project type: STM32F4xx C/C+ Project
   2.3. Toolchains: Cross ARM GCC
3. Target processor settings:
   3.1. Chip family: STM32F411RE
   3.2. External clock: 8000000 (hiszen 8 MHz-es oszcillátorunk van)
   3.3. Content: Empty (add your own content)
4. Finish

Most a CubeMX által generált kódot másoljuk be az Eclipse-es projekt mappájába.
Az Eclipse-es projekt helye: ~/workspace/mars/button_and_led
A CubeMX által generált kód helye: ~/STM32Toolchain/cubemx_test/button_and_led
A CubeMX/Src tartalmát másoljuk át az Eclipse-es button_and_led/src mappába, írjuk felül az ott lévő fájlokat. Ehhez hasonlóan a CubeMX/Inc tartalmát másoljuk át az Eclipse-es button_and_led/include mappába, szintén írjuk felül az eddigi tartalmat.
Teszteljük le, hogy minden szükséges fájl a helyén van-e a projekt fordításával. Ha minden rendben ment, alul a Console-ban megjelenik a Build Finished szöveg:



Kód megismerése

Nézzük meg és értsük meg a CubeMX által generált kódot.
Kezdjük a main.c-vel, ezen belül is a static void MX_GPIO_Init(void) függvénnyel:

```
161 */
162 static void MX_GPIO_Init(void)
163 {
164
165   GPIO_InitTypeDef GPIO_InitStruct;
166
167   /* GPIO Ports Clock Enable */
168   __HAL_RCC_GPIOC_CLK_ENABLE();
169   __HAL_RCC_GPIOH_CLK_ENABLE();
170   __HAL_RCC_GPIOA_CLK_ENABLE();
171   __HAL_RCC_GPIOB_CLK_ENABLE();
172
173   /*Configure GPIO pin Output Level */
174   HAL_GPIO_WritePin(led_GPIO_Port, led_Pin, GPIO_PIN_RESET);
175
176   /*Configure GPIO pin : gomb_bemenet_Pin */
177   GPIO_InitStruct.Pin = gomb_bemenet_Pin;
178   GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
179   GPIO_InitStruct.Pull = GPIO_NOPULL;
180   HAL_GPIO_Init(gomb_bemenet_GPIO_Port, &GPIO_InitStruct);
181
182   /*Configure GPIO pins : USART_TX_Pin USART_RX_Pin */
183   GPIO_InitStruct.Pin = USART_TX_Pin|USART_RX_Pin;
184   GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
185   GPIO_InitStruct.Pull = GPIO_NOPULL;
186   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
187   GPIO_InitStruct.Alternate = GPIO_AF7_USART2;
188   HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
189
190   /*Configure GPIO pin : led_Pin */
191   GPIO_InitStruct.Pin = led_Pin;
192   GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
193   GPIO_InitStruct.Pull = GPIO_NOPULL;
194   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
195   HAL_GPIO_Init(led_GPIO_Port, &GPIO_InitStruct);
196
197 }
198
```

A kommentezés elég jól leírja, hogy mi mit csinál, de nézzünk meg néhány részletet.
A __HAL_RCC_GPIOC_CLK_ENABLE() függvényhívás például a C porthoz tartozó GPIO-k órajelét engedélyezi. Ez ahhoz kell, hogy a C porton lévő gomb_bemenet pint (push button) használni tudjuk.
A HAL_GPIO_WritePin(led_GPIO_Port, led_Pin, GPIO_PIN_RESET) függvényhívás a jelenlegi egyetlen kimenetünket - a LED-et - kikapcsolja, hiszen most még csak inicializálunk, nincs szükség kimeneti értékekre.
A következő kódrészlet beállítja a gomb_menet pint:

```
GPIO_InitStruct.Pin = gomb_bemenet_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(gomb_bemenet_GPIO_Port, &GPIO_InitStruct);
```

Beállítja a megfelelő pint (C13) bemenetre.
A LED-et ehhez hasonlóan:

```
GPIO_InitStruct.Pin = led_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
```

