MATERIÁLY POTŘEBNÉ K PRÁCI S PROCESOREM ARM M4 F401

- **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**

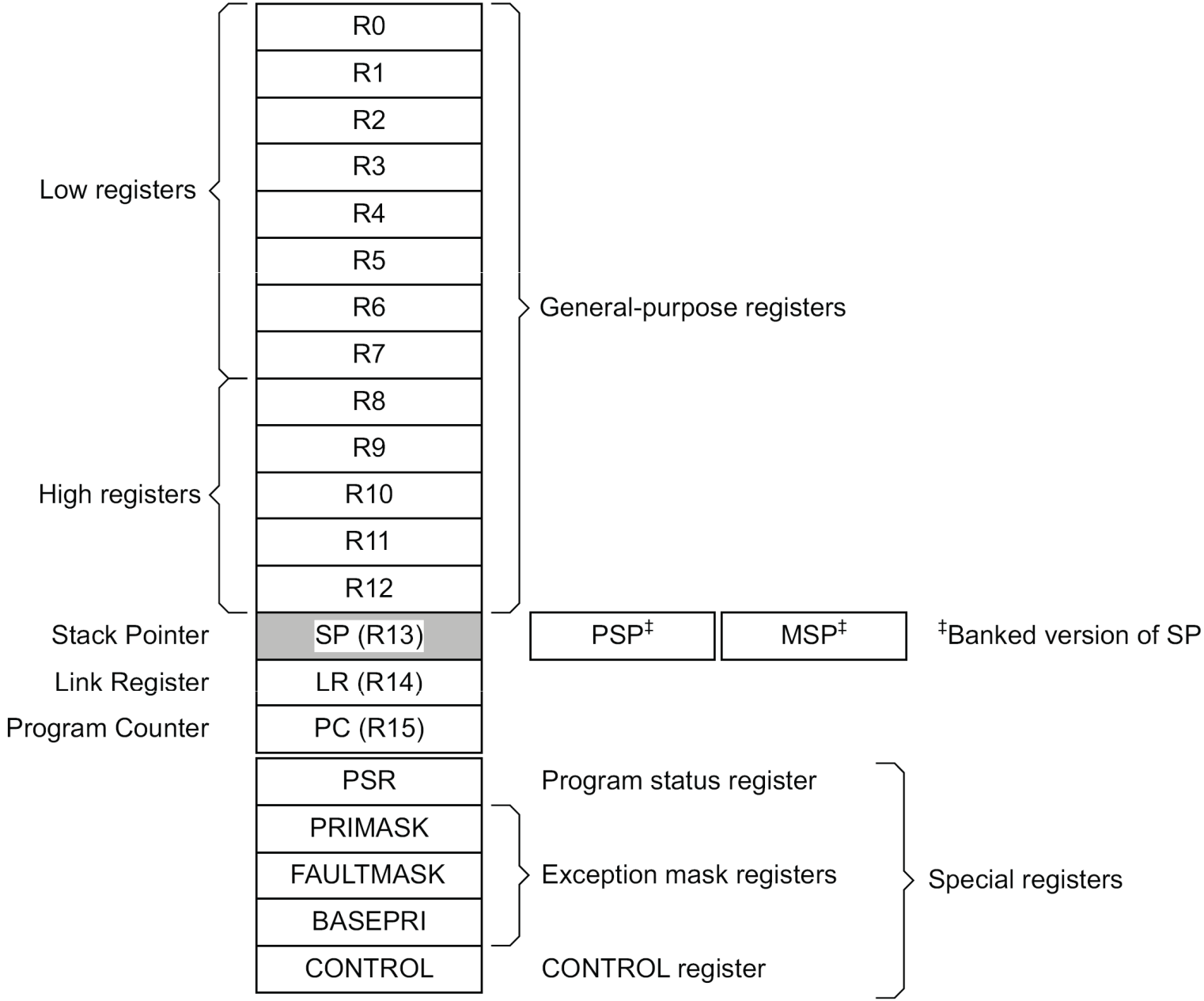
Pro podrobnější studium

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

