

➤ **Nezbytná literatura pro základní informace**

- **RM0368 - Reference manual_F401.pdf**
- **Cortex-M4_Generic User Guide.pdf**
- **STM32F401RE.pdf**
- **STM32F4xx_Clock_Configuration_V1.1.0.xls**
- **MDK5-getting-started.pdf**

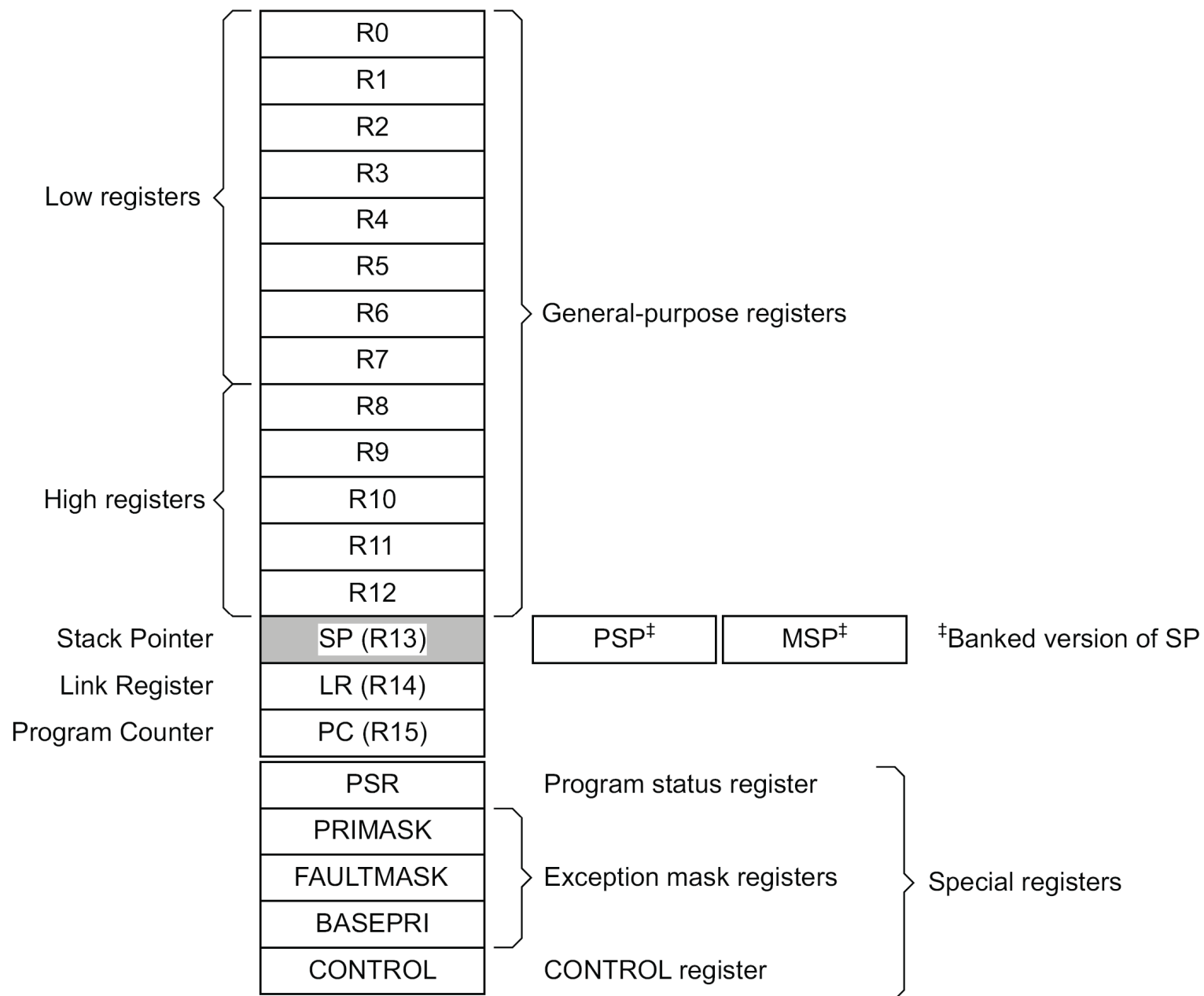
Pro podrobnější studium

- **M4_Technical Reference Manual.pdf**
- **Timers_STM32F4.pdf** nebo **Timers_STM32L4.pdf**
- **Clock configuration_Application note_AN3988.pdf**
- **STM32F4 HAL and LL drivers_Manual_UM1725.pdf**

Pro práci v JSA (Assembleru)

- **Cortex-M3_M4F Instruction Set.pdf**
- **DUI0473C_Using_the_ARM_Assembler_0.pdf**
- **DUI0489C_Arm_Assembler_Reference.pdf**

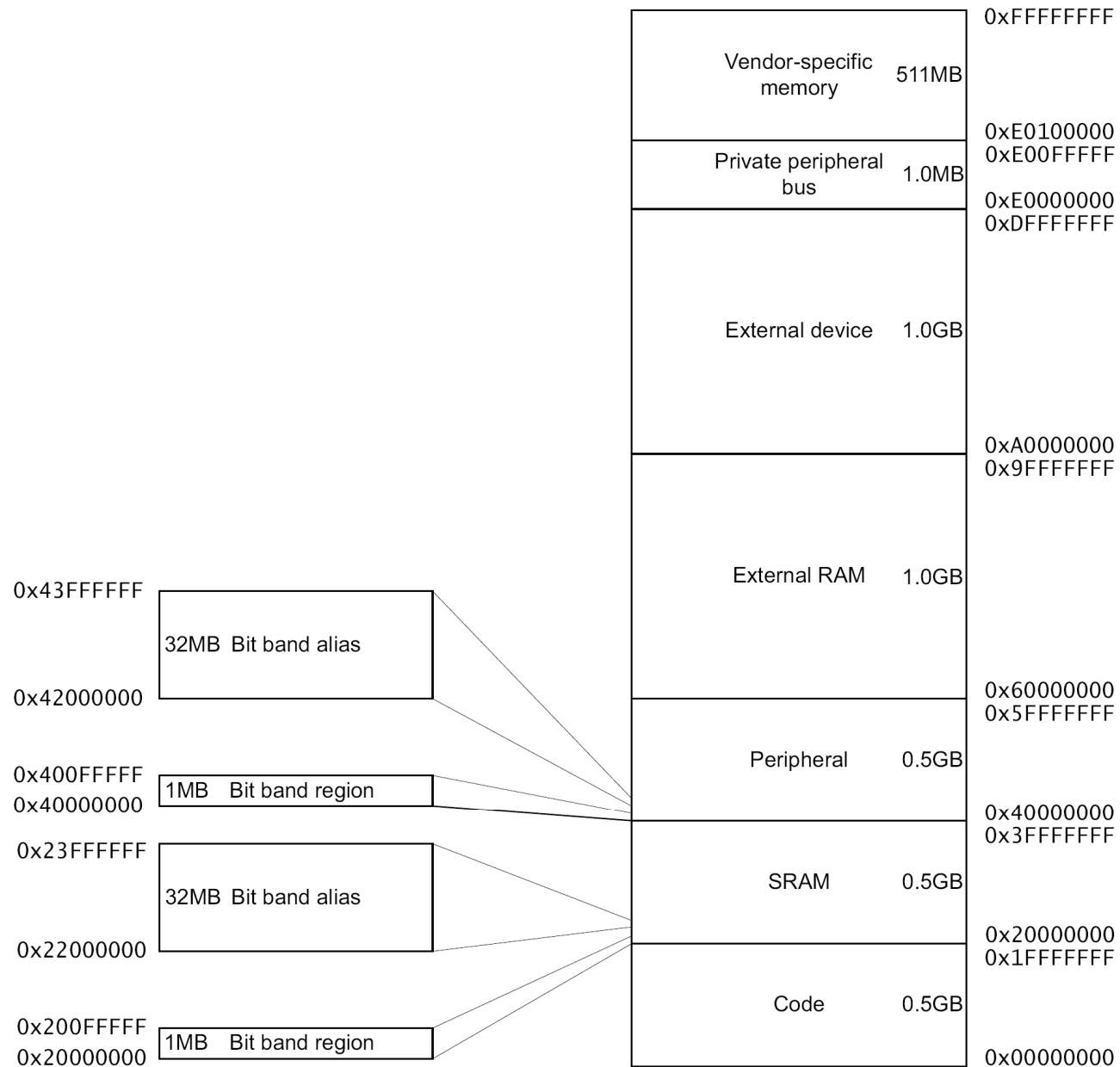
REGISTRY ARM M3 A M4 A JEJICH FUNKCE



REGISTRY NA PROCESORECH ARM

- **R0–R12: všeobecně použitelné registry.** Některé 16-ti bitové instrukce Thumb® mají přístup pouze ke registrům R0 až R7.
- **R13: Ukazatel zásobníku (stack pointer)** – dva ukazatele
 - **MSP** defaultní - operačním systémem a obsluha přerušení.
 - **PSP** používaný v aplikacích. Oba zarovnány mod4 (.align 4) na dvě slova nebo adresu.
- **R14: Link Register.** - jednoúrovňový zásobník
- **R15: Program Counter** - adresa čteného bytu programu (instrukce). Oproti jiným procesorům je přístupný.
- **Speciální registry** - mají speciální funkci ovlivněnou speciálními instrukcemi.
 - **xPSR** – Program status registr - **příznaky**
 - **PRIMASK, FAULTMASK a BASEPRI** - Interrupt Mask registers,
 - **CONTROL** – řídicí registr.

ROZLOŽENÍ ADRESOVÉHO PROSTORU ARM M3 A M4



PŘÍZNAKY CORTEX M3 A M4

xPSR (Program Status Register) – je rozdělený na tři stavové registry:

- ❖ Stavový registr aplikačního programu (APSR)
- ❖ Stavový registr přerušeni (IPSR)
- ❖ Stavový registr prováděného programu (EPSR)

Stavové registry mohou být přístupné společné nebo odděleně pomocí instrukcí MSR a MRS. Při společném přístupu se používá označení xPSR. Registry EPSR a IPSR mohou být pouze čteny (MRS), registr APSR může být měněn instrukcí MSR.

	31	30	29	28	27	26:25	24	23:20	19:16	15:10	9	8	7	6	5	4:0
APSR	N	Z	C	V	Q											
IPSR												Exception number				
EPSR						ICI/IT	T				ICI/IT					

Registr stavových příznaků xPSR má 32 bitů, z nichž 5 nejvyšších bitů je pro začátek nejdůležitějších. Umístění příznaků v xPSR je zobrazeno na obrázku a jejich význam je následující:

	31	30	29	28	27	26:25	24	23:20	19:16	15:10	9	8	7	6	5	4:0	
xPSR	N	Z	C	V	Q	ICI/IT	T				ICI/IT		Exception number				

N – Negativní nebo menší než je nastaven jestliže po instrukci je do bitu b31 přenesena hodnota 1.

PŘÍZNAKY CORTEX M3 A M4

Z – Nula – nastaven aritmetickou nebo instrukcí porovnání.

