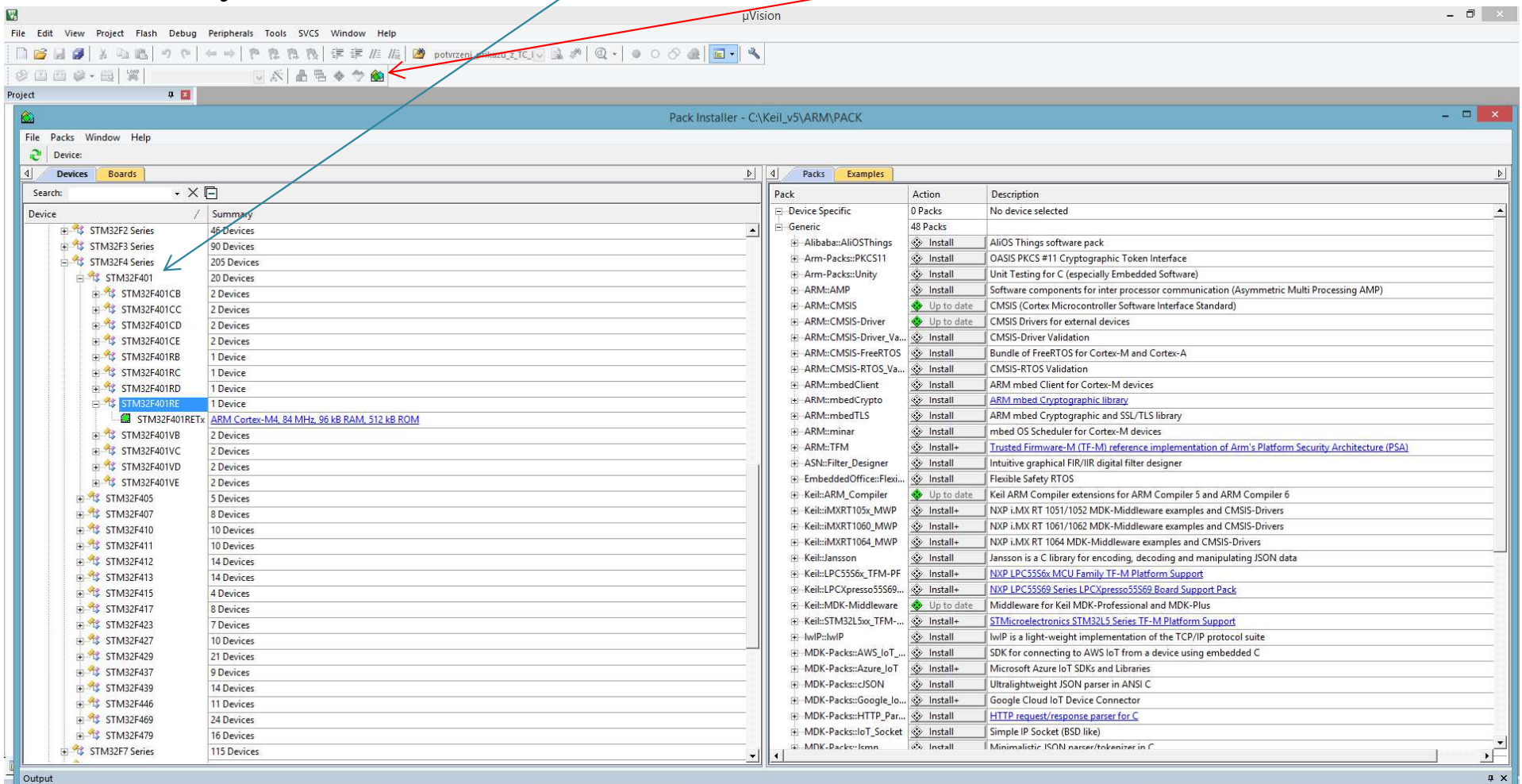


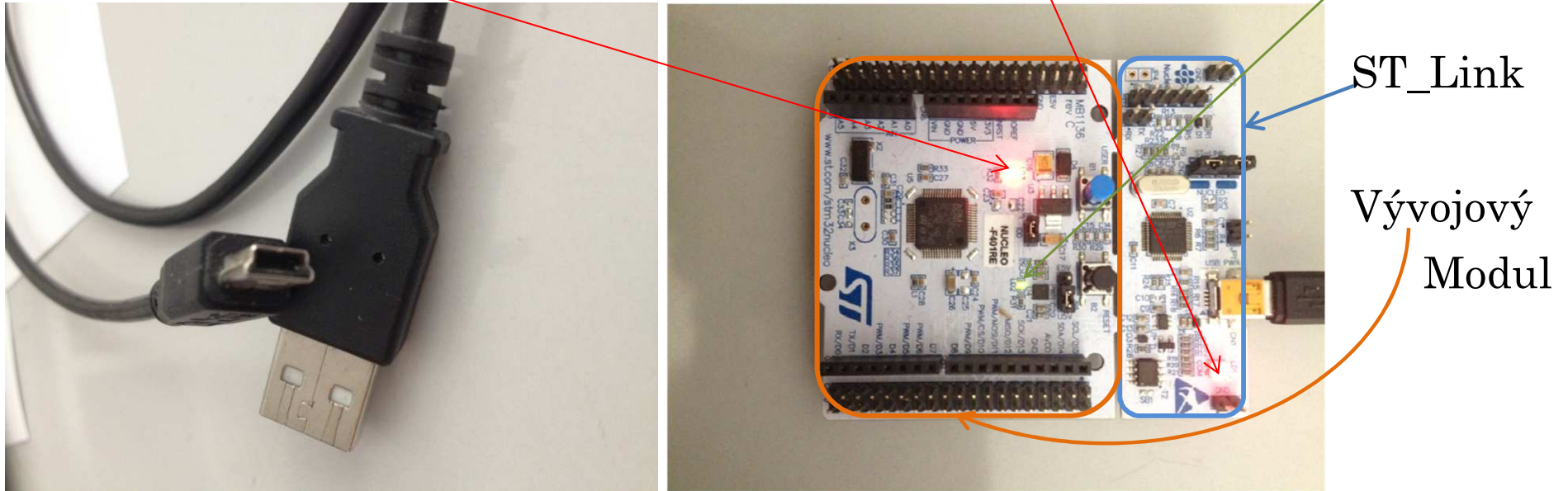
PRVNÍ KROKY K REALIZACI PROGRAMU

- ❖ Nainstalujeme si na PC program Keil uVision5 5.42a a STM32 ST-LINK Utility V3.8.0, který je k dispozici na CourseWare.
- ❖ Po instalaci nebo po prvním spuštění zmáčkne ikonu programu pack. Vybereme ST electronic, STM32F4 a poklepeme na ni. Nahrají se potřebné knihovny.