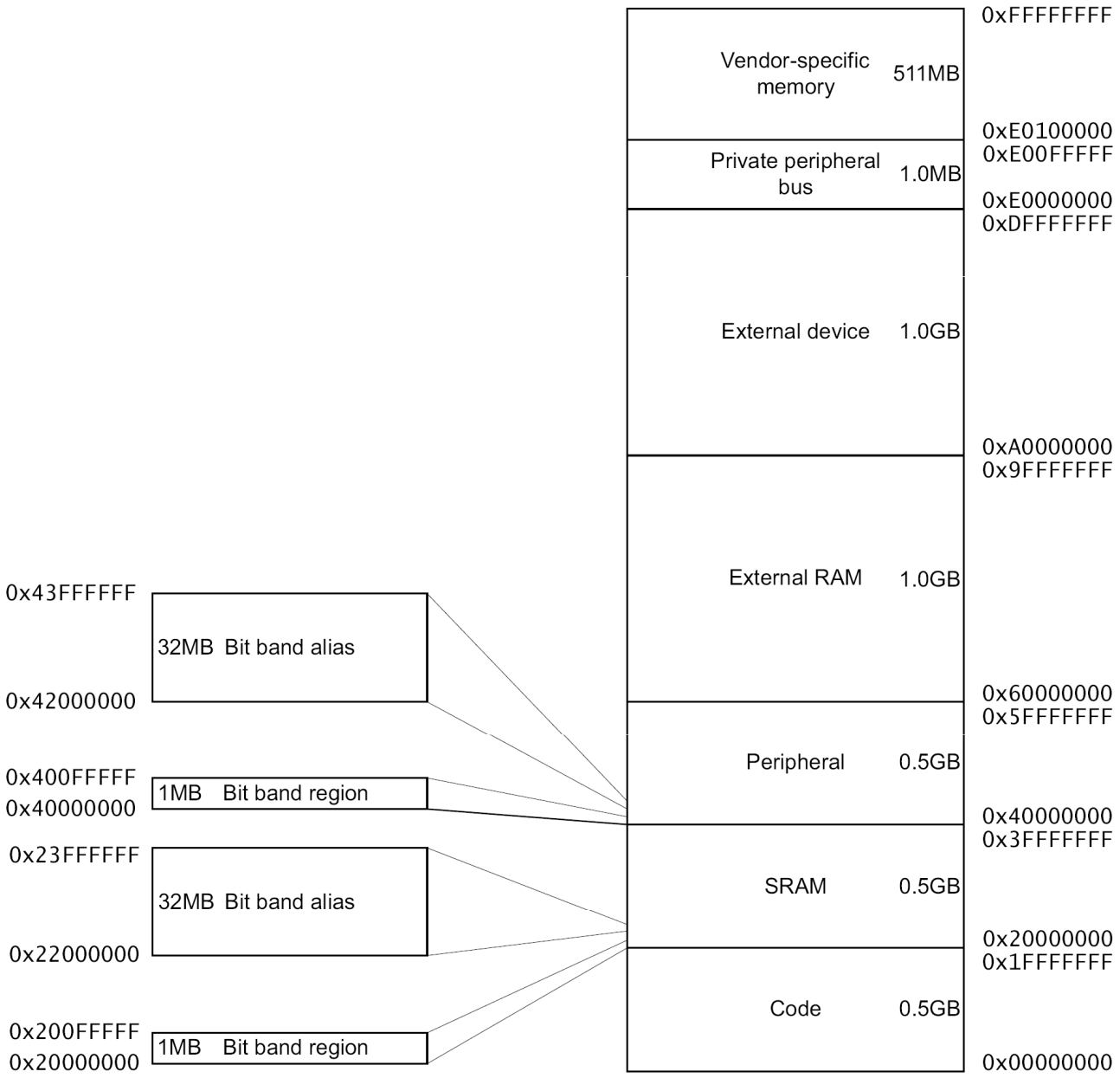


REGISTRY NA PROCESORECH ARM

Procesory Cortex-M jsou pro vykonávání programu vybaveny 16 registry R0 až R15. Jednotlivé registry jsou předurčeny k následujícímu použití:

- **R0–R12: všeobecně použitelné registry.** Některé 16-ti bitové instrukce Thumb® mají přístup pouze ke registrům R0 až R7.
- **R13: Ukazatel zásobníku** (stack pionter). Procesor je vybaven dvěma ukazateli (MSP a PSP) z nichž je v danou dobu použitelný pouze jeden. MSP defaultní ukazatel využívaný operačním systémem a v obsluze přerušení. PSP je ukazatel používaný v aplikacích. Spodní dva bity ukazatele zásobníku jsou vždy nulové, čímž je dosaženo zarovnání na dvě slova nebo adresu.
- **R14: Link Register.** Je-li volán podprogram, pak návratová adresa je uložena do LR (jednoúrovňový zásobník, může být přesunut do MSP nebo PSP).
- **R15: Program Counter (čítač instrukcí).** Čítač instrukcí obsahuje adresu zpracovávaného programu (instrukce). Zápisem hodnoty do R15 je realizována změna ve zpracování programu (skok, volání podprogramu nebo přerušení).
- **Speciální registry -** mají speciální funkci ovlivněnou speciálními instrukcemi. Nemohou být použity pro zpracování dat. Jedná se: **xPSR** – Program status registr, **PRIMASK, FAULTMASK a BASEPRI -** Interrupt Mask registers, **CONTROL** – řídící registr.

ROZLOŽENÍ ADRESOVÉHO PROSTORU ARM M3 A M4



VÝVOJOVÉ PROSTŘEDÍ FIRMY KEIL ELECTRONIC

Program ověříme ve vývojovém prostředí od firmy KEIL takto:

1) **Spustíme prostředí označené ikonkou Keil uVision5_Verze 37**
2) **V menu Projekt vybereme položku New Project**
3) **Zapíšeme jméno nového projektu**
4) **Vybereme CPU, pro který bude vytvářený SW určen (v našem případě STMicroelectronics STM32F401)**
5) **Po otevření okna Manage Run-Time Environment zašktneme v položce Device Startup pro programy v jazyce C a v položce CMSIS položku CORE**
6) **V okně projektu se vám objeví adresář Target 1 a v něm podadresář Source Group 1, do kterého vložíme vytvořený soubor v jazyce C. Vložení zajistíme buď kliknutím pravým tlačítkem myši na adresáři Source Group 1 a výběrem položky Add Existing Files to Group 'Source Group 1'. Po rozbalení podadresáře Source Group 1 se v něm tento soubor objeví.**

7) V menu Project > Option for Target 'Target 1' se zde v jednotlivých záložkách nastaví parametry projektu. V záložce Device můžeme změnit typ procesoru. V záložce Otput>Name of Executable můžeme změnit jméno spustitelného souboru. V záložce Target můžeme nastavit hodinový kmitočet procesoru. Při SW simulaci je vhodné zaškrtnout položku Use microLIB, která zajistí správné chování simulátoru při spuštění programu. Při práci s vývojovým modulem nemá zaškrtnutí položky Use microLIB vliv na chování programu ve vývojové prostředí. V záložce Debug se volíme v levé části Use Simulator pro SW simulaci vytvořeného programu nebo volíme ST-link Debuger při ladění nebo spouštění programu na vývojovém modulu.

8) Otevření hotového projektu realizujeme v menu Project > Open Project. Nestačí otevřít pouze zdrojový soubor !!

9) Zdrojový kód lze upravovat v interním editoru poklepáním na soubor se zdrojovým kódem v okně projektu. Soubor se otevře a můžete jej upravovat. Pozor v položce Open lze otevřít jakýkoliv zdrojový soubor, který lze upravovat, ale překládat se bude ten co je v projektovém okně.

10) K přeložení projektu můžete využít jak ikony v panelu nástrojů, tak v menu Project > Build target - zkratková klávesa F7. V dialogovém okně ve spodní části obrazovky se vypíše hlášení o úspěšnosti překladu. Došlo-li při překladu k chybám, jsou zde uvedeny druhy chyb a čísla řádků, na kterých se vyskytují. Na příslušný řádek je možné se dostat poklepáním na chybové hlášení. Po odstranění všech chyb se v dialogovém okně objeví hlášení 0 Error(s).