C – Carry/Borrow - nastaven při přenosu z bitu b31 do bitu b32 nebo výpůjčce. Pro instrukce **ADC**, **ADD** a **CMN** je produktem přetečení. Pro **CMP**, **SBC** a **SUB**, je bit nastaven při výpůjčce. Při instrukcích posunu se do něj přenáší poslední bit.

V - Příznak přetečení (Overflow) indikuje přetečení při aritmetické operaci s čísly se znaménkem. Kdy součet záporných čísel je kladný nebo součet kladných čísel je záporný.

Q – Indikace saturace je nastaven, dojde-li při aritmetice se saturací k dosažení maximální nebo minimální hodnoty.

ICI/IT – slouží ke správnému vykonání instrukce IF-THEN, při které dojde k přerušení

T – Thumb mód=log.1

Exception number – indikuje, které přerušení procesor zpracovává

ZÁKLADNÍ ROZDÍLY CORTEX M3, M4 PROTI 8051, AVR atd.

V jazyce C i assembleru musíme před programem main:

- ❖ Vložit soubor s adresami symbolických názvů registrů procesoru např. `system_STM32F4xx.c`
- ❖ Vložit soubor se zdroji přerušení, nastavením ukazatele SP a programem se skokem na main - `startup_STM32F4xx.s`
- ❖ **Nastavit konfigurační bity μ P – Boot, zdroj hodin, WD, JTAG**
- V MAIN je nejdříve potřeba:
 - ❖ Případně nastavit nový hodinový kmitočet procesoru
 - ❖ Nastavit hodinový kmitočet pro sběrnice APB1 a APB2, ke kterým jsou připojeny Vámi používané periferie.
 - ❖ **Nakonfigurovat používané vstupně/výstupní vodiče 16 bitových bran GPIOA, GPIOB, až GPIOK, atd. podle možností procesoru.**
- **Nastavit ostatní periferie, povolit přerušovací systém, atd.**
- **Vytvářet požadovaný program**

Udělat pro procesor ARM ♣, ♠ a ♡ (všechno), AVR ♣, ♠ a ♡, 8051 jen ♡

Start procesoru – uvedení do definovaného počátečního stavu, který zjišťuje nulování μP (tzv. Reset).

Reset může být vyvolán

- Log.0 na vývodu NRST – externí reset
- Oknovým watchdogem WWDG při podtečení
- Nezávislým watchdogem IWDG dosažením konce čítání
- Programovým nulováním SW reset
- Obvodem pro kontrolu napájecího napětí a spotřeby
 - Nastavením **Standby mode** při nRST_STDBY=0
 - Nastavením **Stop mode** při nRST_STOP=0
 - Aktivací obvodu POR/PDR nebo BOR (prahové úrovně)

Konfiguraci hodinového signálu a jeho přivedení k potřebným jednotkám zajišťuje skupina registrů

RCC (Reset and Clock Control)

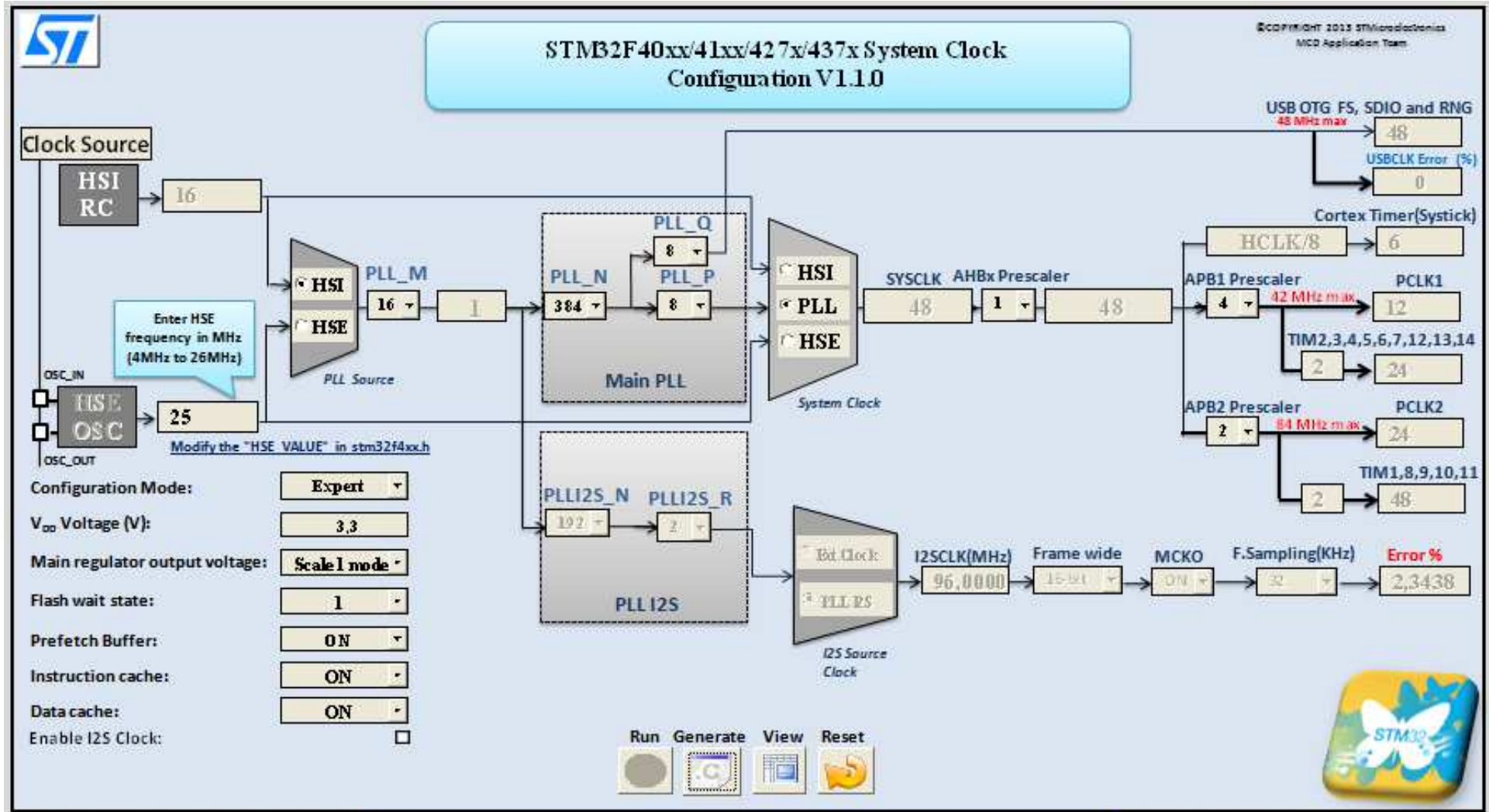
Hodinový signál – se konfiguruje nejen pro samotný procesor a přenos po sběrnících, ale ke každé periférii, kterou budeme v aplikaci potřebovat (rozdíl vůči 8051, AVR, HC11, atd.).

Zdroje hodinového signálu:

- **HSI** RC oscilátor – defaultní kompenzovaný interní zdroj
- **HSE** interní oscilátor
 - S externím krystalovým/keramickým rezonátorem
 - S externím hodinovým signálem
- **PLL** fázový závěs řízený HSI nebo HSE
- Sekundární zdroje hodin
 - 32 kHz low-speed internal RC (LSI RC) pro WDT
 - 32.768 kHz low-speed external crystal (LSE crystal) pro RTC.