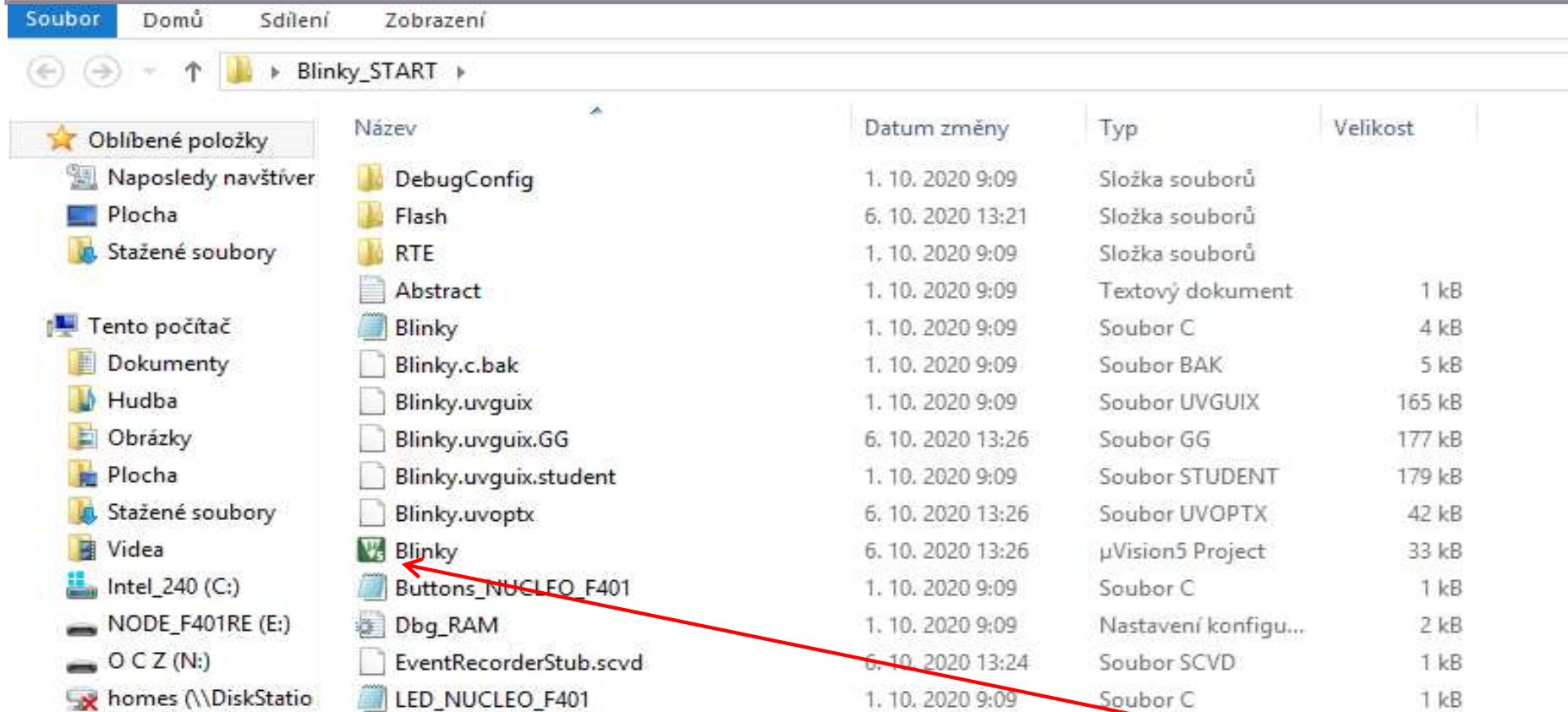
PRVNÍ KROKY K REALIZACI PROGRAMU

- ❖ Propojíme kabelem PC s vývojovým modulem. Na modulu se rozsvítí červená dioda a v části ST-Link svítí dioda též červeně viz. obrázek. Zelená dioda ~~může a nemusí svítit~~, záleží na posledním uloženém programu.



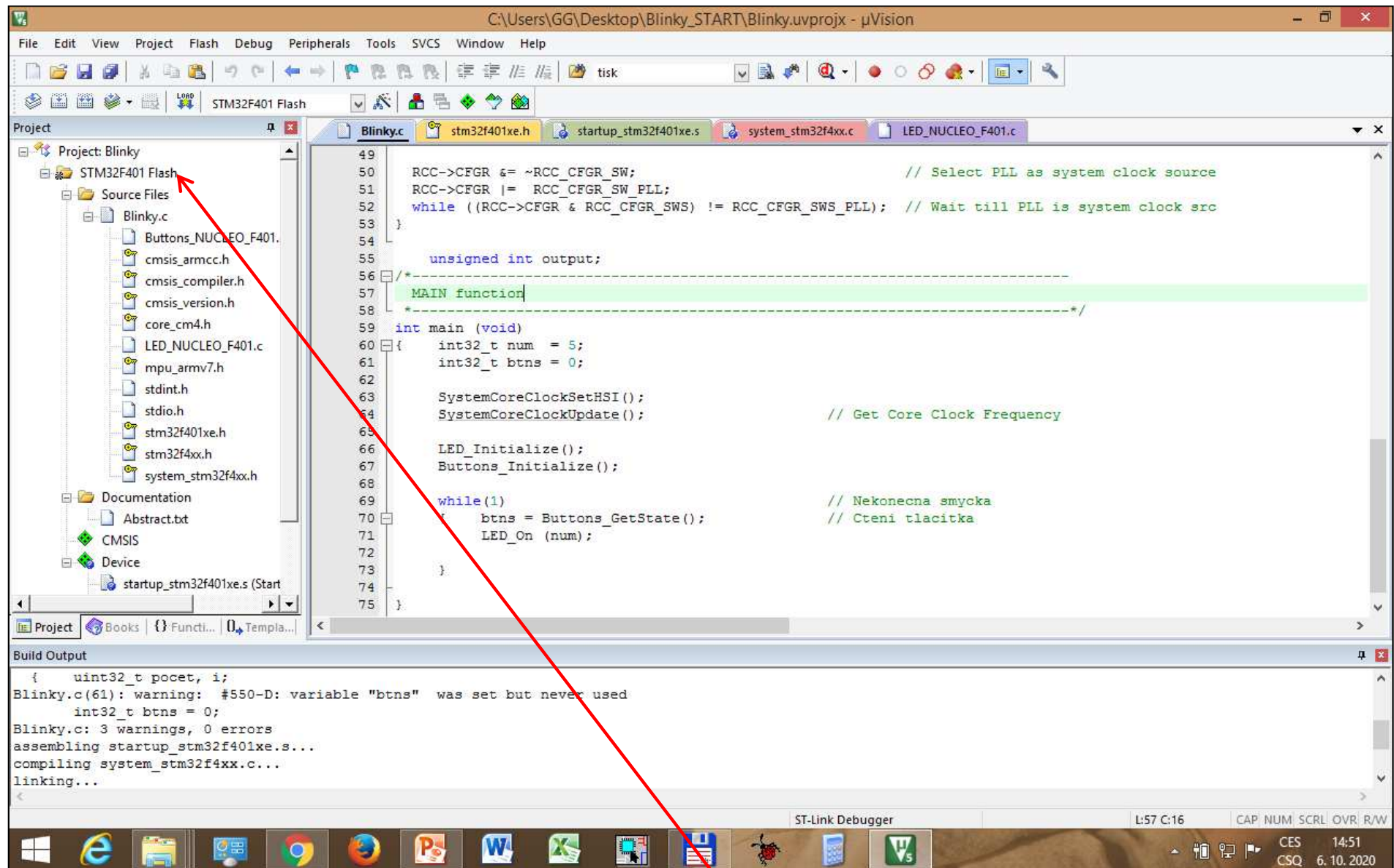
- ❖ Z Moodle si nakopírujeme soubor Blinky_START.zip, rozbalíme direktorář Blinky_START umístíme na disk.
- ❖ Otevřeme direktorář Blinky_START, kde budou umístěny následující programy.

