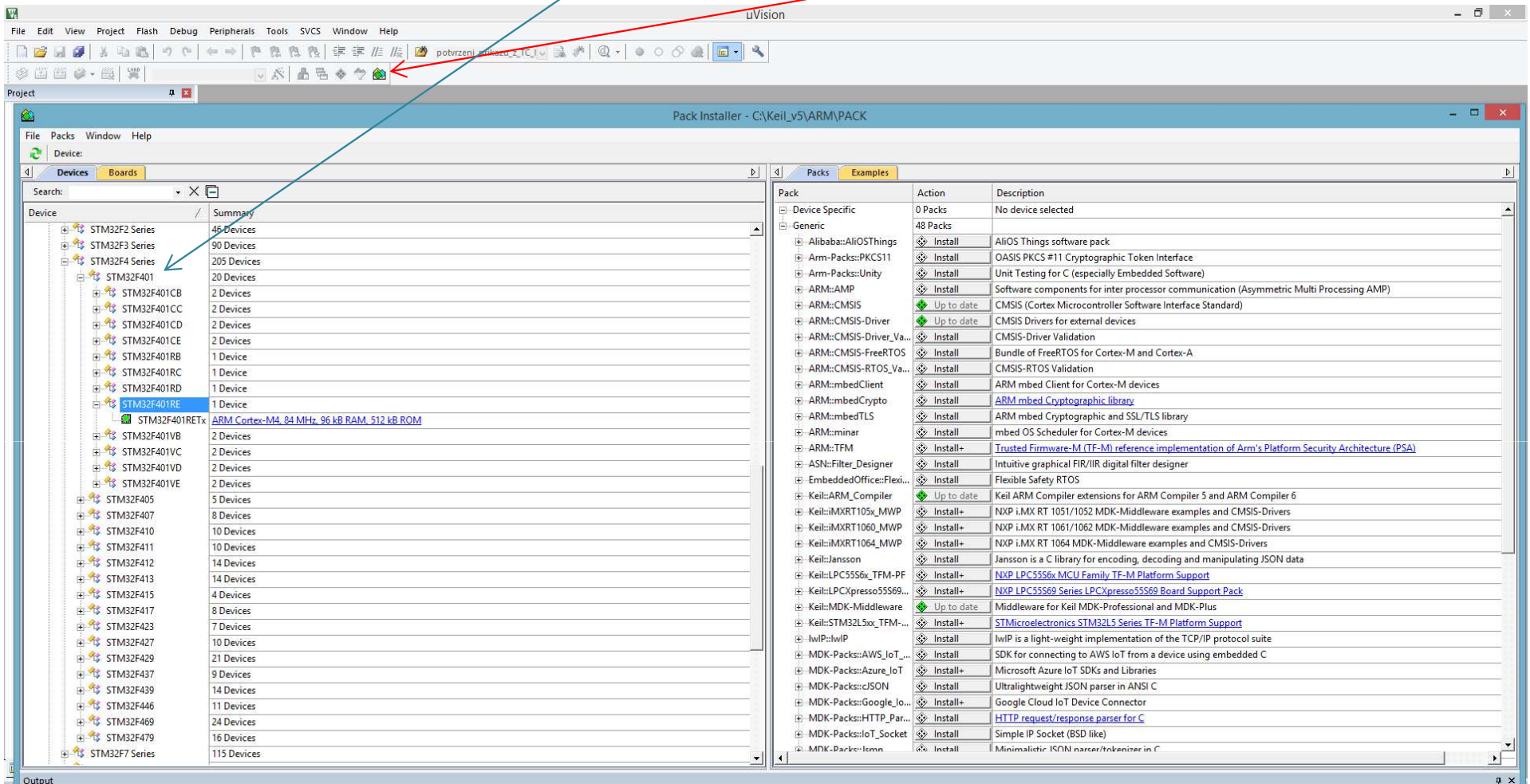


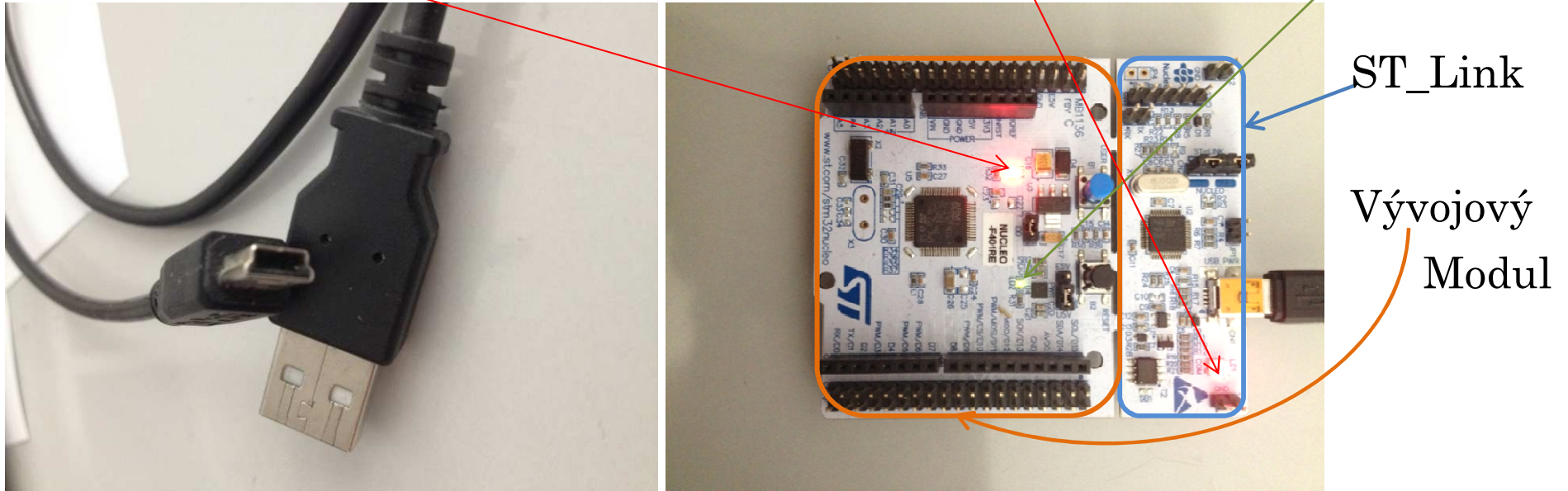
PRVNÍ KROKY K REALIZACI PROGRAMU

- ❖ Nainstalujeme si na PC program Keil uVision5 5.36 nebo vyšší a STM32 ST-LINK Utility V3.8.0, který je k dispozici na Moodle.
- ❖ Po instalaci nebo po prvním spuštění zmáčkne ikonu programu pack. Vybereme ST electronic, STM32F4 a poklepeme na ni. Nahrají se potřebné knihovny.



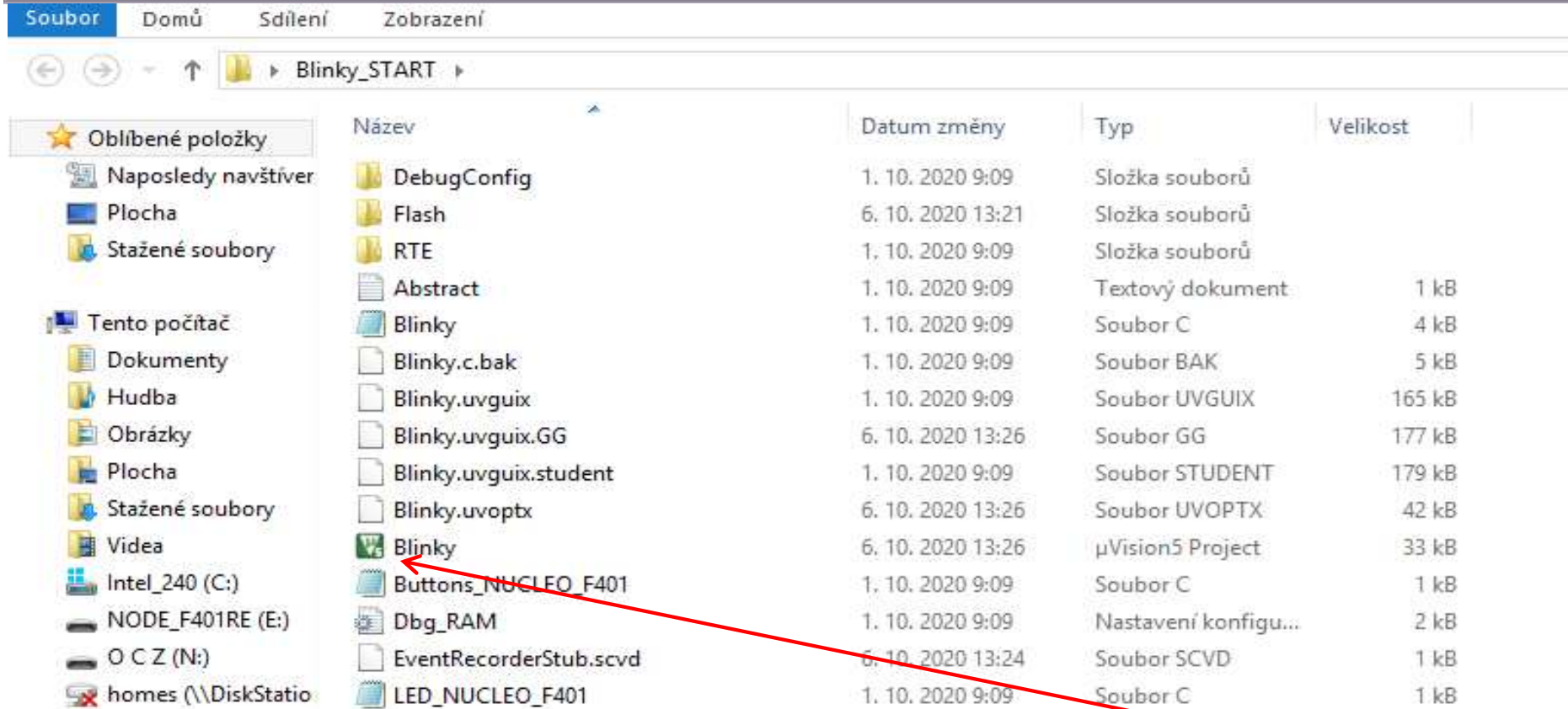
PRVNÍ KROKY K REALIZACI PROGRAMU

- ❖ 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

The screenshot shows the µVision IDE interface. The top menu bar includes File, Edit, View, Project, Flash, Debug, Peripherals, Tools, SVCS, Window, and Help. The toolbar contains various icons for file operations and debugging. The Project Explorer on the left shows a project named 'Blinky' with a sub-folder 'STM32F401 Flash' containing 'Source Files' and 'Documentation'. The 'Source Files' folder contains files like 'Blinky.c', 'Buttons_NUCLEO_F401.c', 'cmsis_armcc.h', 'cmsis_compiler.h', 'cmsis_version.h', 'core_cm4.h', 'LED_NUCLEO_F401.c', 'mpu_armv7.h', 'stdint.h', 'stdio.h', 'stm32f401xe.h', 'stm32f4xx.h', and 'system_stm32f4xx.h'. The 'Documentation' folder contains 'Abstract.txt', 'CMSIS', and 'Device'. The main editor window shows the code for 'Blinky.c', which includes initialization of the PLL, core clock, and LEDs, and a main function that enters an infinite loop checking a button state and turning an LED on. The Build Output window at the bottom shows the compilation process, including warnings and errors, and a red arrow points from the project name 'Blinky' in the Build Output window to the project name 'Blinky' in the Project Explorer.