Konfiguraci umožňují **RCC Registry**

KONFIGURACE HODINOVÉHO SYSTÉMU PROCESORU F401



KONFIGURACE HODINOVÉHO SYSTÉMU PROCESORU F401

```
//      SystemCoreClockConfigure: configure SystemCoreClock using HSI (HSE is not populated on
Nucleo board)

void SystemCoreClockConfigure(void)
{
    //RCC->CR |= ((uint32_t)RCC_CR_HSION);           // Enable HSI
    while ((RCC->CR & RCC_CR_HSIRDY) == 0);        // Wait for HSI Ready
    RCC->CFGR = RCC_CFGR_SW_HSI;                   // HSI is system clock
    while ((RCC->CFGR&RCC_CFGR_SWS)!=RCC_CFGR_SWS_HSI); // Wait for HSI used as system clock

    FLASH->ACR = FLASH_ACR_PRFTEN;                // Enable Prefetch Buffer
    FLASH->ACR |= FLASH_ACR_ICEN;                  // Instruction cache enable
    FLASH->ACR |= FLASH_ACR_DCEN;                  // Data cache enable
    FLASH->ACR |= FLASH_ACR_LATENCY_5WS;          // Flash 5 wait state

    RCC->CFGR |= RCC_CFGR_HPRE_DIV1;               // HCLK(sběrnice AHB) = SYSCLK
    RCC->CFGR |= RCC_CFGR_PPRE1_DIV4;              // APB1 = HCLK/4    MAX 42MHz
    RCC->CFGR |= RCC_CFGR_PPRE2_DIV2;              // APB2 = HCLK/2    MAX 84MHz

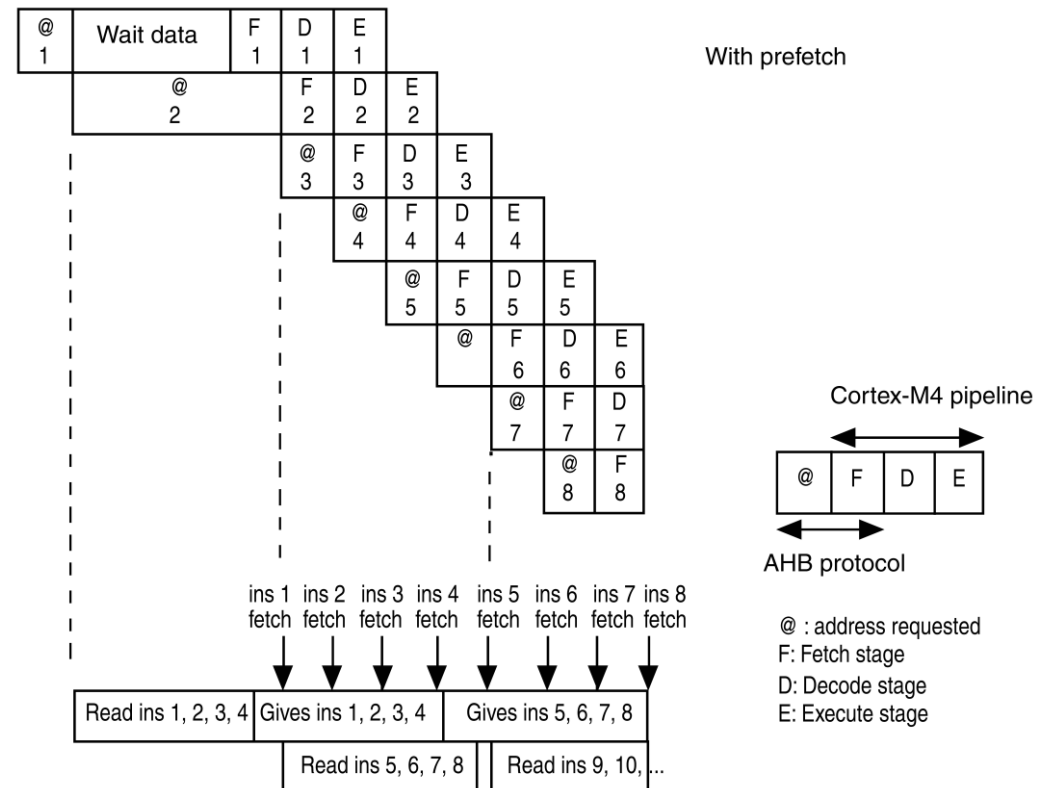
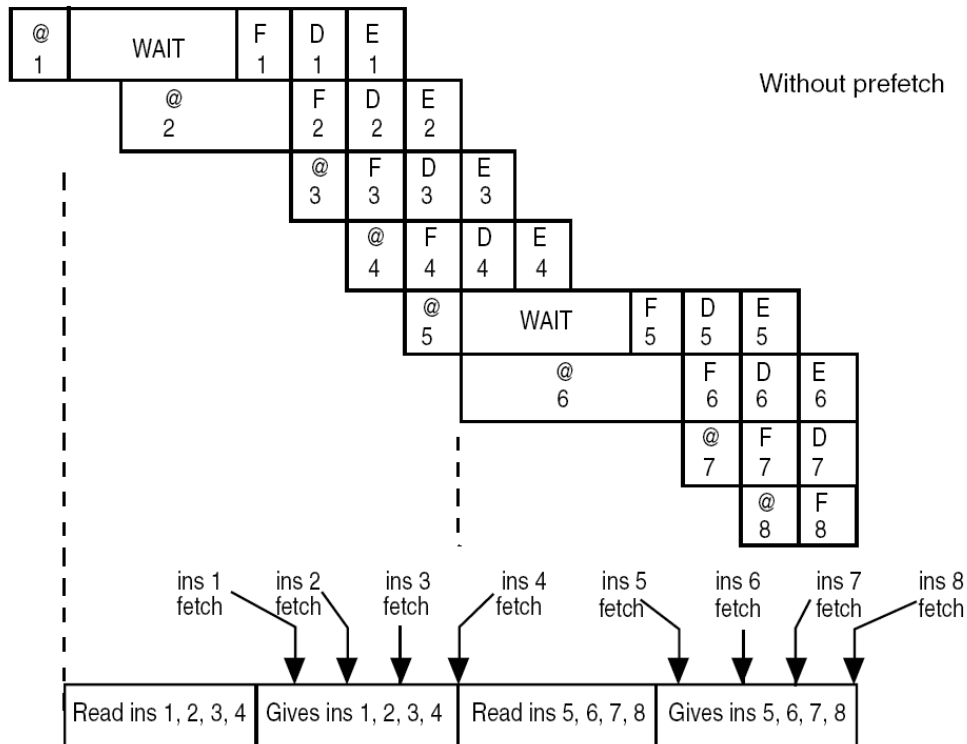
    RCC->CR &= ~RCC_CR_PLLON;                       // Disable PLL
    //      PLL configuration: VCO = HSI/M * N, Sysclk = VCO/P
    RCC->PLLCFGR = ( 16ul |                           // PLL_M = 16
                   (384ul << 6) |                    // PLL_N = 384
                   ( 3ul << 16) |                    // PLL_P = 8
                   (RCC_PLLCFGR_PLLSRC_HSI) |        // PLL_SRC = HSI
                   ( 8ul << 24) );                   // PLL_Q = 8

    RCC->CR |= RCC_CR_PLLON;                         // Enable PLL
    while((RCC->CR & RCC_CR_PLLRDY) == 0) __NOP();   // Wait till PLL is ready
    RCC->CFGR &= ~RCC_CFGR_SW;                       // Select PLL as system clock source
    RCC->CFGR |= RCC_CFGR_SW_PLL;
    while ((RCC->CFGR & RCC_CFGR_SWS) != RCC_CFGR_SWS_PLL); // Wait till PLL is system
                                                                // clock src
}
}
```

SYSTEM ČTENÍ PROGRAMU Z PAMĚTI FLASH U PROCESORŮ F4

```

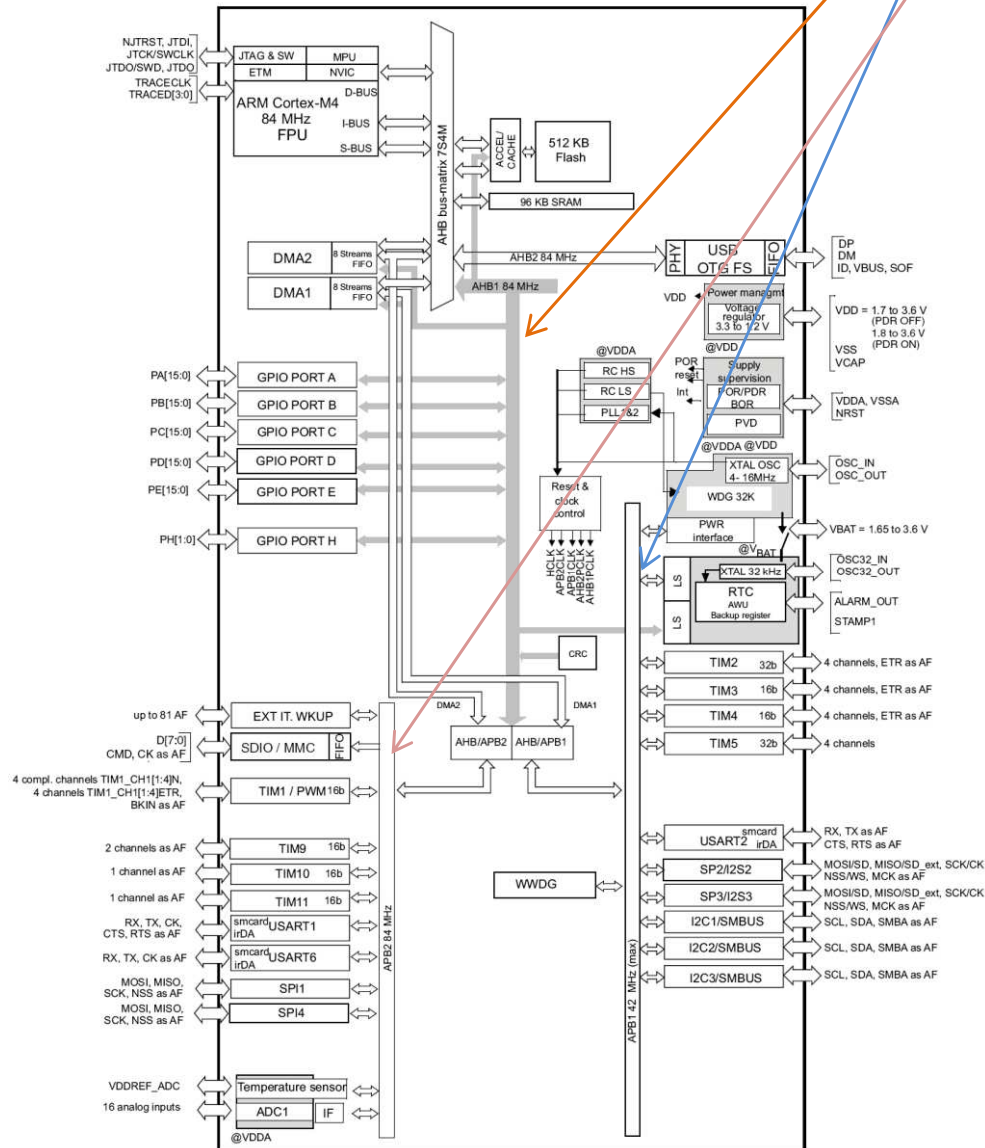
FLASH->ACR = FLASH_ACR_PRFTEN; // Enable Prefetch Buffer
FLASH->ACR |= FLASH_ACR_ICEN; // Instruction cache enable
FLASH->ACR |= FLASH_ACR_DCEN; // Data cache enable
FLASH->ACR |= FLASH_ACR_LATENCY_5WS; // Flash 5 wait state
    
```



KONFIGURACE HODINOVÉHO SIGNÁLU PRO SBĚRNICE

```

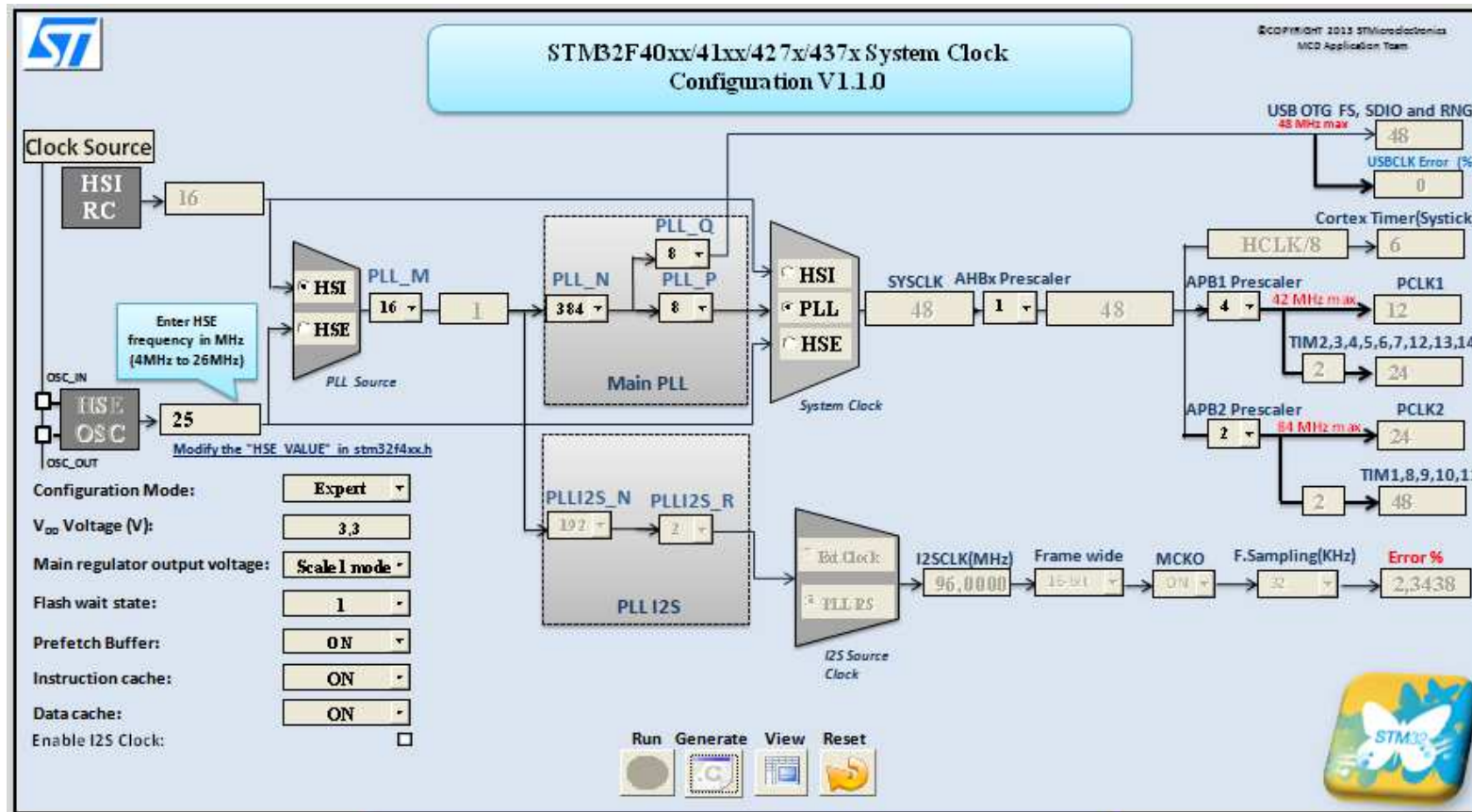
RCC->CFGR | = RCC_CFGR_HPRE_DIV1; // HCLK(sběrnice AHB) = SYSCLK
RCC->CFGR | = RCC_CFGR_PPRE1_DIV4; // APB1 = HCLK/4    MAX 42MHz
RCC->CFGR | = RCC_CFGR_PPRE2_DIV2; // APB2 = HCLK/2    MAX 84MHz
    
```