PRVNÍ KROKY K REALIZACI PROGRAMU



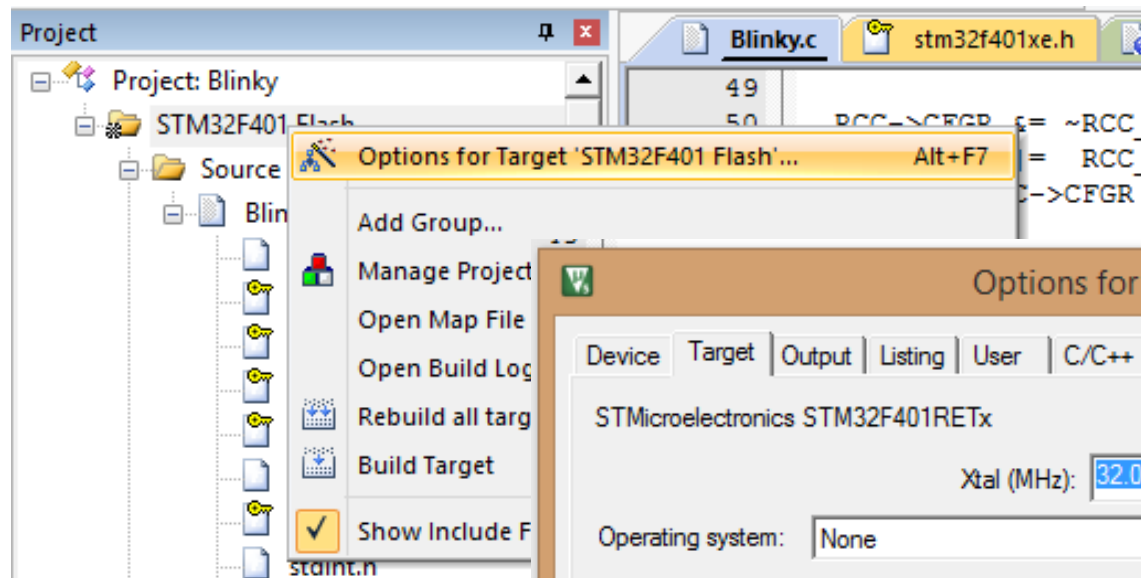
- ❖ Vývojové prostředí Keil uVision5 spustíme poklepáním na zelenou ikonku Blinky
- ❖ Pro seznámení se ze systémem a ověření funkčnosti bude potřeba doplnit programy Blinky.c, LED_NUCLEO_F401.c a Buttons_NUCLEO_F401.c. Dříve, než k tomu přistoupíte, zkontrolujeme propojení modulu s vývojovým prostředím.
- ❖ Po spuštění prostředí uvidíme obrazovku