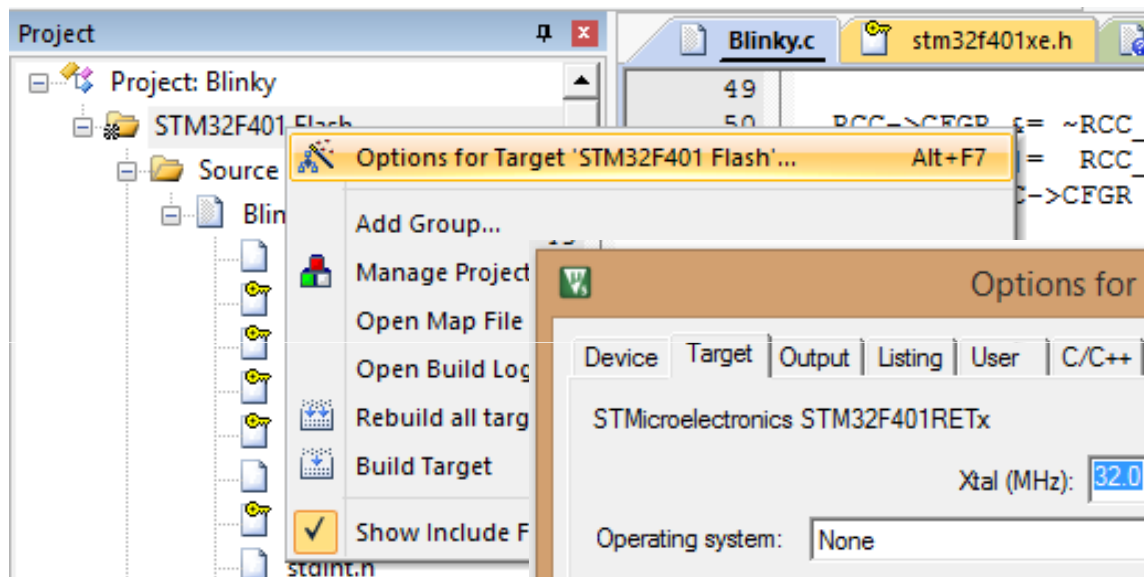
```
49
50 RCC->CFGR &= ~RCC_CFGR_SW; // Select PLL as system clock source
51 RCC->CFGR |= RCC_CFGR_SW_PLL;
52 while ((RCC->CFGR & RCC_CFGR_SWS) != RCC_CFGR_SWS_PLL); // Wait till PLL is system clock src
53 }
54
55 unsigned int output;
56 /*-----*/
57 MAIN function
58 /*-----*/
59 int main (void)
60 {
61     int32_t num = 5;
62     int32_t btns = 0;
63
64     SystemCoreClockSetHSI();
65     SystemCoreClockUpdate(); // Get Core Clock Frequency
66
67     LED_Initialize();
68     Buttons_Initialize();
69
70     while(1) // Nekonecna smycka // Cteni tlacitka
71     {
72         btns = Buttons_GetState();
73         LED_On (num);
74     }
75 }
```

Build Output

```
{ uint32_t pocet, i;
Blinky.c(61): warning: #550-D: variable "btns" was set but never used
int32_t btns = 0;
Blinky.c: 3 warnings, 0 errors
assembling startup_stm32f401xe.s...
compiling system_stm32f4xx.c...
linking...
```