KONFIGURACE PLL HODINOVÉHO SYSTÉMU PROCESORU F401

```

RCC->CR &= ~RCC_CR_PLLON; // Disable PLL
// PLL configuration: VCO = HSI/M * N, Sysclk = VCO/P
RCC->PLLCFGR = (16ul|(384ul<<6)|(3ul<<16)| // PLL_M=16, PLL_N=384, PLL_P=8
                (RCC_PLLCFGR_PLLSRC_HSI)|(8ul<<24)); // PLL_SRC=HSI, PLL_Q=8
RCC->CR |= RCC_CR_PLLON; // Enable PLL
while((RCC->CR & RCC_CR_PLLRDY) == 0) __NOP(); // Wait till PLL is ready
RCC->CFGR &= ~RCC_CFGR_SW; // Select PLL as system clock
RCC->CFGR |= RCC_CFGR_SW_PLL;
while ((RCC->CFGR & RCC_CFGR_SWS) != RCC_CFGR_SWS_PLL); // Wait till PLL is systém clock src
    
```



KONFIGURACE BITŮ V REGISTRECH

Nastavování bitů v konfiguračních registrech

```
#define setbit(reg, bit)    ((reg) |= (1U << (bit)))  
#define clearbit(reg, bit) ((reg) &= (~(1U << (bit))))  
#define togglebit(reg, bit) ((reg) ^= (1U << (bit)))  
#define getbit(reg, bit)  (((reg) & (1U << (bit))) >> (bit))
```

Jednoduchý způsob \Rightarrow nutné zjišťování polohy bitu v registru, nutná kontrola při přechodu na jiný procesor ARM. Pro každý procesor existuje soubor (zde NUCLEO401)

stm32f401xe.h

obsahující symbolické názvy registrů a jeho bitů. Řada registrů je koncipována jako **struktury** zajišťující odstup adres jednotlivých registrů. Při použití symbolických názvů ARM ST elektronik nemusíme **ověřovat** umístění jednotlivých bitů v konfiguračních registrech při přechodu na jiný procesor.

KONFIGURACE VÝVODŮ S POMOCÍ STM32F401XE.H

Konstrukce přístupu k registrům jednotlivých bran procesoru

```
#define PERIPH_BASE          0x40000000U /*!< Peripheral base address in the alias region
#define AHB1PERIPH_BASE    (PERIPH_BASE + 0x00020000U)
#define GPIOA_BASE        (AHB1PERIPH_BASE + 0x0000U)
#define GPIOB_BASE        (AHB1PERIPH_BASE + 0x0400U)
// atd.
#define GPIOA                ((GPIO_TypeDef *) GPIOA_BASE)

typedef struct
{
  __IO uint32_t MODER;      /*!< GPIO port mode register,           Address offset: 0x00 */
  __IO uint32_t OTYPER;    /*!< GPIO port output type register,      Address offset: 0x04 */
  __IO uint32_t OSPEEDR;   /*!< GPIO port output speed register,     Address offset: 0x08 */
  __IO uint32_t PUPDR;     /*!< GPIO port pull-up/pull-down register, Address offset: 0x0C */
  __IO uint32_t IDR;       /*!< GPIO port input data register,       Address offset: 0x10 */
  __IO uint32_t ODR;       /*!< GPIO port output data register,      Address offset: 0x14 */
  __IO uint32_t BSRR;      /*!< GPIO port bit set/reset register,    Address offset: 0x18 */
  __IO uint32_t LCKR;      /*!< GPIO port configuration lock register, Address offset: 0x1C */
  __IO uint32_t AFR[2];    /*!< GPIO alternate function registers,   Address offset: 0x20-0x24 */
} GPIO_TypeDef;
```

KONFIGURACE VÝVODŮ S POMOCÍ STM32F401XE.H

Dál následuje označení jednotlivých bitů registrů např. **MODER** 2 bit

```
#define GPIO_MODER_MODE1_Pos    (2U)
#define GPIO_MODER_MODE1_Msk    (0x3U << GPIO_MODER_MODE1_Pos) /*!< 0x0000000C */
#define GPIO_MODER_MODE1
#define GPIO_MODER_MODE1_0      (0x1U << GPIO_MODER_MODE1_Pos) /*!< 0x00000004 */
#define GPIO_MODER_MODE1_1      (0x2U << GPIO_MODER_MODE1_Pos) /*!< 0x00000008 */
```

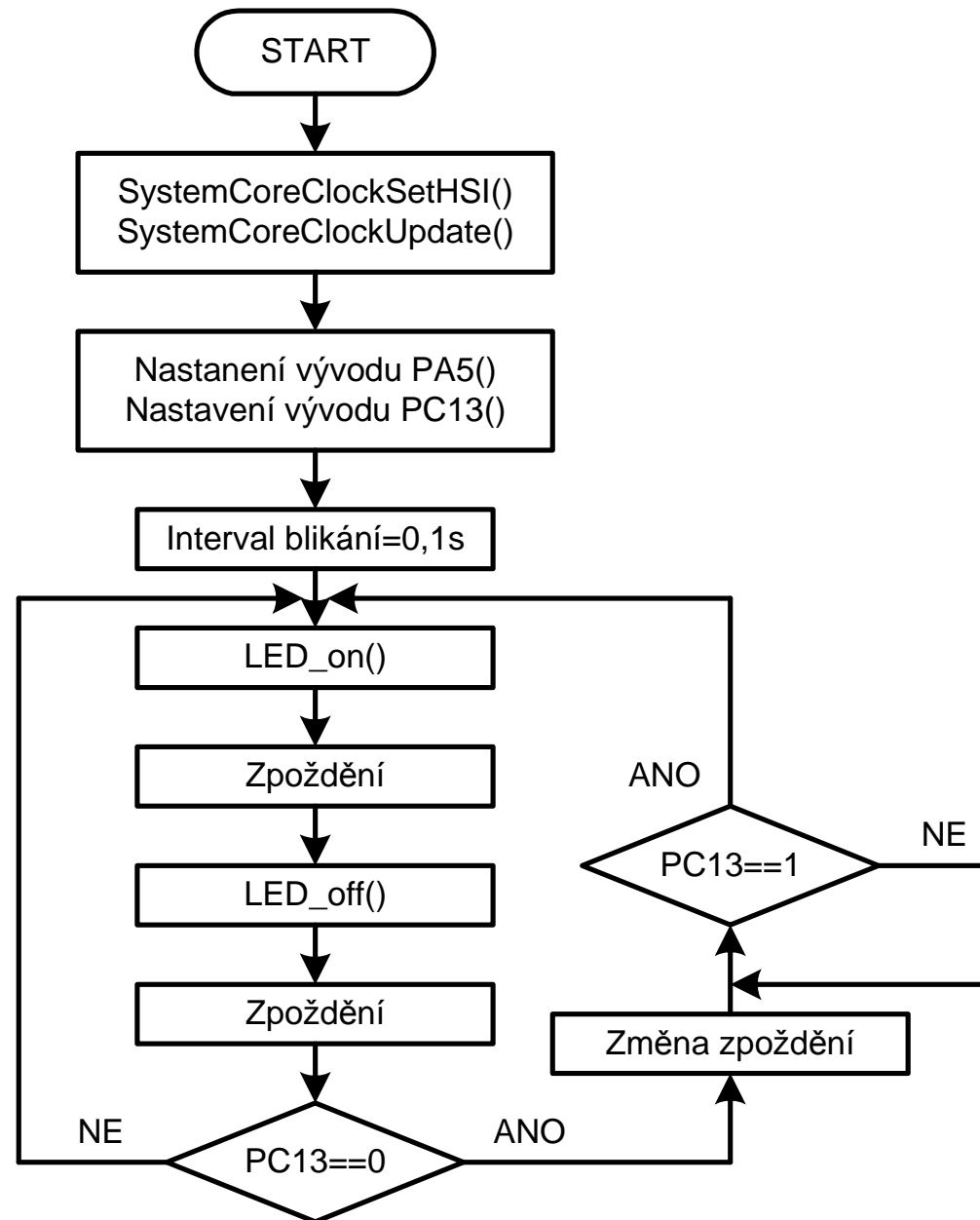
Registr **IDR** 3 bit

```
#define GPIO_IDR_ID2_Pos        (2U)
#define GPIO_IDR_ID2_Msk        (0x1U << GPIO_IDR_ID2_Pos) /*!< 0x00000004 */
#define GPIO_IDR_ID2
#define GPIO_IDR_IDR_2          GPIO_IDR_ID2
```

Registr **ODR** 2 bit

```
#define GPIO_ODR_OD1_Pos        (1U)
#define GPIO_ODR_OD1_Msk        (0x1U << GPIO_ODR_OD1_Pos) /*!< 0x00000002 */
#define GPIO_ODR_OD1
#define GPIO_ODR_ODR_1          GPIO_ODR_OD1
```