PRVNÍ KROKY K REALIZACI PROGRAMU



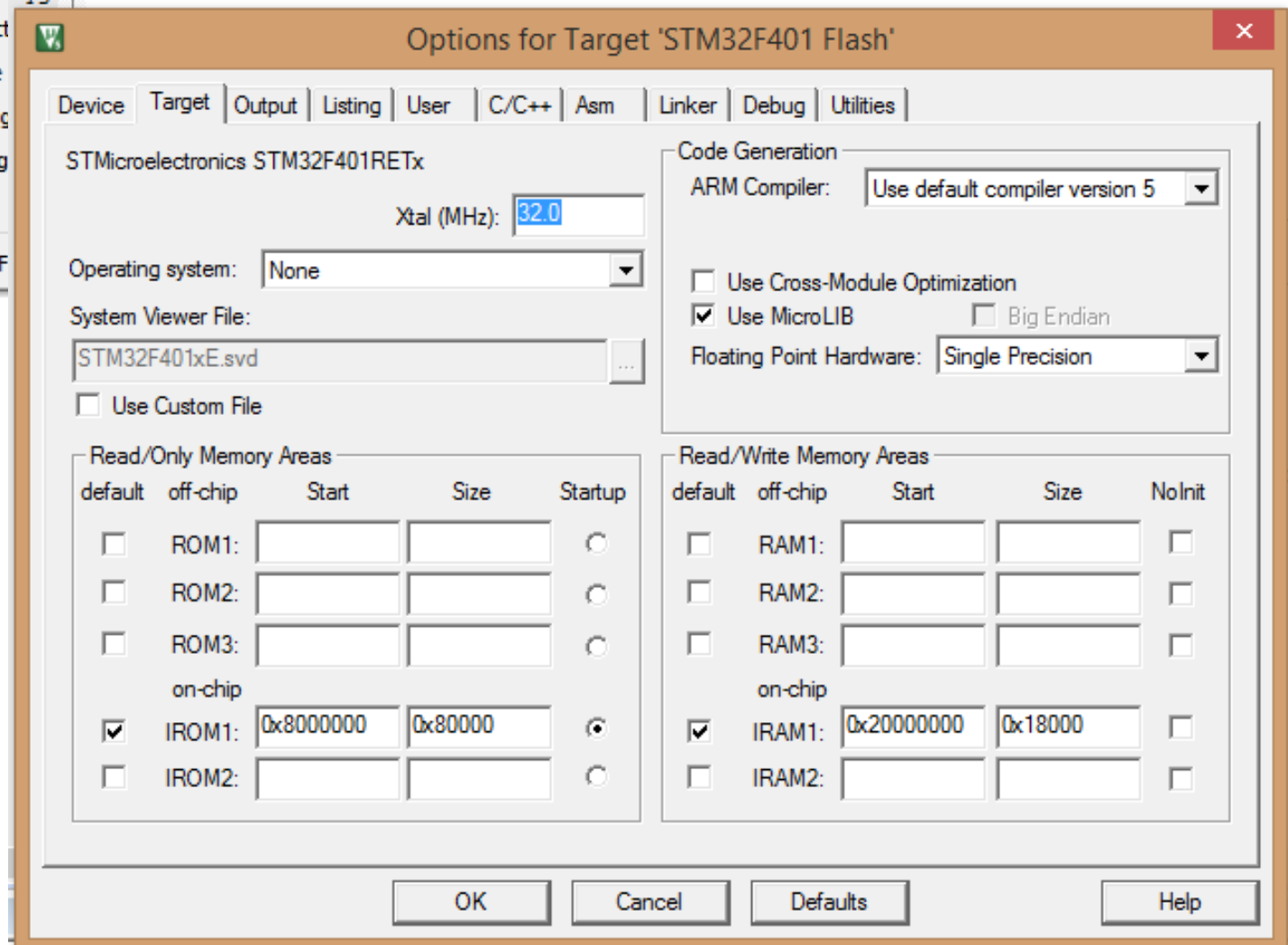
❖ Pravé tlačítko myši na název projektu

NEJDŮLEŽITĚJŠÍ NASTAVENÍ

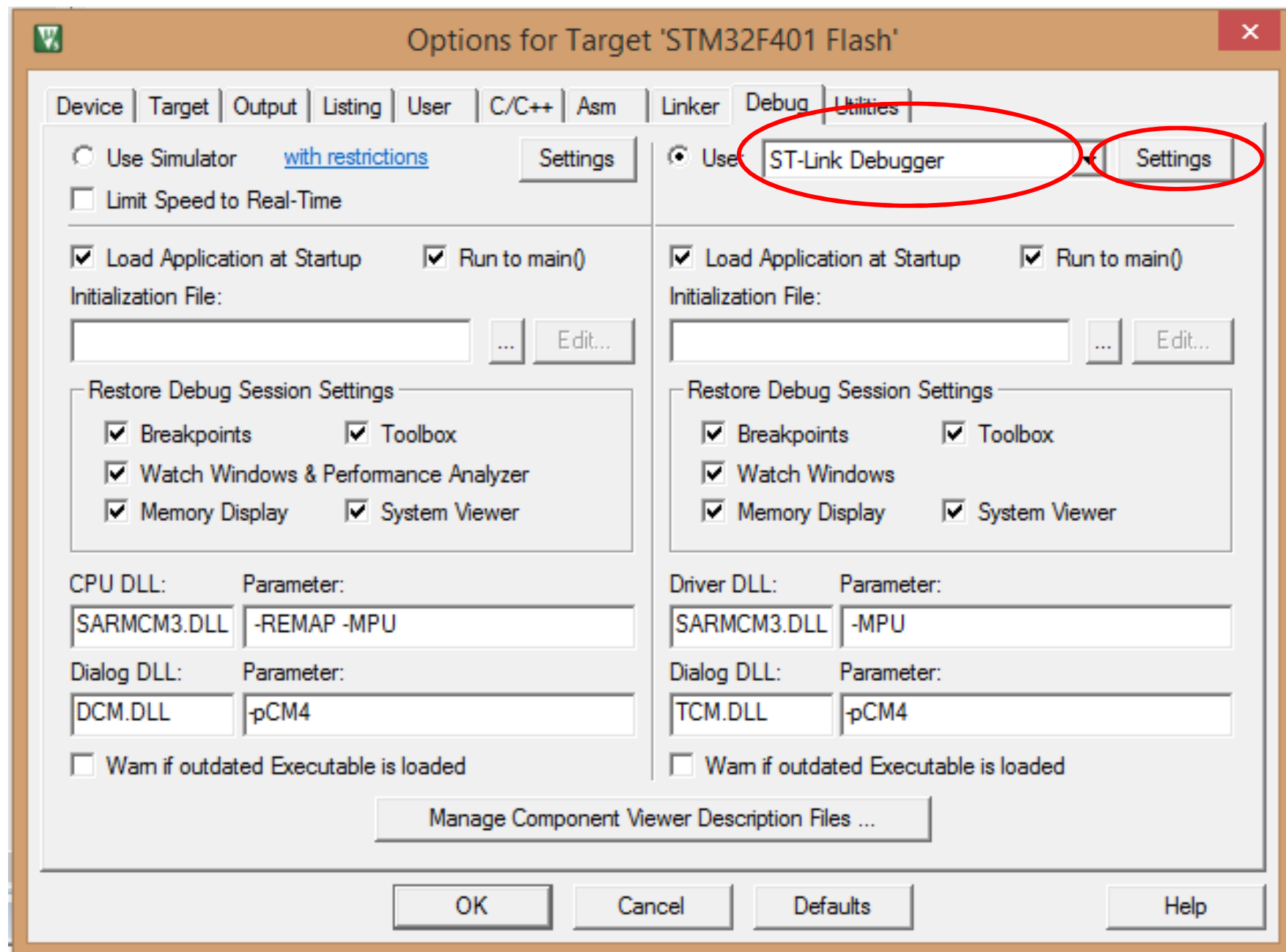


- ❖ Po vybrání položky Option for....
- ❖ Uvidíme okno

- ❖ Device – použitý procesor
- ❖ Target – hodinový kmitočet chybí u novějších verzí
- ❖ Debug – další stránka



NEJDŮLEŽITĚJŠÍ NASTAVENÍ



NEJDŮLEŽITĚJŠÍ NASTAVENÍ

Cortex-M Target Driver Setup

Debug | Trace | Flash Download | Pack

Debug Adapter
Unit: **ST-LINK/V2-1**

☐ Shareable ST-Link

Serial Number:
066EFF575550897767162335

Version: HW: **V2-1** FW: **V2J29M18**

☐ Check version on start **Update**

Target Com
Port: **SW**

Clock
Req: 10 MHz Selected: 0 MHz

SW Device

IDCODE	Device Name	Move
0x2BA01477	ARM CoreSight SW-DP (ARM Core	Up Down

☒ Automatic Detection ID CODE:

☐ Manual Configuration Device Name:

Add **Delete** **Update** IR len: AP: **0**

Debug

Connect & Reset Options
Connect: **Normal** Reset: **Autodetect**

☒ Reset after Connect ☐ Stop after Reset

Cache Options
☒ Cache Code
☒ Cache Memory

Download Options
☒ Verify Code Download
☐ Download to Flash

OK **Storno** **Použít**

NEJDŮLEŽITĚJŠÍ NASTAVENÍ

Cortex-M Target Driver Setup

Debug | Trace | Flash Download | Pack

Download Function

 ☐ Erase Full Chip ☒ Program
☒ Erase Sectors ☒ Verify
☐ Do not Erase ☒ Reset and Run

RAM for Algorithm

Start: 0x20000000 Size: 0x0000008

Programming Algorithm

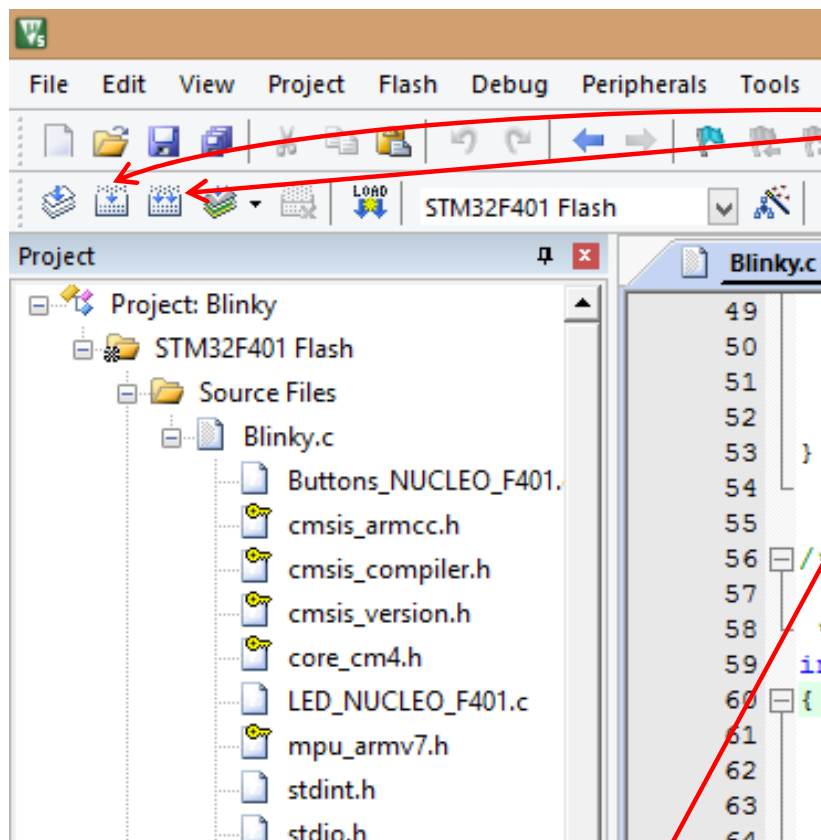
Description	Device Size	Device Type	Address Range
STM32F4xx 512kB Flash	512k	On-chip Flash	08000000H - 0807FFFFH