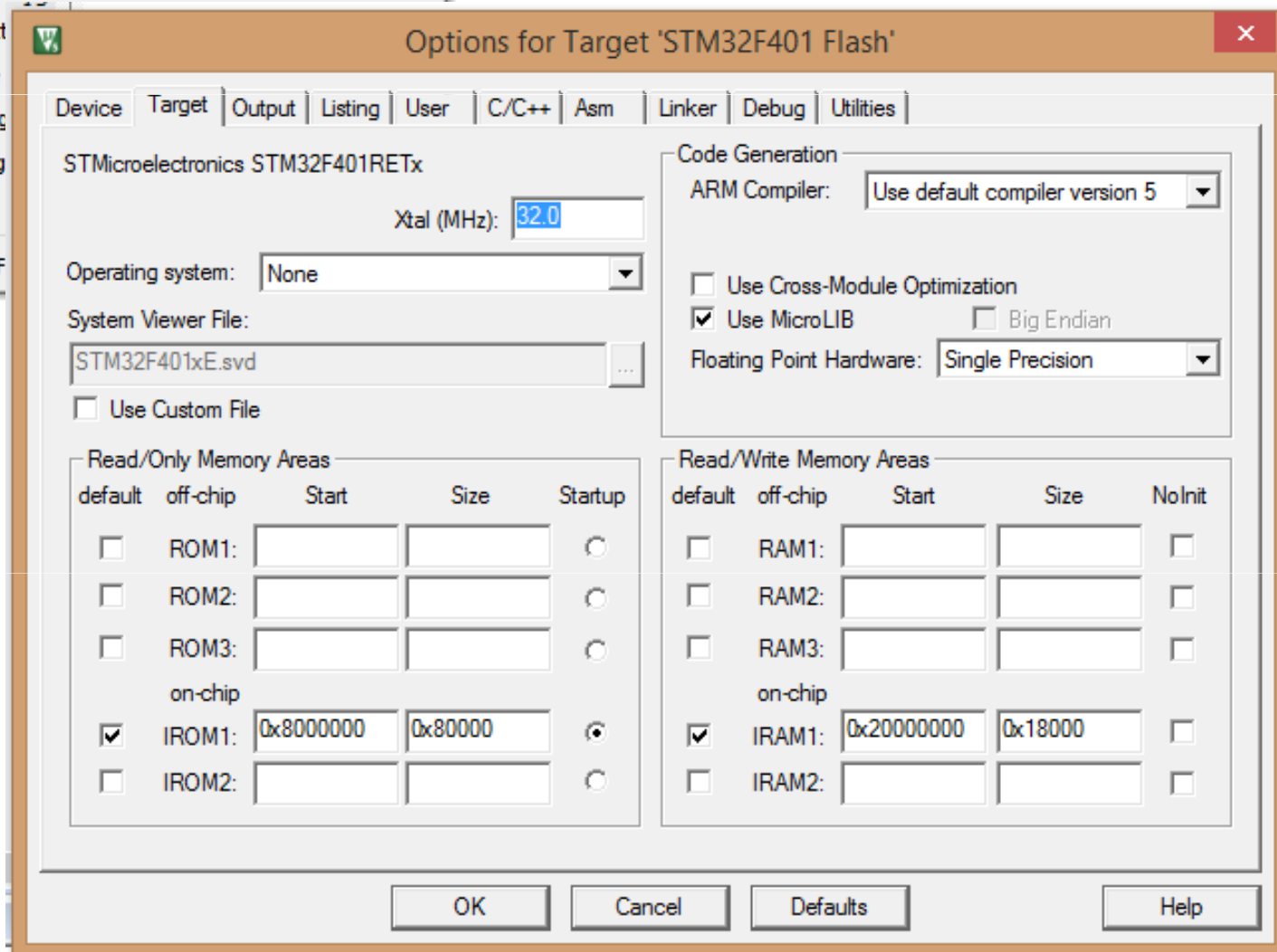
❖ Pravé tlačítko myši na název projektu

NEJDŮLEŽITĚJŠÍ NASTAVENÍ

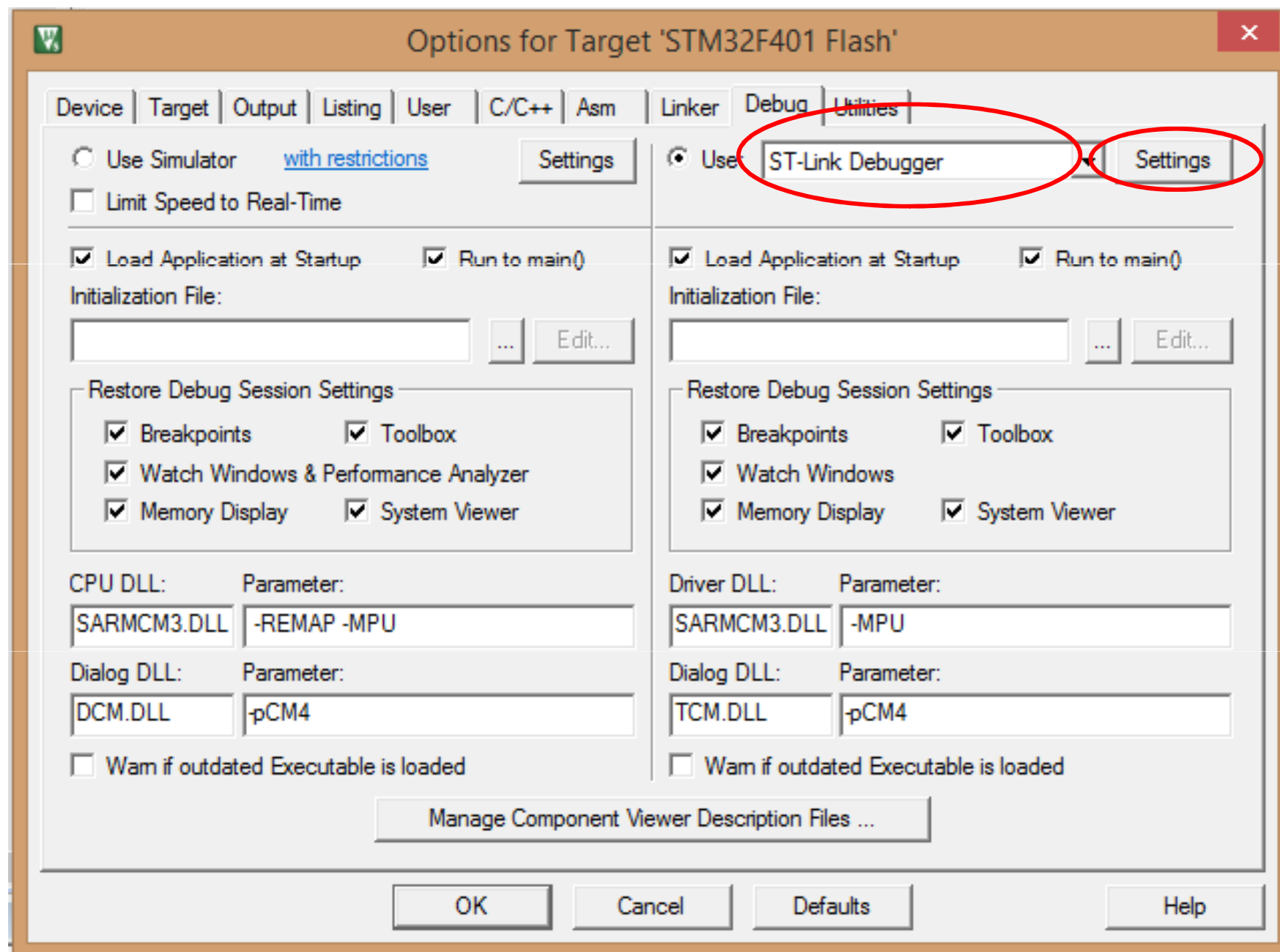


- ❖ Po vybrání položky Option for....
- ❖ Uvidíme okno

- ❖ Device – použitý procesor
- ❖ Target – hodinový kmitočet
- ❖ 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