Az A5 pinre és kimenetre állítja.
Nézzük meg, hogy a main.h-ban milyen értékek tartoznak az előbb használt led_Pin és hasonló makrókhoz:

```
#define gomb_bemenet_Pin GPIO_PIN_13
#define gomb_bemenet_GPIO_Port GPIOC
#define USART_TX_Pin GPIO_PIN_2
#define USART_TX_GPIO_Port GPIOA
#define USART_RX_Pin GPIO_PIN_3
#define USART_RX_GPIO_Port GPIOA
#define led_Pin GPIO_PIN_5
#define led_GPIO_Port GPIOA
#define TMS_Pin GPIO_PIN_13
#define TMS_GPIO_Port GPIOA
#define TCK_Pin GPIO_PIN_14
#define TCK_GPIO_Port GPIOA
#define SWO_Pin GPIO_PIN_3
#define SWO_GPIO_Port GPIOB
```

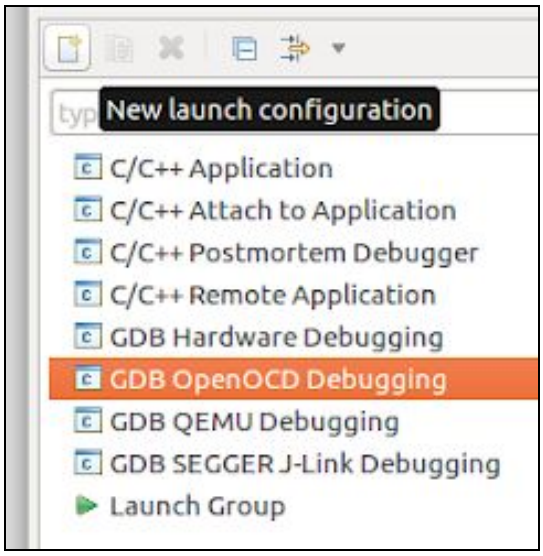
Látható, hogy például a gomb_bemenet a C13-as pinhez tartozik, ezért a gomb_bemenet_Pin értéke a GPIO sorszáma (GPIO_PIN_13), a gomb_bemenet_GPIO_Port értéke pedig GPIOC, hiszen a C porton van.
Ehhez hasonlóan a LED az A5-ös pinhez rendelése így történik:
#define led_Pin GPIO_PIN_5
#define led_GPIO_Port GPIOA

Kódolás

Kezdjük egy egyszerűbb résszel, másodpercenként kapcsoljuk be és ki a LED-et.
Ehhez a main.c-ben a végtelen ciklusba írjuk bele a következőt:
HAL_GPIO_WritePin(led_Pin, led_GPIO_Port, GPIO_PIN_RESET);

Tesztelés

Készítsünk Debug configot, hogy rá tudjuk tölteni a kódot a boardra és nyomon tudjuk követni, hogy mi történik.
Bogár ikon → Debug Configurations → GBD OpenOCD Debugging → New lauch configuration



A bogár ikon ( ) megnyomása kiváltható a Run → Debug configurations menüponttal, ez ugyanoda vezet.

A Debugger fülön a Config options mezőben szerepeljen a -f board/st_nucleo_f4.cfg beállítás, hogy tudja az Eclipse, hogy milyen paraméterekkel kell a debugolást indítani. Ez board és processzor specifikus, az st_nucleo_f4.cfg fájl a mi boardunk beállításait tartalmazza.

Github repository

```
git config --global user.name "sbence"
git config --global user.email "sebok.bence.003@gmail.com"
git config --global credential.helper cache
git config --global credential.helper 'cache --timeout=3600'
git clone https://github.com/sbence/nucleo64_stm32f411re.git
git remote -v
git remote add upstream https://github.com/sbence/nucleo64_stm32f411re
git remote -v
cp -R * ../stm32f411re/
git add *
git commit -m "first commit"
git push
```