IDEOVÉ ŘEŠENÍ ZÁKLADU ÚLOHY 1



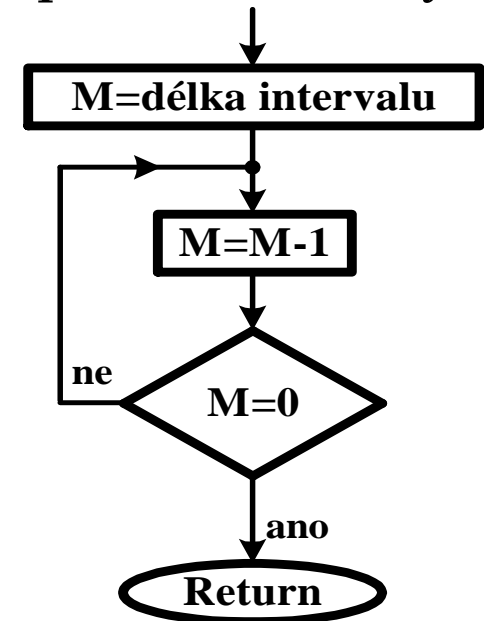
PROGRAMOVĚ GENEROVANÉ ZPOŽDĚNÍ

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

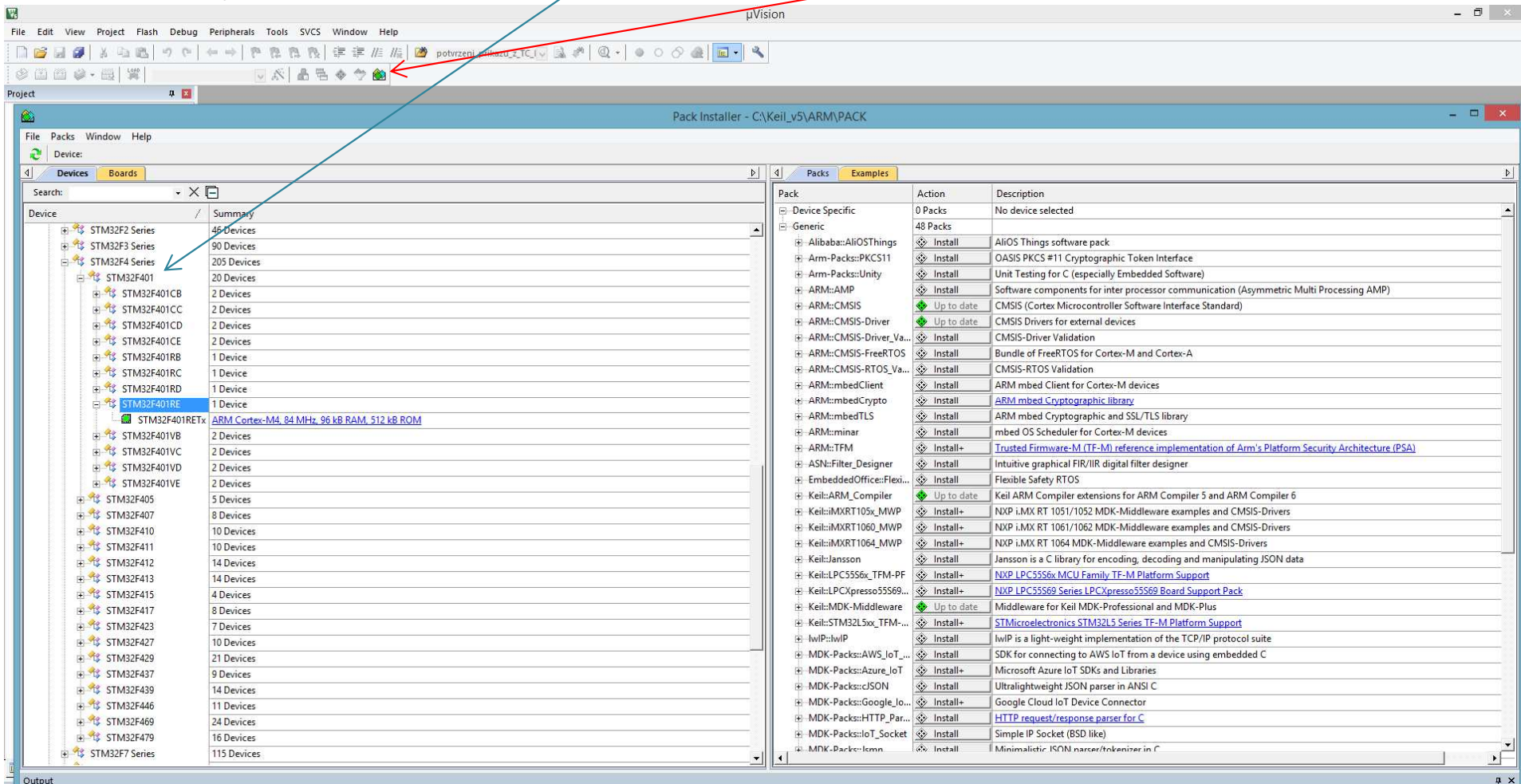
```
const uint32_t doba_zpozdeni = 0x22800; // nutno vyzkoušet dobu trvání. Změna hodnoty
// může způsobit v překladu použití jiné instrukce
// a díky tomu dojde ke skokové změně zpoždění.

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 nepřeloží
```

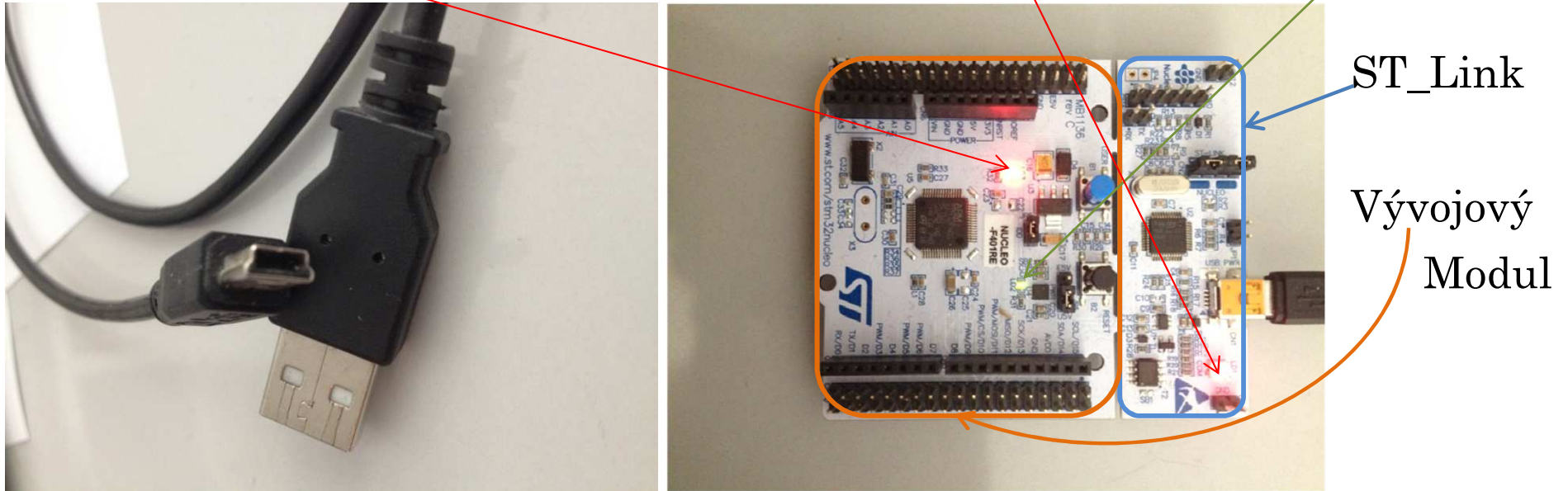
PRVNÍ KROKY K REALIZACI PROGRAMU

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



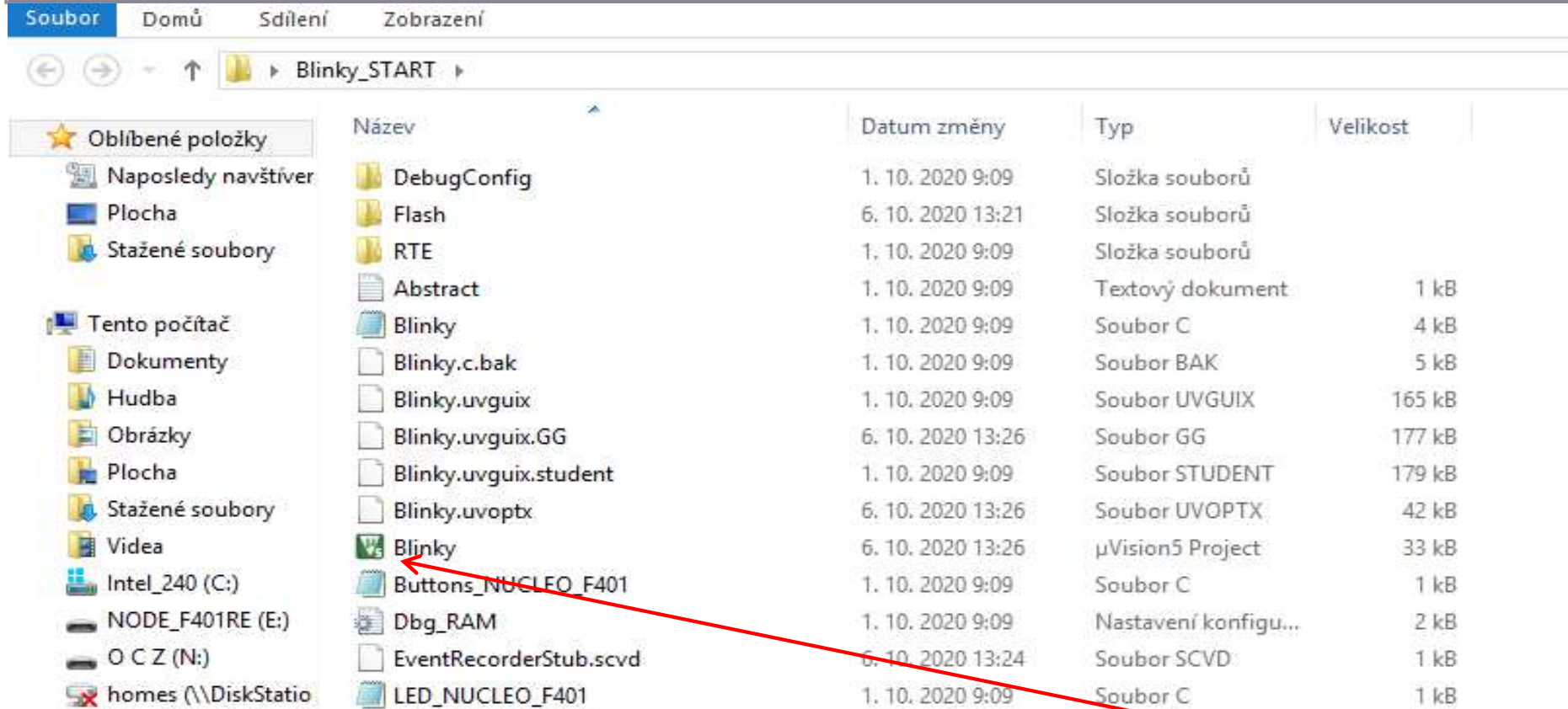
PRVNÍ KROKY K REALIZACI PROGRAMU

- ❖ 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.