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:

IR len: AP: 0

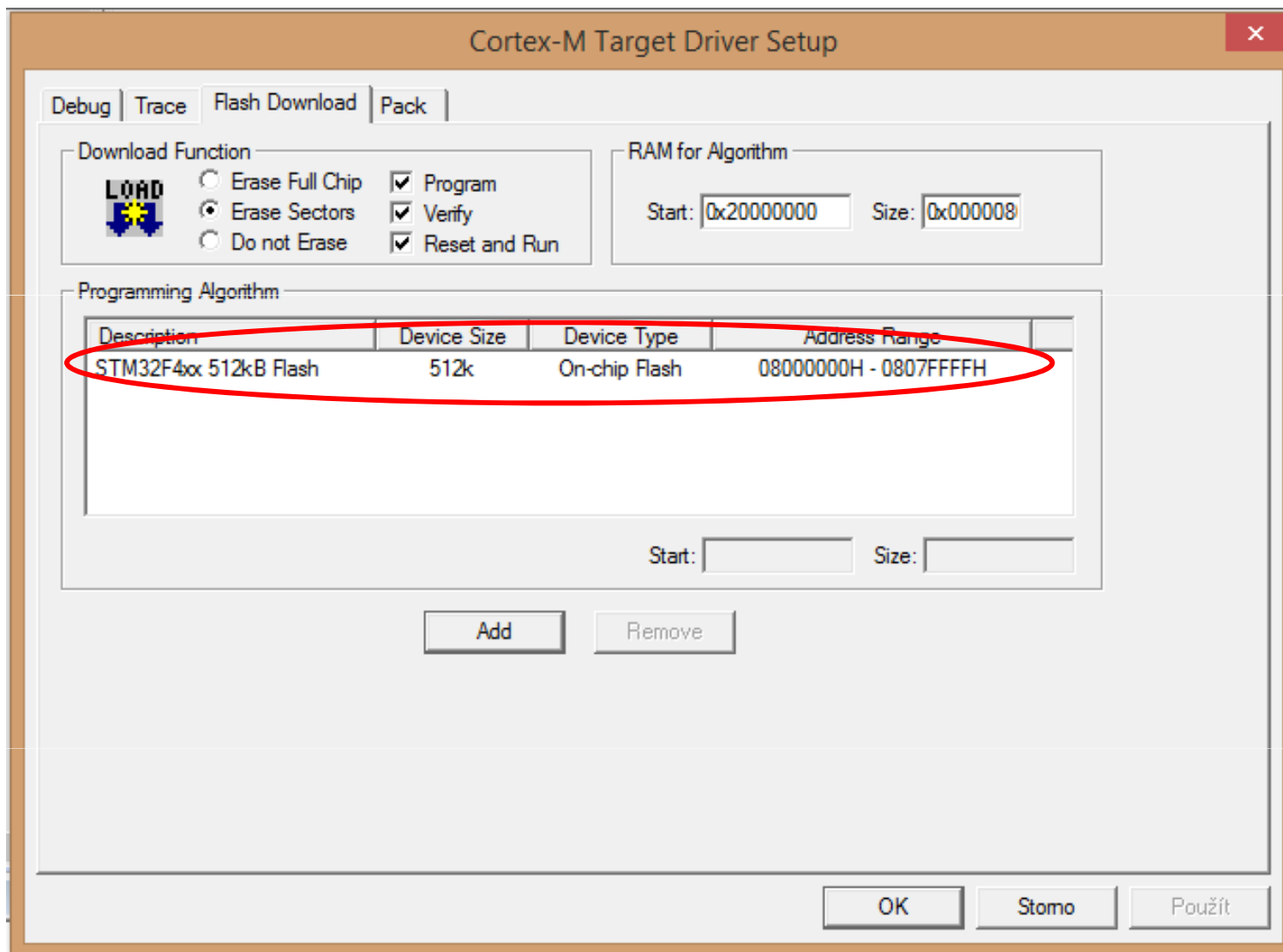
Debug

Connect & Reset Options
Connect: Normal Reset: Autodetect
 Reset after Connect Stop after Reset