Start: Size:

Add Remove

OK Storno Použít

PŘEKLAD PROGRAMU

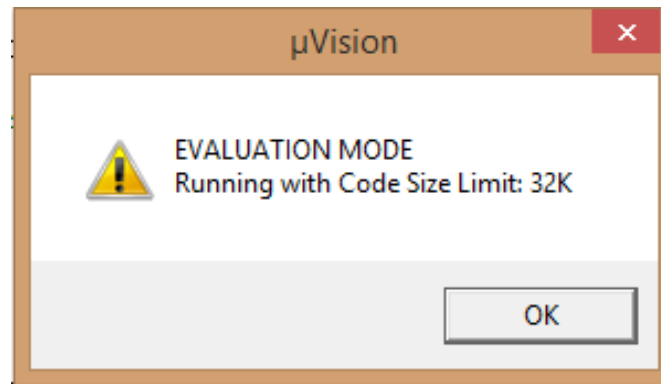


- ❖ Překlad programu zajistíme pomocí ikon
- ❖ Po překladu zkontrolovat zda je bez chyb, varování jsou většinou nevýznamná
- ❖ Code – velikost strojového kódu programu bytech
- ❖ RO-data – velikost konstant v programu bytech
- ❖ RW-data – velikost paměti pro proměnné bytech

```
Build Output
Blinky.c: 3 warnings, 0 errors
assembling startup_stm32f401xe.s...
compiling system_stm32f4xx.c..
linking...
Program Size: Code=694 RO-data=462 RW-data=4 ZI-data=1028
".\Flash\Blinky.axf" - 0 Error(s), 3 Warning(s).
Build Time Elapsed: 00:00:03
```

SPUŠTĚNÍ PROGRAMU

- ❖ Spuštění programu zajistíme v záložce Debug – Start/Stop Debug Session nebo pomocí ikonky lupy s písmenem d. Je-li vše v pořádku, pak by se v levém dolním rohu měl na krátkou dobu objevit modrý proužek indikující přenos strojového kódu do vývojového modulu a následující zpráva.



- ❖ Po zmáčknutí OK přecházíme do okna Debug, které je na následující stránce. Pro začátek se spokojíme s následujícími okny:
 - ✓ Okno se zdrojovým programem nebo obsahem zvoleného souboru
 - ✓ Okno **Disassembly** s překladem řádku, na který ukážete ve zdrojovém programu.
 - ✓ Okno **Project** se soubory projektu nebo se stavem registrů **Registers**
 - ✓ Můžeme si aktivovat okno **Memory, Watch** a další.

OBRAZOVKA DEBUG PROSTŘEDÍ

The screenshot displays the µVision IDE interface for debugging the Blinky program. The main window shows the source code of the program, which is a simple LED blinker. The program is running, and the main function is visible in the source code panel. The registers panel shows the current state of the CPU registers, and the disassembly panel shows the corresponding assembly instructions. The watch and memory panels provide additional information about the program's state.

Registers Panel:

Register	Value
R0	0x08000295
R1	0x20000408
R2	0x00000000
R3	0x0800043D
R4	0x08000484
R5	0x08000484
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x20000408
R14 (LR)	0x08000425
R15 (PC)	0x08000294
xPSR	0x61000000

Disassembly Panel:

```

60: {   int32_t num = 5;
    0x08000294 2405   MOVS      r4,#0x05
61:   int32_t btns = 0;
62:
0x08000296 BF00   NOP
63:   SystemCoreClockSetHSI();
0x08000298 F7FFFF93   BL.W      SystemCoreClockSetHSI (0x080001C2)
64:   SystemCoreClockUpdate();           // Get Core Clock Frequency
    
```

Source Code Panel:

```

53: }
54:
55: unsigned int output;
56: /*-----
57:  MAIN function
58:  *-----
59: int main (void)
60: {
61:     int32_t num = 5;
62:     int32_t btns = 0;
63:
64:     SystemCoreClockSetHSI();
65:     SystemCoreClockUpdate();           //
66:     LED_Initialize();
67:     Buttons_Initialize();
68:
69:     while(1)
    
```

Watch Panel:

Name	Value	Type
num	0x08000484	int

Memory Panel:

Address	Value
0x20000000	00 24 F4 00 00 00 00 00 00 00 00 00 00 00 00 00
0x2000000E	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x2000001C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x2000002A	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Call Stack + Locals Panel:

Name	Location/Value	Type
main	0x00000000	int f()
num	0x08000484	auto - int
btns	<not in scope>	auto - int

Command Panel:

```

Running with Code Size Limit: 32K
Load "C:\\Users\\GG\\Desktop\\Blinky_START\\Flash\\Blinky.axf"
>
ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet
    
```

ST-Link Debugger Panel:

t1: 0.00016820 sec L:62 C:6 CAP NUM SCRL OVR R/W

System Tray:

CES 16:28
CSQ 6. 10. 2020

Spuštění programu - F5, dioda ST Link – bliká a mění barvu. Krokování - F11 nebo F10. Nastavení systémových hodin nekrokovat přeskočit na Breakpoint.

ÚPRAVA PROGRAMU BLINKY PRO REALIZACI ZÁKLADU ÚLOHY 1

- ❖ Program Blinky.c obsahuje dva systémové podprogramy a dva inicializační podprogramy pro tlačítko a diodu LED. Oba se skrývají v souborech LED_NUCLEO_F401.c a Buttons_NUCLEO_F401.c. V těchto podprogramech jsou prázdné podprogramy, do kterých je potřeba napsat inicializaci vývodů viz. MAM_2025-Cviceni_4.pdf a soubor Konfigurace GPIO bran.pdf. Dále je potřeba doplnit rutinu pro čtení tlačítka a rozsvícení a zhasnutí LED.

```
SystemCoreClockSetHSI();  
SystemCoreClockUpdate(); // Get Core Clock Frequency  
  
LED_Initialize();  
Buttons_Initialize();
```

- ❖ Vámi navržená řešení je možné porovnat s řešeními uvedenými na následujících dvou stranách.
- ❖ Pro další úlohy již nebudeme vypisovat inicializaci I/O vývodu uvedeným způsobem, ale použijeme připravenou knihovnu **Nastaveni_GPIO.c**. Ta nám umožní inicializovat vývod zápisem na jeden řádek ve zdrojovém kódu takto:

```
PIN_OUTPP_Initialize (GPIOA,9);  
PIN_OUTPP_Initialize (GPIOB,5);  
PIN_IN_Un_Initialize (GPIOA,1);
```