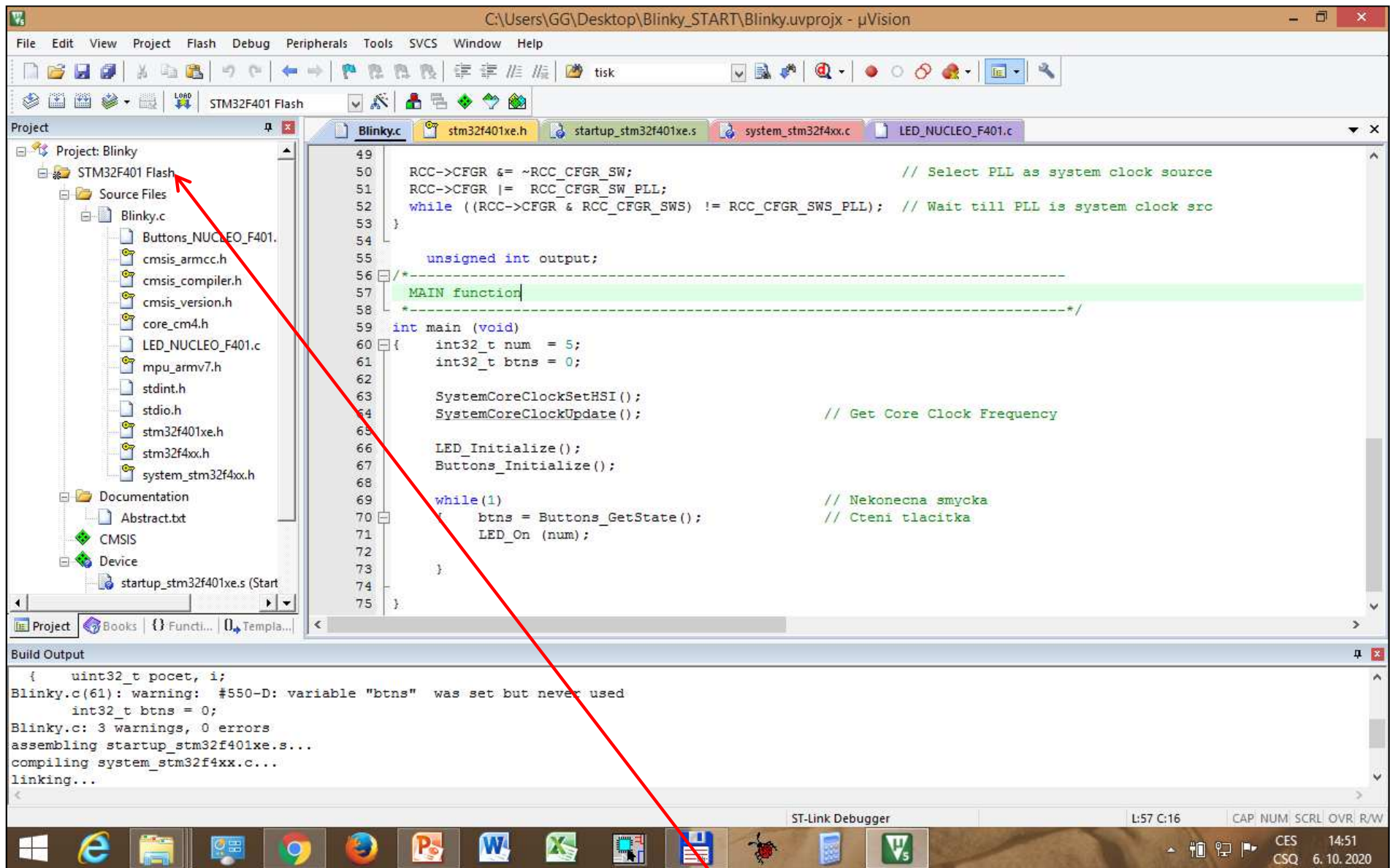
- ❖ Z Moodle si nakopírujeme soubor Blinky_START.zip, rozbalíme direktorář Blinky_START umístíme na disk.
- ❖ Otevřeme direktorář Blinky_START, kde budou umístěny následující programy.

PRVNÍ KROKY K REALIZACI PROGRAMU



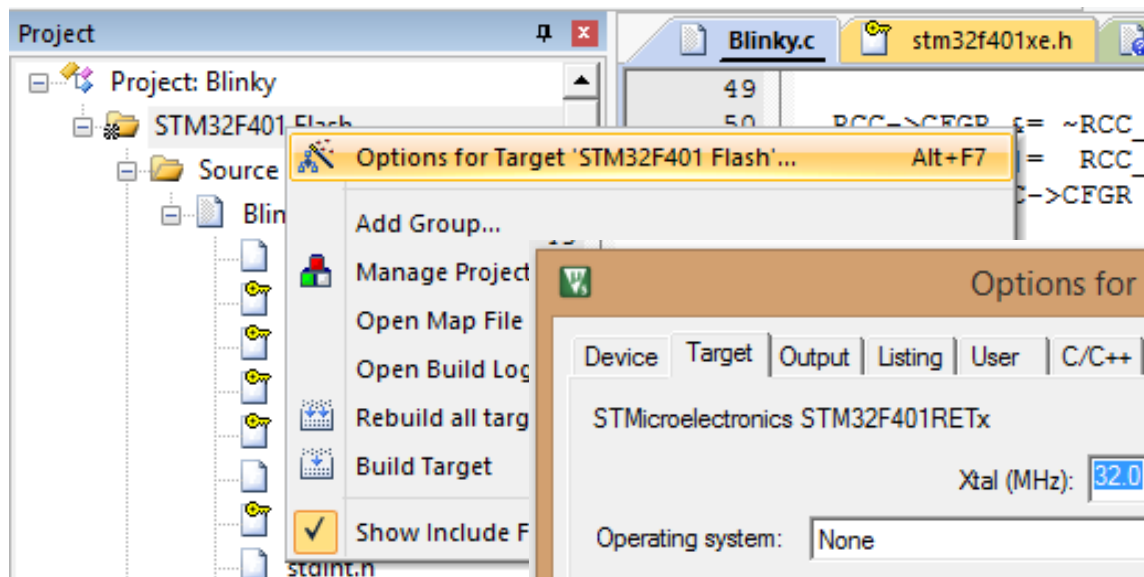
- ❖ Vývojové prostředí Keil uVision5 spustíme poklepáním na zelenou ikonku Blinky
- ❖ Pro seznámení se ze systémem a ověření funkčnosti bude potřeba doplnit programy Blinky.c, LED_NUCLEO_F401.c a Buttons_NUCLEO_F401.c. Dříve, než k tomu přistoupíte, zkontrolujeme propojení modulu s vývojovým prostředím.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