11) Simulace vytvářeného programu se spouští po úspěšném překladu z menu Debug > Start/Stop Debug Session zkratková klávesa CTRL+F5 nebo ikonou. Po potvrzení hlášení o Evaluation Mode verzi je simulátor připraven ke spuštění. V levé části se objeví okno s výpisem registrů a otevřou se případná další okna s výpisem stavu periférií a paměti, pokud si je vyberete menu Pheripherals nebo View. Simulace probíhá po Debug > Run klávesová zkratka F5 buď trvale nebo do bodu zastavení BreakPoints (vkládání nebo zrušení dvou klikem na tmavě šedivé místo před požadovaným řádkem červená značka viz.další blána).

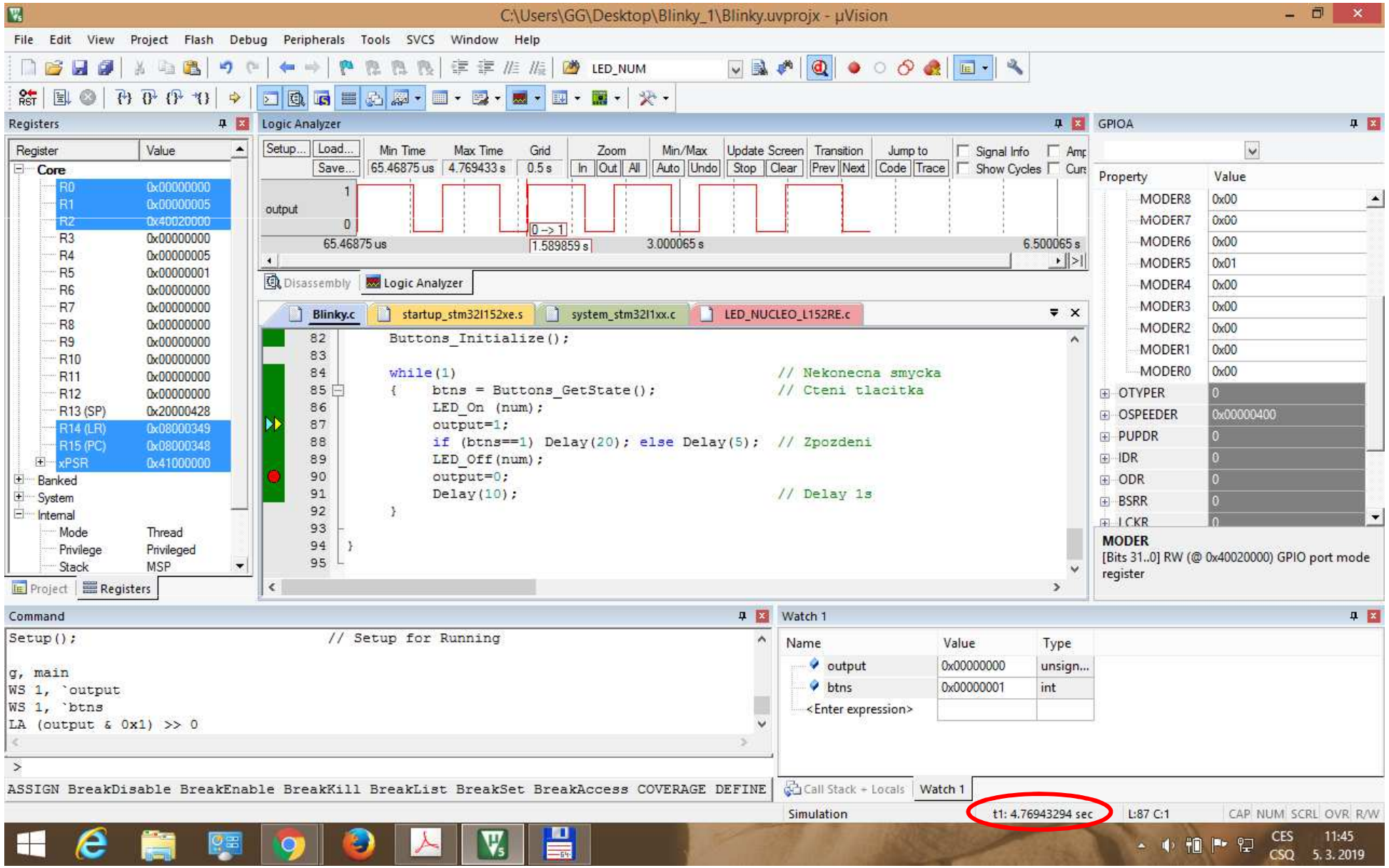
12) Krokování po jednotlivých instrukcích nebo řádcích jazyka C klávesa F11