KONFIGURACE A OVLÁDÁNÍ LED NA BRÁNĚ GPIOA VÝVOD PA5

```
// PODPROGRAM PRO INICIALIZACI A NÁSLEDNÉ ROSVÍCENÍ a ZHASNUTÍ LED NA BRÁNĚ PA5

#include "stm32f4xx.h" // Device header
#define LED 5

int32_t LED_Initialize (void) // void možno nahradit proměnou
{
    RCC->AHB1ENR |= (1ul << 0); // Povolení hodinového signálu pro GPIOA
    // Nastavení vývodu PA.5 (Zelena LED) na výstup push-pull
    // bez upnutí k napájení nebo zemi
    GPIOA->MODER  &= ~(3ul << 2*LED); // Stav po nulování (rušení předchozího
stavu)
    GPIOA->MODER  |= ((1ul << 2*LED)); // Vystup
    GPIOA->OTYPER  &= ~(1ul << LED); // Push-Pull
    GPIOA->OSPEEDR &= ~(3ul << 2*LED); // Rušení předchozího stavu
    GPIOA->OSPEEDR |= ((1ul << 2*LED)); // Medium speed
    GPIOA->PUPDR  &= ~(3ul << 2*LED); // Bez Pull DOWN i Pull UP
    return (0);
}

int32_t LED_On (uint32_t num)
{
    if (num < 16)
    {
        GPIOA->BSRR |= (1ul << LED); }
    return (0);
}

int32_t LED_Off (uint32_t num)
{
    if (num < 16)
    {
        GPIOA->BSRR |= (1ul << LED)<<16; }
    return (0);
}
```

KONFIGURACE A ČTENÍ STAVU TLAČÍTKA NA VÝVODU PC13

```
// PODPROGRAM PRO INICIALIZACI A NÁSLEDNÉ ZJIŠTĚNÍ STAVU TLAČÍTKA NA VÝVODU PC13

#include "stm32f4xx.h"                                // Device header

int32_t Buttons_Initialize (void)
{
    RCC->AHB1ENR |= (1ul << 2);                      // Povolení hodinového signálu pro GPIOC
                                                    // V registru AHB1ENR nastaven bit 2
                                                    // (počítáno od 0)
    // Nastavení vývodu PC.13 (Modré tlačítko) na vstup, bez upnutí k napájení nebo
    zemi
    GPIOC->MODER   &= ~(3ul << 2*13);                // Vstup
    GPIOC->OSPEEDR &= ~(3ul << 2*13);                // Rušení předchozího stavu
    GPIOC->OSPEEDR |= (1ul << 2*13);                  // Medium speed
    GPIOC->PUPDR   &= ~(3ul << 2*13);                // Bez upínacích odporu
    return (0);
}

uint32_t Buttons_GetState (void)
{
    if ((GPIOC->IDR & (1ul << 13)) == 0) return(1);
    else return(0);
}
```

VOLNĚ POUŽITELNÉ GPIO_x VÝVODY, KTERÉ MŮŽEME KONFIGUROVAT

GPIOA – PA0 až PA12,

PA2 a PA3 - realizují sériový kanál využívaný ST Linkem

PA13, PA14, PA15 – realizují rozhraní JTAG/SWO

zápis do PA2, 3, 13, 14, 15 vede ke ztrátě komunikace s modulem
Odstranění je popsáno na následujících dvou stránkách

GPIOB – PB0 až PB10, PB12 až PB15

GPIOC – PC0 až PC15

VÝVODY PŘEDURČENÉ PRO ALTERNATIVNÍ FUNKCE

USART2 – PA2, PA3 - nejsou propojeny na konektor

**A/D převodník (skupina A) – PA0, PA1, PA2, PA3, PA4÷7, PB1,
PB12÷15, PC0÷5**

Komparační systém kanál 1 – PA6, PB4, PC6

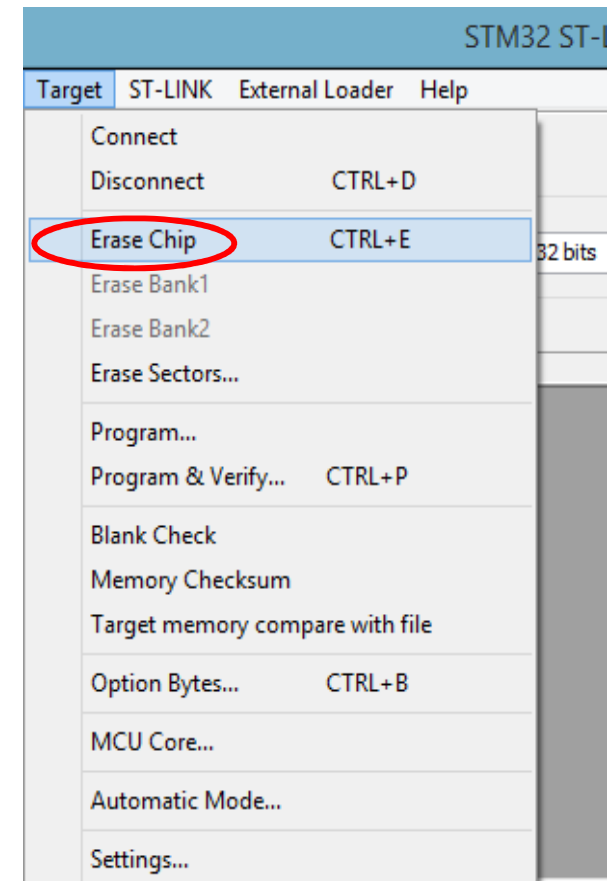
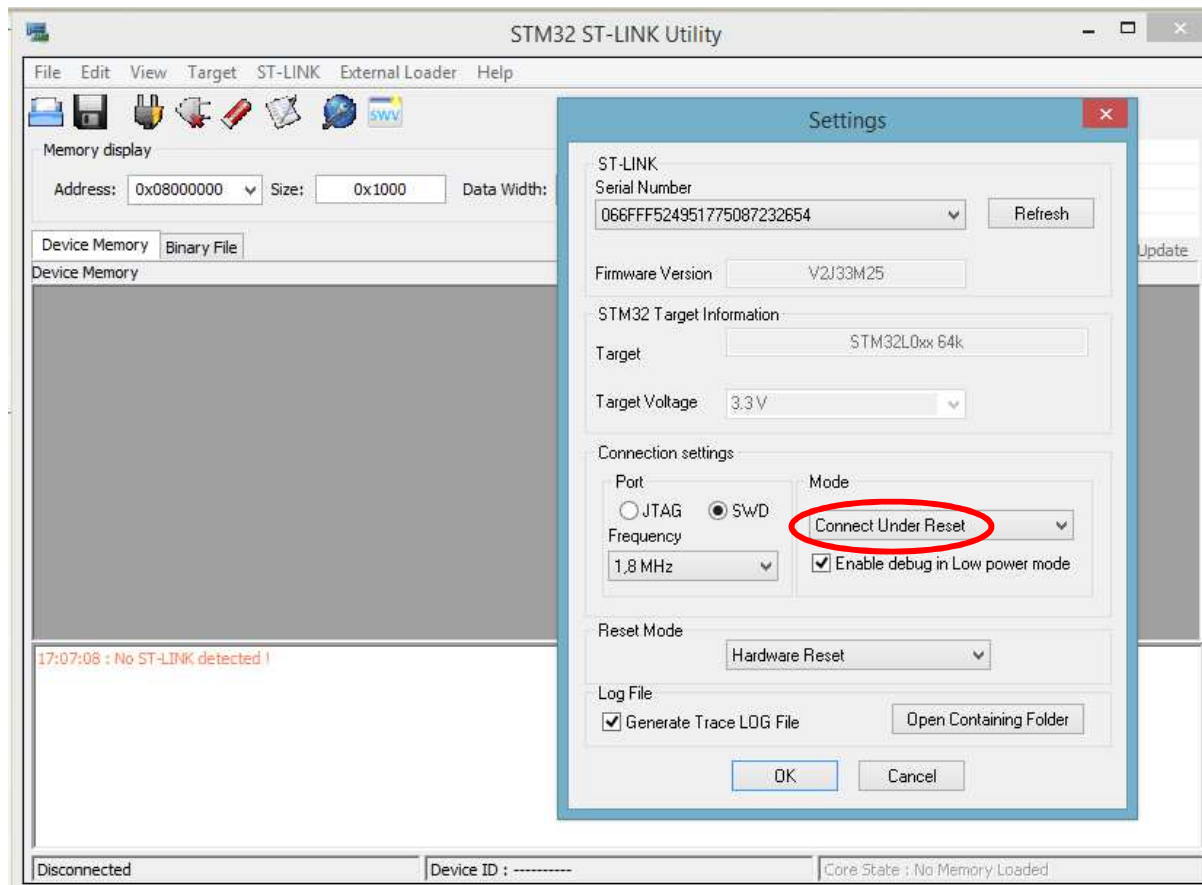
kanál 2 – PA7, PB5, PC7