PRVNÍ KROKY K REALIZACI PROGRAMU



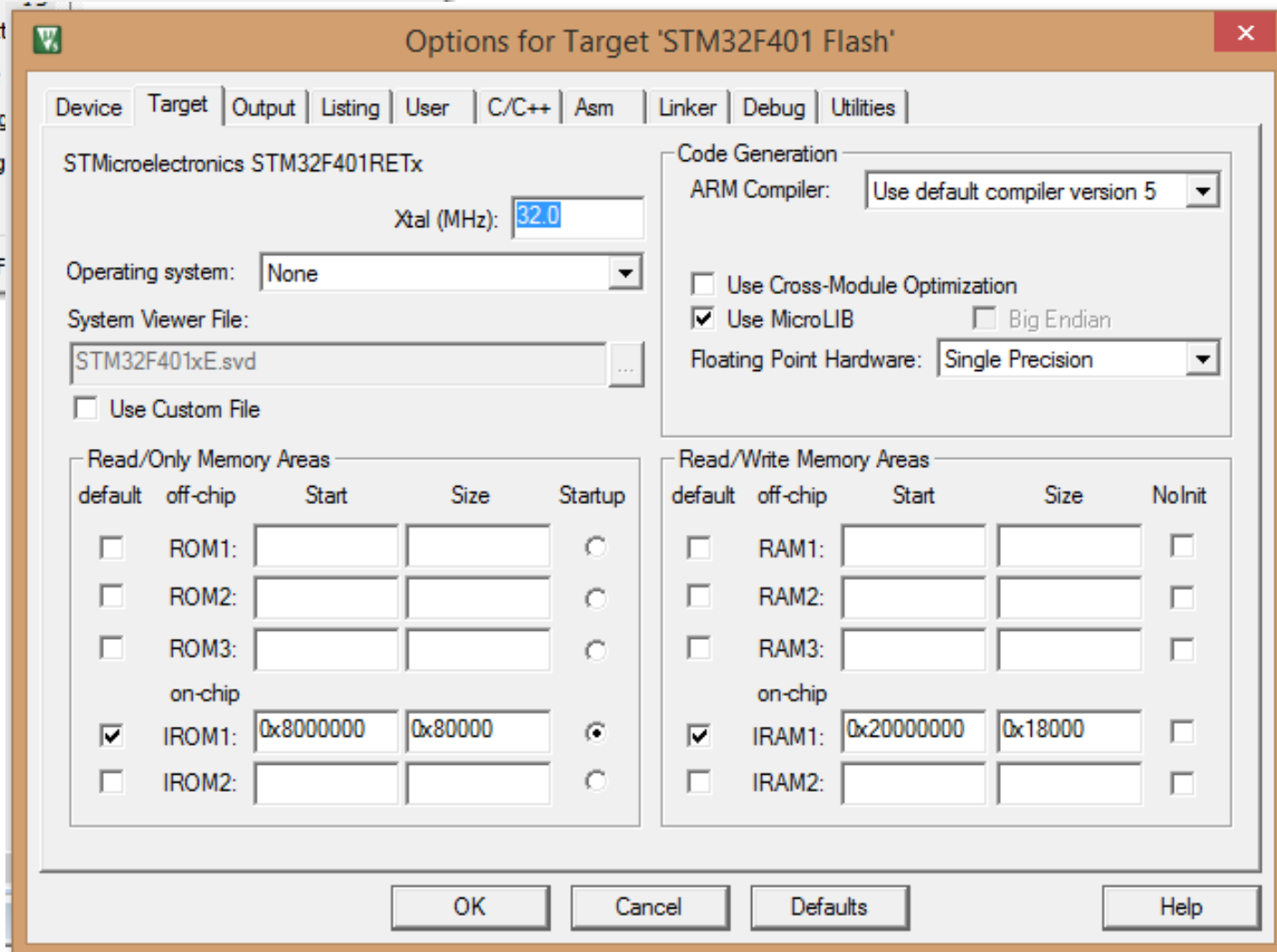
❖ Pravé tlačítko myši na název projektu

NEJDŮLEŽITĚJŠÍ NASTAVENÍ

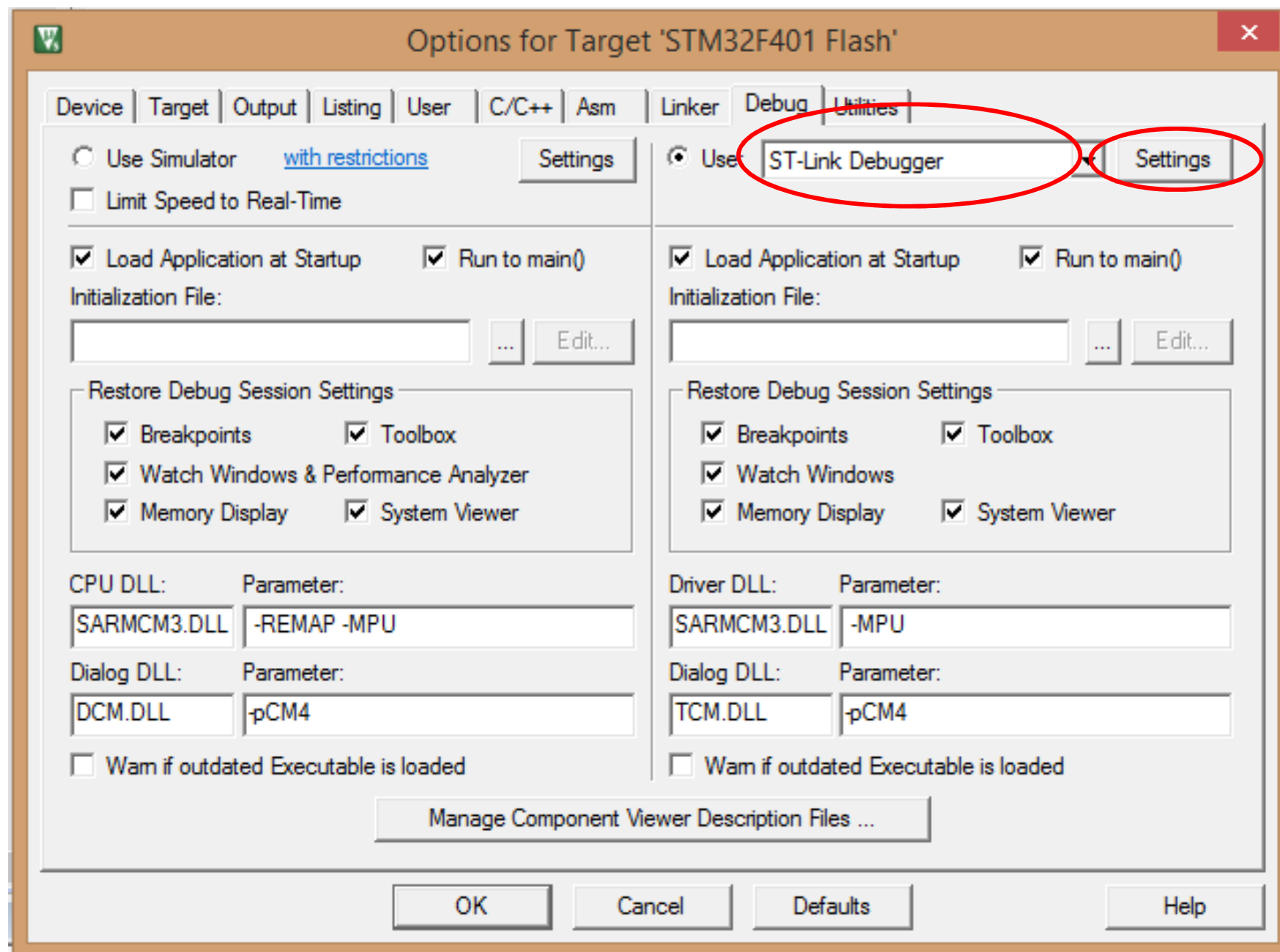


- ❖ Po vybrání položky Option for....
- ❖ Uvidíme okno

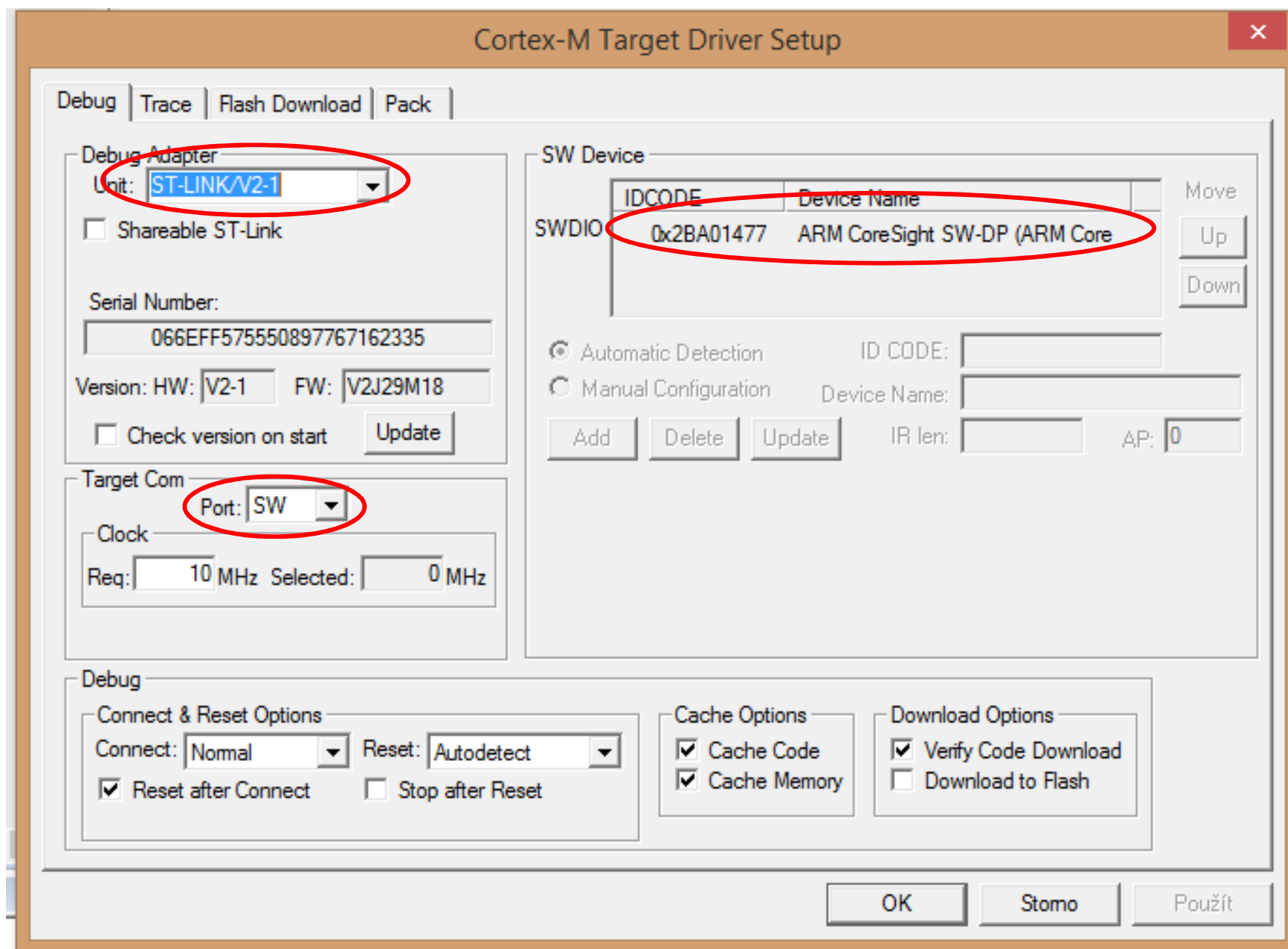
- ❖ Device – použitý procesor
- ❖ Target – hodinový kmitočet
- ❖ Debug – další stránka