13) Krokování s rychlým vykonáním podprogramů klávesa F10

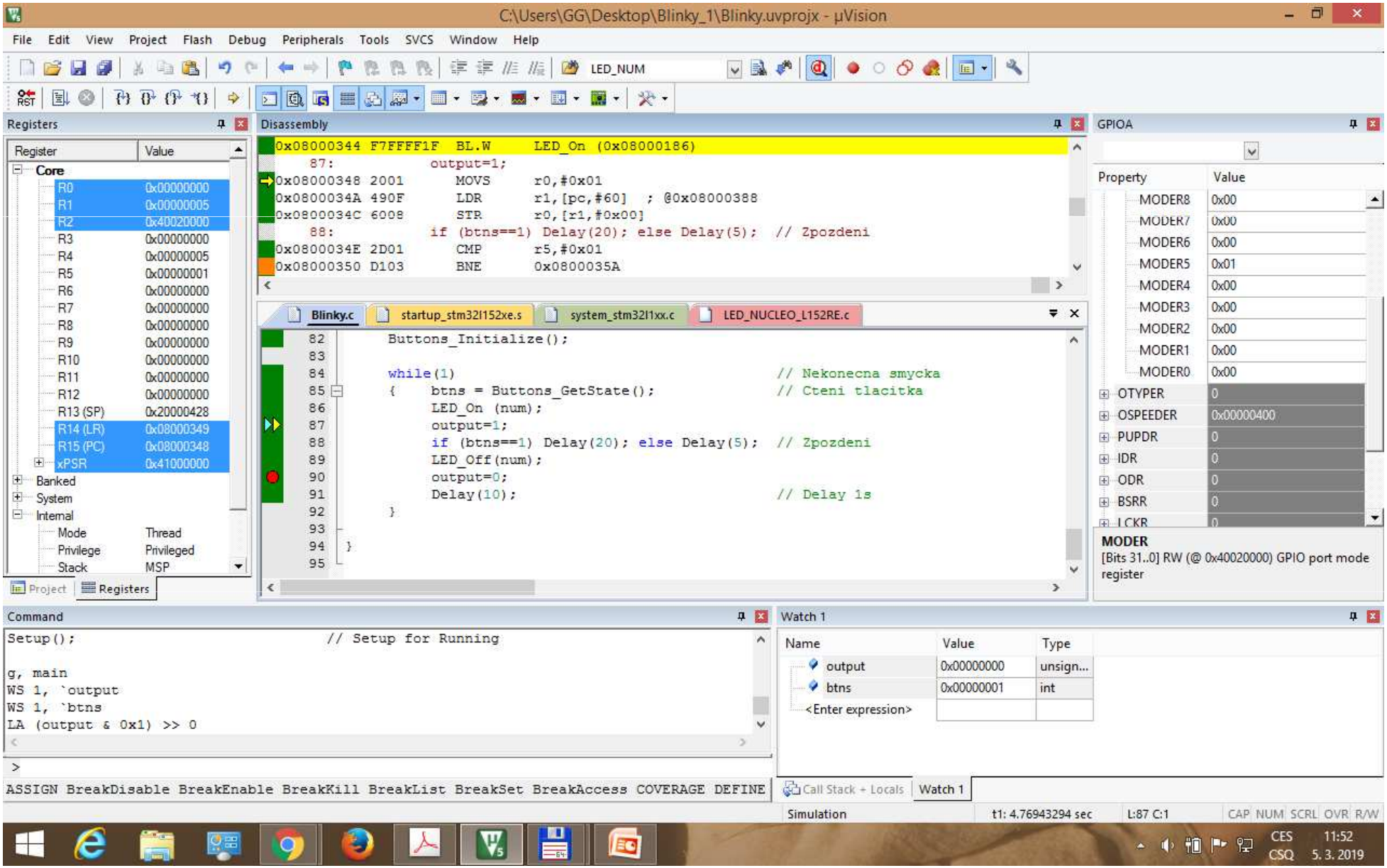
14) Aktuálně prováděný řádek je znázorněn žlutou šipkou. Části kódu, které byly během simulace alespoň jednou vykonány, jsou označeny na začátku řádku zeleně.