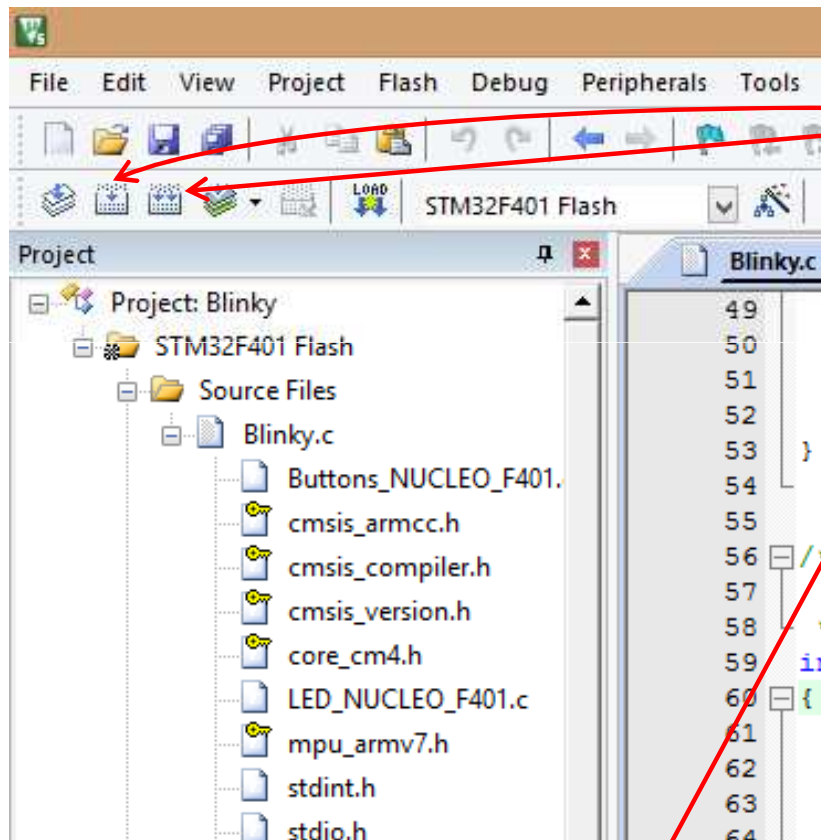
Cache Options
 Cache Code
 Cache Memory

Download Options
 Verify Code Download
 Download to Flash

NEJDŮLEŽITĚJŠÍ NASTAVENÍ



PŘEKLAD PROGRAMU

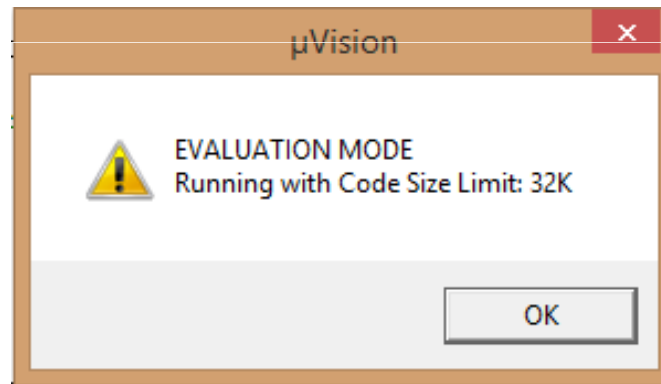


- ❖ Překlad programu zajistíme pomocí ikon
- ❖ Po překladu zkontrolovat zda je bez chyb, varování jsou většinou nevýznamná
- ❖ Code – velikost strojového kódu programu
- ❖ RO-data – velikost konstant v programu
- ❖ RW-data – velikost paměti pro proměnné

```
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 during a debug session. The top menu bar includes File, Edit, View, Project, Flash, Debug, Peripherals, Tools, SVCS, Window, and Help. The main workspace is divided into several panels:

- Registers:** Shows the state of various registers. R15 (PC) is highlighted with a value of 0x08000294.
- Disassembly:** Shows the assembly code being executed. The instruction at address 0x08000294 is highlighted: `MOVS r4, #0x05`.
- Source:** Shows the C source code for `main`. A breakpoint is set at line 60, which is highlighted in green.
- Watch:** Shows a watchpoint for the variable `num` with a value of 0x08000484.
- Memory:** Shows the memory dump starting at address 0x20000000.
- Call Stack + Locals:** Shows the call stack with the `main` function at address 0x00000000. Local variables `num` and `btns` are listed.

The Command window at the bottom shows the execution status: "Running with Code Size Limit: 32K" and "Load 'C:\\Users\\GG\\Desktop\\Blinky_START\\Flash\\Blinky.axf'". The status bar at the bottom indicates the ST-Link Debugger is active, with a time of 0.00016820 sec and other system information.

Spuštění programu - F5, dioda ST Link – bliká a mění barvu. Krokování - F11 nebo F10. Nastavení 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. Soubor MAM_2022-Konfigurace GPIO bran.pptx. 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 u řešeními uvedenými na následujících dvou stranách.
- ❖ Pro další úlohy již nebudeme vypisovat inicializaci I/O vývodu uvedeným způsobem, ale použijeme připravenou knihovnu **Nastaveni_GPIO.zip**. Ta nám umožní inicializovat vývod zápisem na jeden řádek 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 NÁSLEDNĚ 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); }
    // nebo GPIOA->BSRRL |= (1ul << LED);
    return (0);
}

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


KONFIGURACE A ČTENÍ STAVU TLAČÍTKA NA VÝVODU PC13

```
// PODPROGRAM PRO INICIALIZACI A NÁSLEDNĚ 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 následující stránce