Záchytný systém kanál 1 – PA6, PB4, PC6

OŽIVENÍ MODULU PO ŠPATNÉM ZÁPISU DO BRÁNY PA

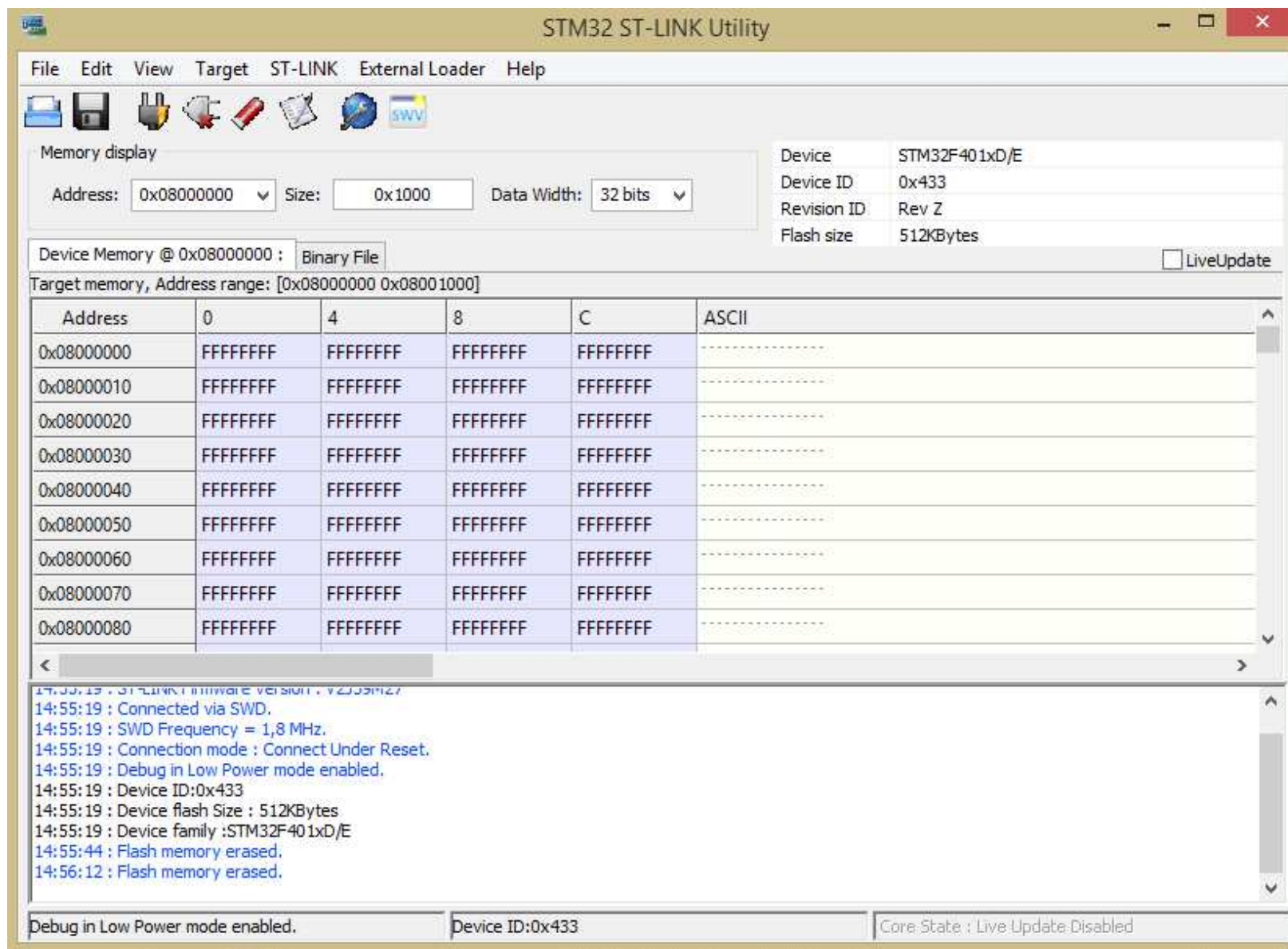
Nefungující modul NUCLEO v případech správně nainstalovaného vývojového prostředí Keil i ST-Link. K situaci dochází při zápisu do celé brány PA. **Odstranění:**

- ❖ Spustit ST link
- ❖ V položce *Target-Settings* nastavit variantu Connect Under Reset a OK
- ❖ Erase Chip