15) Hlášeni error 65: access violation at 0x40023800: no 'read' permission odstraníme zápisem MAP 0x40020000,0x40023FFC READ WRITE do příkazového řádku v daném běhu debugeru. Abychom nemuseli řádek zapisovat po každém spuštěním Debugeru uložíme řádek do souboru Dbg_RAM.ini.

VÝVOJOVÉ PROSTŘEDÍ FIRMY KEIL ELECTRONIC



VÝVOJOVÉ PROSTŘEDÍ FIRMY KEIL ELECTRONIC



Zobrazení průběhu hodnoty bitu, char, int nebo analogové hodnoty realizuje **Logický analyzátor**, který je funkční pouze při programové simulaci. **Zobrazovaná hodnota musí být globální**, lokální proměnné lze zobrazovat pouze v okně **Watch**. Veličina nebo její část může být v logickém analyzátoru omezena pomocí logického součinu příslušnou maskou vybírající bit nebo část veličiny.

V pravém dolním rohu je zobrazována doba, která uplynula do příslušného bodu zastavení nebo po každém kroku (**F11** nebo **F10**) v okně pro zdrojový program (jazyk C) nebo v okně **Disassembly**. V horním okně je zobrazen překlad zdrojového kódu do instrukcí assembleru. Řádek začíná adresou, kde je uložena instrukce, za ní následuje **operační kód**, případně ještě potřebná data a nakonec je uvedeno mnemonické označení instrukce v **jazyce symbolických adres** (assembleru).

PROMĚNNÉ V JAZYCE C PRO ARM V KEIL MDK 5

Datový typ	Bitů	Bytů	Rozsah
char (unsigned char)	8	1	0 ÷ 255 (zarovnaný byte)
signed char	8	1	-128 ÷ 127 (zarovnaný byte)
(signed) short	16	2	-32768 ÷ 32767 (zarovnaná polovina slova)
unsigned short	16	2	0 ÷ 65535 (zarovnaná polovina slova)
(signed) int	32	4	–2,147,483,648 ÷ 2,147,483,647
unsigned int	32	4	0 ÷ 4,294,967,295
signed long	32	4	–2,147,483,648 ÷ 2,147,483,647
unsigned long	32	2	0 ÷ 4,294,967,295
signed long long 64	64	8	–9,223,372,036,854,775,808 ÷ 9,223,372,036,854,775,807
unsigned long long 64	64	8	0 ÷ 18,446,744,073,709,551,615
float	32	4	±1.175494E-38 ÷ ±3.402823E+38
double 64 long double 64	64	8	±2.2250738585072014e-308 ÷ ±1.7976931348623157e+308
wchar_t	16	2	0 ÷ 65535
wchar_t	32	4	0 ÷ 4,294,967,295 je-li compilován s –wchar32

PŘÍZNAKY CORTEX M3 A M4

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

- Stavový registr aplikačního programu (APSR)
- Stavový registr přerušení (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 je nastaven, je-li výsledek aritmetické operace 0 nebo výsledek porovnání je shoda. V opačném případě je nulový.

C – Carry/Borrow je nastaven při aritmetické operaci, při které dochází k přenosu z bitu (b31 do bitu (b32) nebo při výpůjčce při odečítání. Pro instrukce ADC, ADD a CMN je nastavení bitu produktem přetečení (*unsigned* overflow). Pro instrukce CMP, SBC a SUB, je bit nastaven při výpůjčce (*unsigned underflow a borrow*).