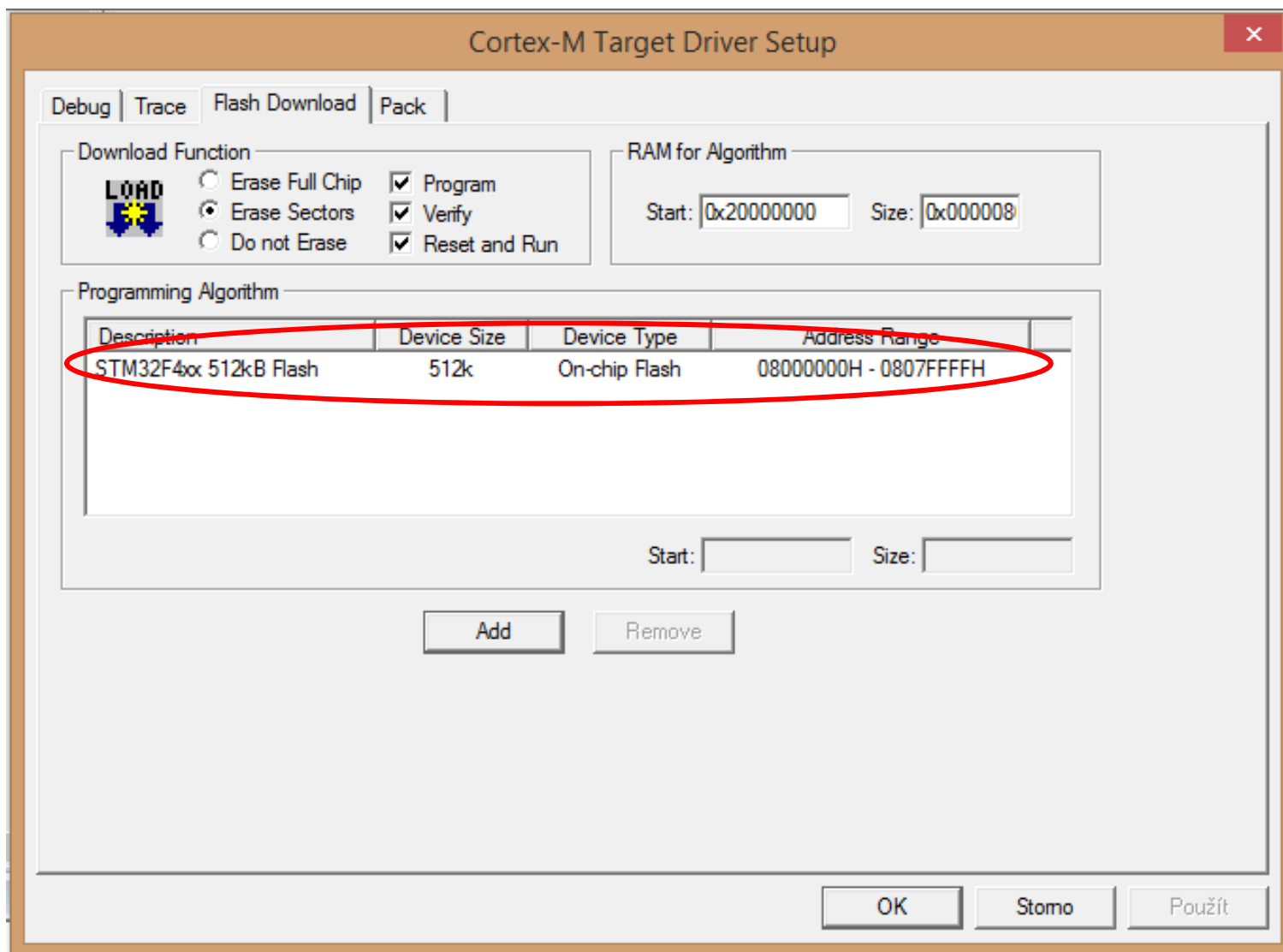
NEJDŮLEŽITĚJŠÍ NASTAVENÍ



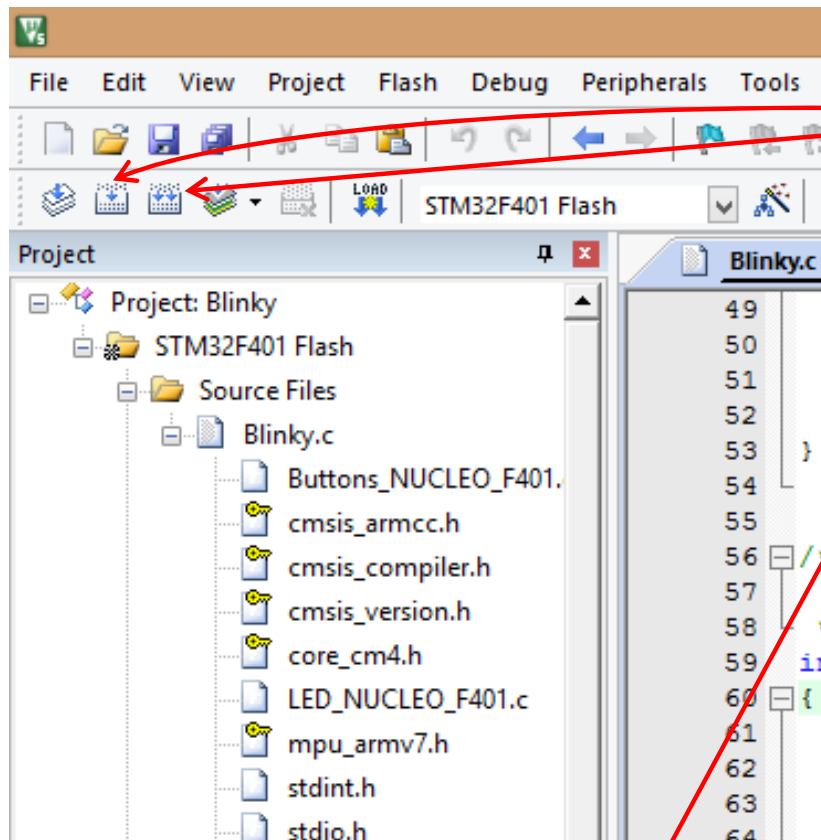
NEJDŮLEŽITĚJŠÍ NASTAVENÍ



NEJDŮLEŽITĚJŠÍ NASTAVENÍ



PŘEKLAD PROGRAMU

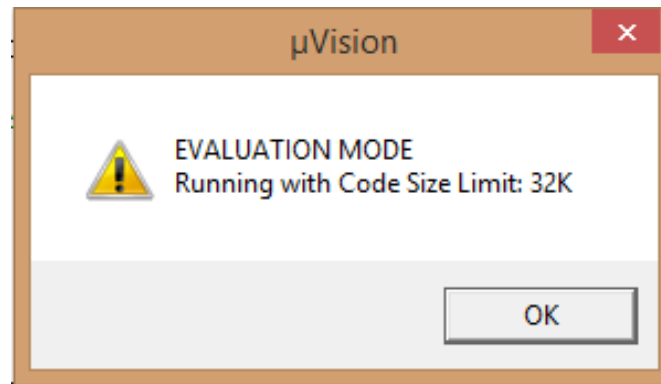


- ❖ Překlad programu zajistíme pomocí ikon
- ❖ Po překladu zkontrolovat zda je bez chyb, varování jsou většinou nevýznamná
- ❖ Code – velikost strojového kódu programu
- ❖ RO-data – velikost konstant v programu
- ❖ RW-data – velikost paměti pro proměnné

```
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 panes:

- Registers:** Shows the state of various registers. R4 is highlighted with a value of 0x08000484.
- Disassembly:** Shows the assembly code being executed. The instruction at address 0x08000294 is highlighted: `MOVS r4, #0x05`.
- Source:** Shows the C source code for the main function. A breakpoint is set at line 60.
- Watch:** Shows the value of the variable 'num' as 0x08000484.
- Call Stack + Locals:** Shows the current call stack with the main function at address 0x00000000.

The Command window at the bottom shows the 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 L:62 C:6.

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 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

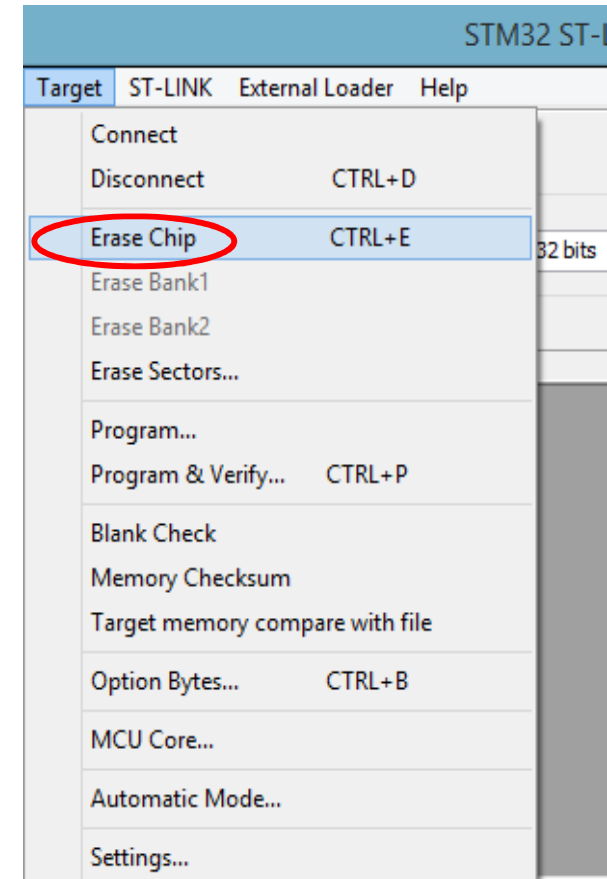
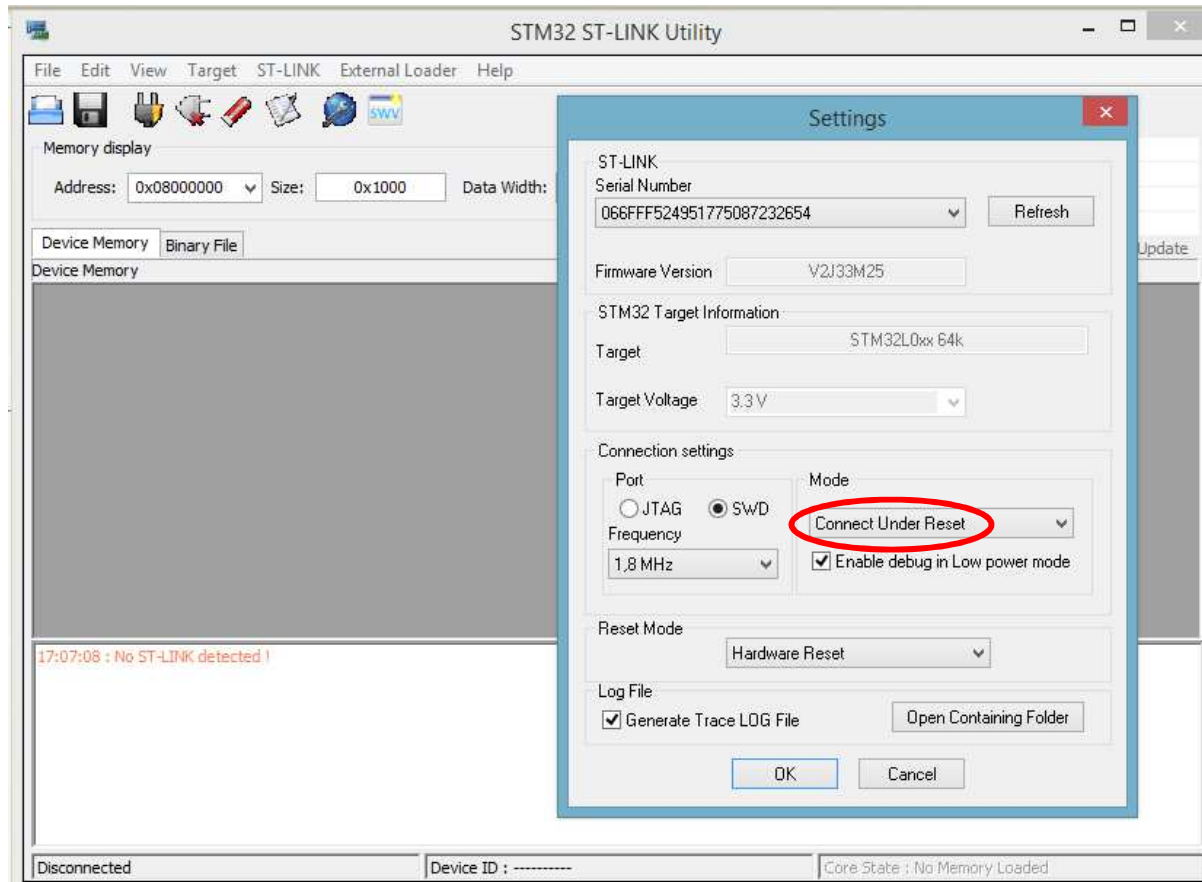
kanál 2 – PA7, PB5, PC7

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:

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". Below the menu is a toolbar with icons for file operations and connection. The "Memory display" section shows "Address: 0x08000000", "Size: 0x1000", and "Data Width: 32 bits". To the right, device information is displayed: "Device: STM32F401xD/E", "Device ID: 0x433", "Revision ID: Rev Z", and "Flash size: 512KBytes". A "LiveUpdate" checkbox is present. The main area shows "Device Memory @ 0x08000000 : Binary File" and "Target memory, Address range: [0x08000000 0x08001000]". A table displays memory contents:

Address	0	4	8	C	ASCII
0x08000000	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF
0x08000010	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF
0x08000020	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF
0x08000030	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF
0x08000040	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF
0x08000050	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF
0x08000060	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF
0x08000070	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF
0x08000080	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF

The bottom section shows a log of events:

```
14:55:19 : ST-LINK firmware version : v2.35M27
14:55:19 : Connected via SWD.
14:55:19 : SWD Frequency = 1,8 MHz.
14:55:19 : Connection mode : Connect Under Reset.
14:55:19 : Debug in Low Power mode enabled.
14:55:19 : Device ID:0x433
14:55:19 : Device flash Size : 512KBytes
14:55:19 : Device family :STM32F401xD/E
14:55:44 : Flash memory erased.
14:56:12 : Flash memory erased.
```

The status bar at the bottom indicates "Debug in Low Power mode enabled.", "Device ID:0x433", and "Core State : Live-Update:Disabled".