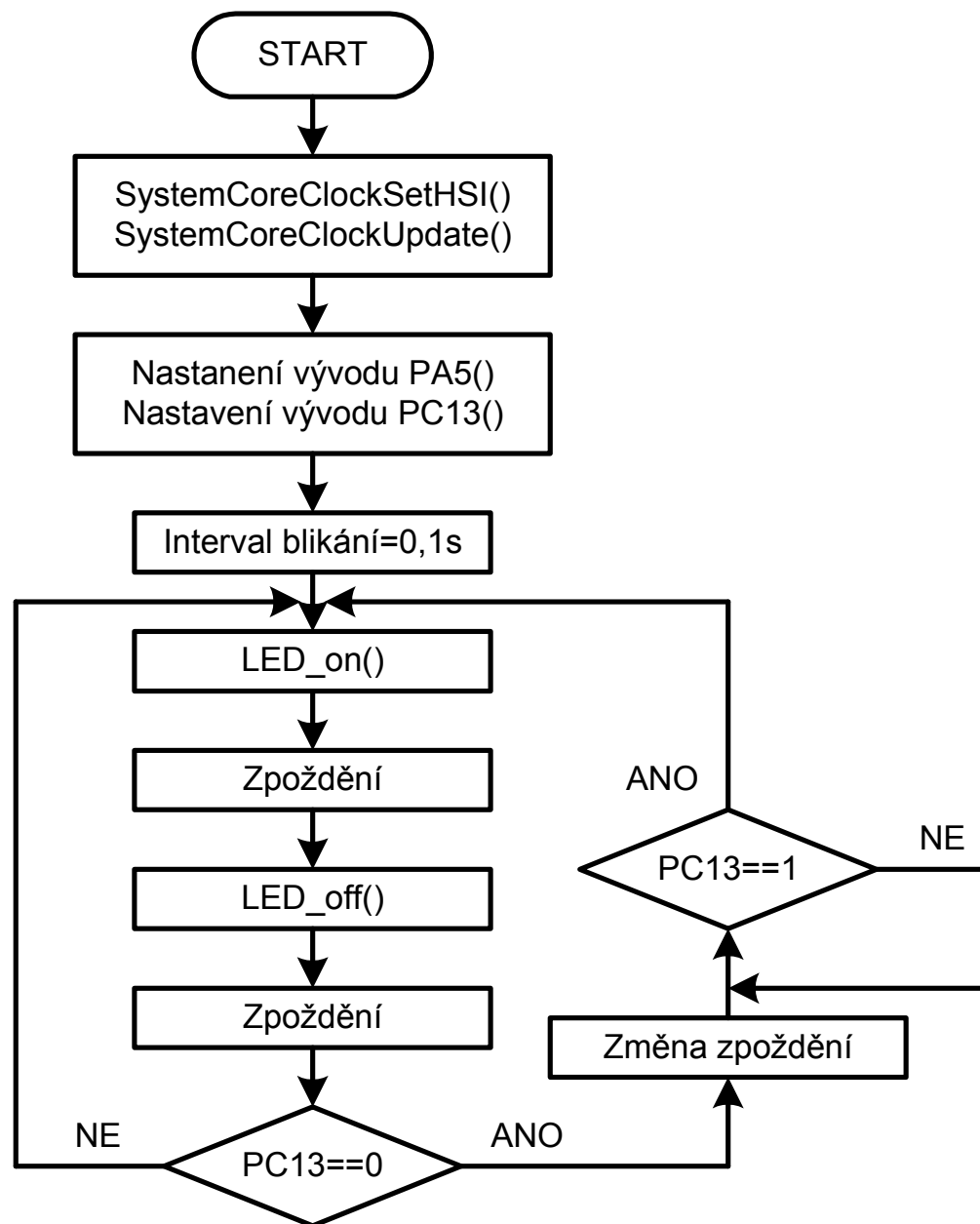


OŽIVENÍ MODULU PO ŠPATNÉM ZÁPISU DO BRÁNY PA

Následně by mělo okno ST Link vypadat takto:



IDEOVÉ ŘEŠENÍ ZÁKLADU ÚLOHY 1



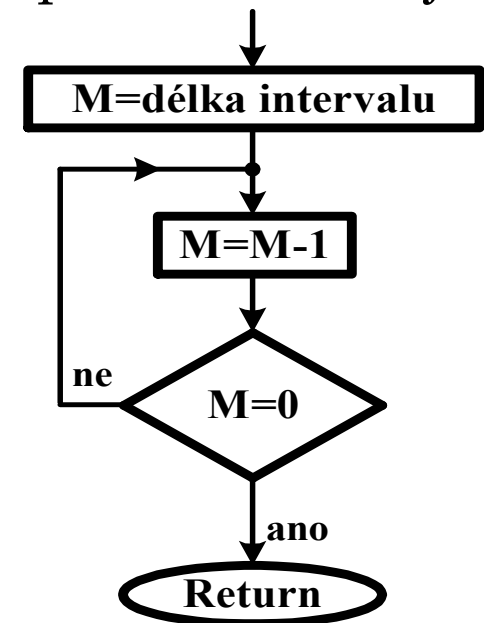
PROGRAMOVĚ GENEROVANÉ ZPOŽDĚNÍ

Vytvoření potřebného zpoždění můžeme realizovat

- Monostabilním obvodem (číslico-analogové řešení)
- Čítačem počítajícím impulzy hodinového signálu (čisté číslicové řešení). V procesorech jsou proto (čítače/časovače) podporované přerušovacím systémem
- Zpožděním vytvořeným dobou trvání programu viz. obrázek.

Zpoždění realizujeme pomocí podprogramu, v kterém zkusmo nastavíme hodnotu odpovídající požadovanému zpoždění. Ta je postupně dekrementována/ inkrementována za pomoci instrukcí. Stabilita programovém řešení Zpoždění je dána stabilitou synchronizačního hodinového signálu procesoru za předpokladu, že:

- Instrukce trvají vždy stejně dlouhou dobu
- Procesor nevykonává jinou činnost, než je vlastní generování zpoždění. Např. se neobjeví obsluha přerušení.



PROGRAMOVĚ GENEROVANÉ ZPOŽDĚNÍ V JAZYCE C A ASEMLERU

```
const uint32_t doba_zpozdeni = 0x22800;    // nutno vyzkoušet dobu trvání. Změna hodnoty
                                           // může způsobit v překladu použití jiné instrukce
                                           // a díky tomu dojde ke skokové změně zpoždění.

void Delay (uint32_t pocet_ms)
{
    uint32_t pocet, i;
    for (i = 0; i < pocet_ms; i++)
    {
        for (pocet = 0; pocet < doba_zpozdeni; pocet++) i=i;
    }
}

;*****
;* Jméno funkce          DELAY
;* Popis                 Softwarové zpoždění procesoru
;* Mění stav registrů    R0 a R3
;* Vstup                 R0 = počet opakování cyklu zpoždění
;* Vystup                Žádný
;* Komentář              Podprogram zpozdí průběh vykonávání programu
;*****
DELAY                ; Navěstí začátku podprogramu
                    PUSH    {LR}    ; Uložení hodnoty návratové adresy - LR do zásobníku
WAIT1
                    LDR      R3, =doba    ; Vložení konstanty doba pro prodlevu do R3
WAIT                SUBS    R3, R3, #1    ; Odečtení 1 od R3,tj. R3 = R3 - 1 a nastavení příznakového
                                           ; registru
                    BNE     WAIT        ; Skok na navěstí při nenulovosti R3 (skok dle příznaku)
                    SUBS    R0, R0, #1    ; Odečtení 1 od R0,tj. R0 = R0 - 1 a nastavení příznakového
                                           ; registru
                    BNE     WAIT1       ; dokud není nula v R0, skočí na wait1
                    POP      {LR}        ; Návrat z podprogramu, obnovení hodnoty LR ze zásobníku
                    BX      LR          ; a návrat do hlavního programu
                    ; Nebo jednodušší varianta POP {PC} místo předchozích dvou řádků
                    POP      {PC}
;*****
END                ;Konec programu, jakýkoliv kód za tímto řádkem překladač nepřełoży
```