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

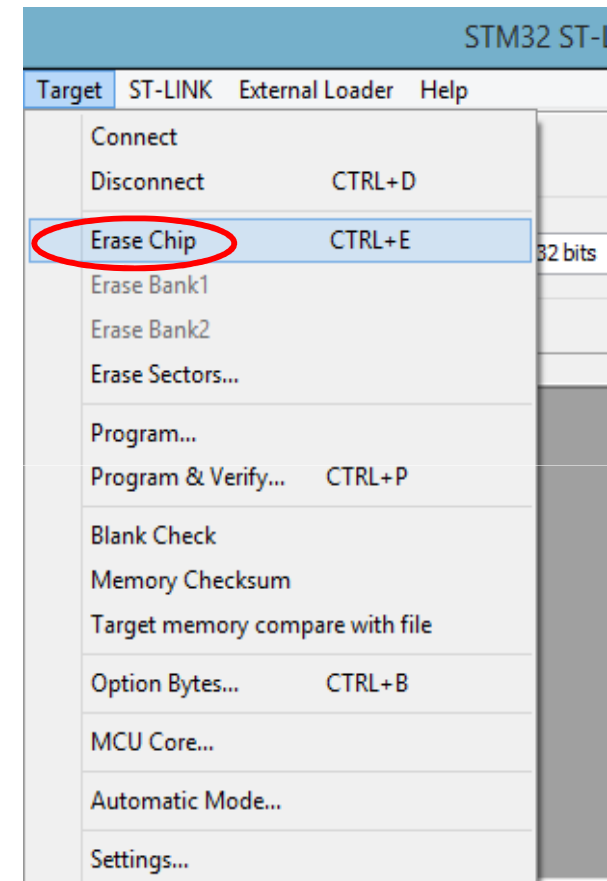
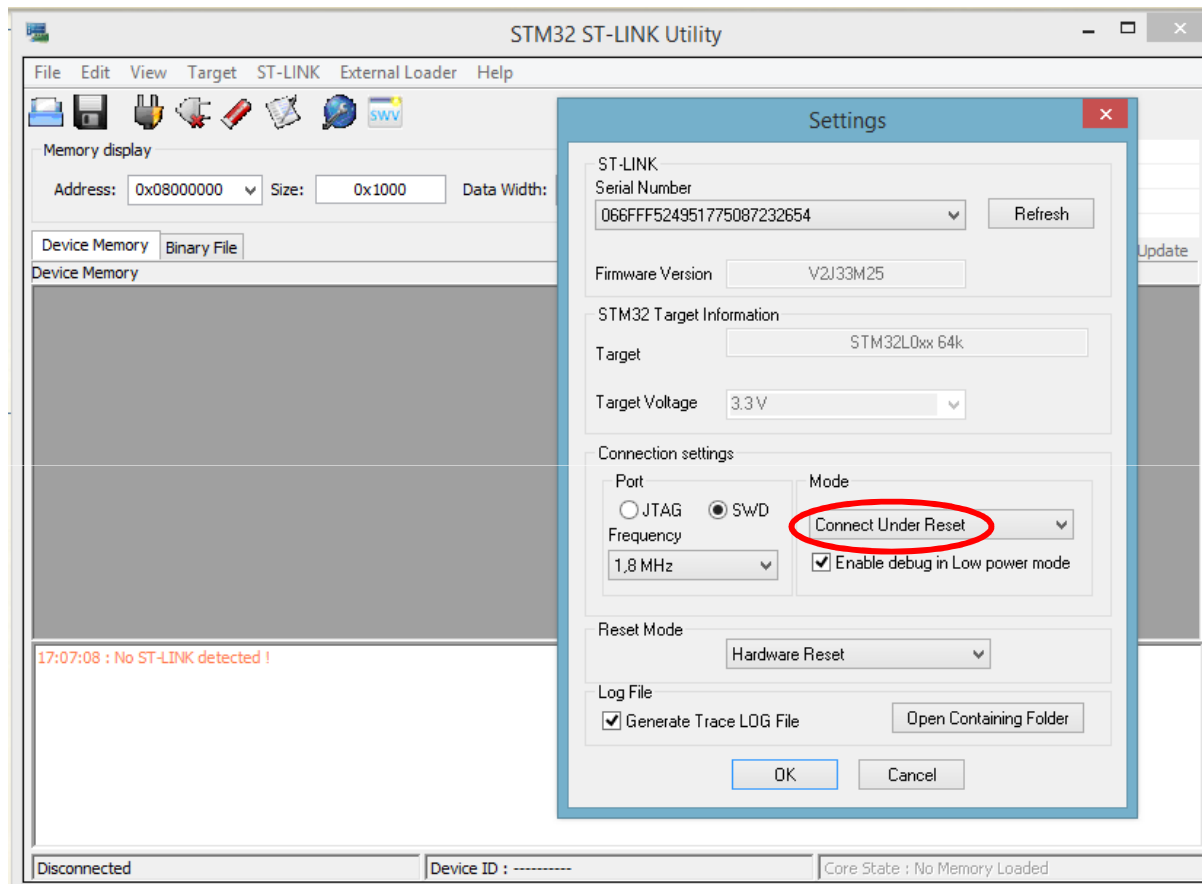
kanál 2 – PA7, PB5, PC7

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:

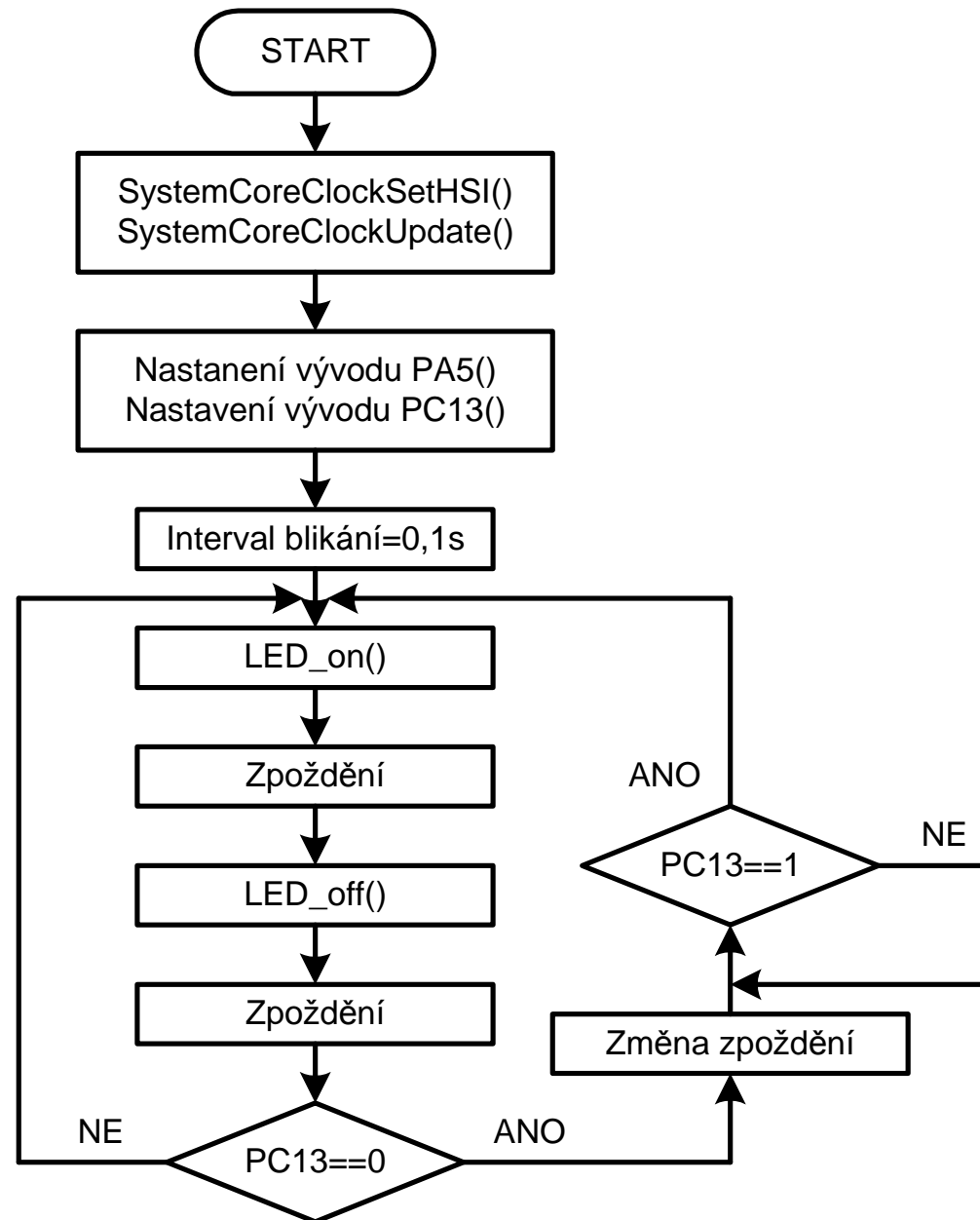
The screenshot shows the STM32 ST-LINK Utility window. The title bar reads "STM32 ST-LINK Utility". The menu bar includes File, Edit, View, Target, ST-LINK, External Loader, and Help. The toolbar contains icons for file operations and connection. The "Memory display" section shows "Address: 0x08000000", "Size: 0x1000", and "Data Width: 32 bits". The "Device Memory @ 0x08000000" section is set to "Binary File" and "LiveUpdate" is disabled. The "Target memory, Address range: [0x08000000 0x08001000]" section contains a table with columns for Address, 0, 4, 8, C, and ASCII. The table shows memory addresses from 0x08000000 to 0x08000B00, all containing 00000000. The bottom status bar shows "Debug in Low Power mode enabled.", "Device ID:0x417", and "Core State : Live Update Disabled".

Address	0	4	8	C	ASCII
0x08000000	00000000	00000000	00000000	00000000
0x08000010	00000000	00000000	00000000	00000000
0x08000020	00000000	00000000	00000000	00000000
0x08000030	00000000	00000000	00000000	00000000
0x08000040	00000000	00000000	00000000	00000000
0x08000050	00000000	00000000	00000000	00000000
0x08000060	00000000	00000000	00000000	00000000
0x08000070	00000000	00000000	00000000	00000000
0x08000080	00000000	00000000	00000000	00000000
0x08000090	00000000	00000000	00000000	00000000
0x080000A0	00000000	00000000	00000000	00000000
0x080000B0	00000000	00000000	00000000	00000000

17:19:50 : Connected via SWD.
17:19:50 : SWD Frequency = 1,8 MHz.
17:19:50 : Connection mode : Connect Under Reset.
17:19:50 : Debug in Low Power mode enabled.
17:19:50 : Device ID:0x417
17:19:50 : Device flash Size : 64KBytes
17:19:50 : Device family :STM32L0xx 64k
17:23:31 : Flash memory erased.
17:23:58 : Flash memory erased.
17:24:22 : Flash memory is blank.

Debug in Low Power mode enabled. Device ID:0x417 Core State : Live Update Disabled

IDEOVÉ ŘEŠENÍ ZÁKLADU ÚLOHY 1



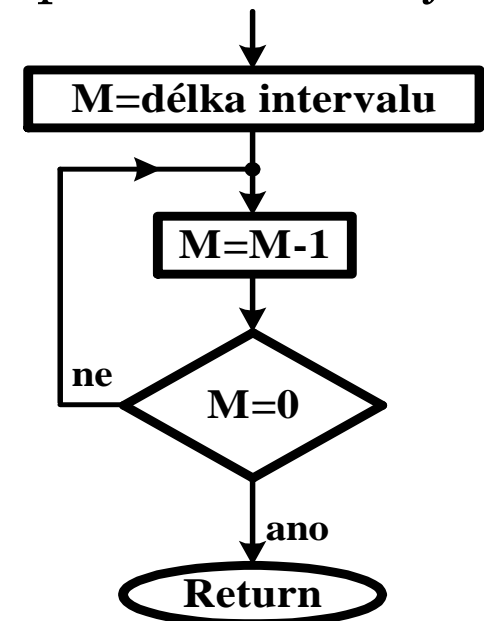
PROGRAMOVĚ GENEROVANÉ ZPOŽDĚNÍ

Vytvoření potřebného zpoždění můžeme realizovat

- Monostabilním obvodem (číslicovo-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 přerušování.



PROGRAMOVĚ GENEROVANÉ ZPOŽDĚNÍ V JAZYCE C A ASEMLERU

```
const uint32_t doba_zpozdeni = 0x22800; // nutno vyzkouset 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 nepřeloží
```