Pro instrukce využívající posun (*shifting*) je příznak nastaven na poslední bit, který je posunem ztracen. Ostatní instrukce tento bit neovlivňují.

V - Příznak přetečení (Overflow) indikuje přetečení při aritmetické operaci sčítání nebo odčítání čísel se znaménkem. Jedná se o případ, kdy součet záporných čísel je kladný (došlo k přenosu mezi bity b31 a b30 a nedošlo k přenosu mezi bity b30 a b29) nebo součet kladných čísel je záporný (nedošlo k přenosu mezi b31 a b30 a došlo k přenosu mezi b30 a b29), kde bit b31 představuje znaménko.

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

LOGICKÉ INSTRUKCE A JEJICH VYUŽITÍ

Instrukce	Funkce
AND Rd, Rs, Oper2	Logický součin registru Rs s Oper2 uložený do registru Rd
ORR Rd, Rs, Oper2	Logický součet registru Rs s Oper2 uložený do registru Rd
EOR Rd, Rs, Oper2	Exclusive OR registru Rs s Oper2 uložený do registru Rd
BIC Rd, Rs, Oper2	Logický součin NOT registru Rs s Oper2 uložený do registru Rd
ORN Rd, Rs, Oper2	Logický součet NOT registru Rs s Oper2 uložený do registru Rd
Syntaxe: kde	*op{S}{cond} {Rd,} Rn, Oper2* ***op*** je AND , ORR, EOR, BIC nebo ORN ***S*** je volitelný sufix, umožňující aktualizaci příznaků podle výsledku ***cond*** je podmínka vykonání instrukce ***Rd*** je cílový registr operace ***Rn*** je zdrojový a případně cílový registr operace ***Oper2*** je druhý operand logické operace, kterým je 32 bitové číslo (*#constant*), registr s posunem (*Rm {, shift}*) nebo instrukce logického nebo aritmetického posunu (ASR *#n,* LSL *#n,* LSR *#n,* ROR *#n* nebo RRX)

LOGICKÉ INSTRUKCE A JEJICH VYUŽITÍ

Operace **logického součinu**, která realizuje logický součin mezi odpovídajícími bity obou operandů (nultý bit s nultým, první s prvním, atd.), se používá k tzv. **maskování bitů** nebo **nulování vybraných bitů**.

Použití – asembler:

Maskování spodních 4 bitů registru R5

```
LDR R5,#7Bh   R5 = 0111 1011
AND R5,R5,#0Fh    0000 1111
              R5 = 0000 1011
```

Použití – jazyk C:

Nulování 5 bitu proměnné param

```
param&=0xDF   interpretován
                  R0 = 0111 1011
ANL R0,R0,#DFh    1101 1111
                  R0 = 0101 1011
```

Operace **logického součtu**, která realizuje logický součet mezi odpovídajícími bity obou operandů (nultý bit s nultým, první s prvním, atd.), se používá k **nastavení vybraných bitů**.

Použití – asembler:

Maskování spodních 4 bitů registru R5

```
LDR R3,#7Bh   R3 = 0111 1011
ORR R5,R3,#04h    0000 0100
              R5 = 0111 1111
```

Použití – jazyk C:

Nastavení 5 bitu proměnné param

```
param|=0x04   interpretován
                  R0 = 0111 1011
ORR R0,R0,#04h    0000 0100
                  R0 = 0111 1111
```

LOGICKÉ INSTRUKCE A JEJICH VYUŽITÍ

Operace **EX-OR**, která realizuje operaci ⊕ mezi odpovídajícími bity obou operandů, se nejčastěji používá ke k **změně vybraných bitů (inverzi)**.

Použití – asembler:

Negace spodních 4 bitů registru R5

```
LDR R5,#7Bh   R5 = 0111 1011
EOR R5,R5,#0Fh    0000 1111
              R5 = 0111 0100
```

Použití – jazyk C:

Negace nejnižšího bitu proměnné param

```
Param^=0xDF  interpretován
                R0 = 0111 1011
EOR R0,R0,#01h    0000 0001
                R0 = 0111 1010
```

Pokud procesor nedisponuje efektivní instrukcí nulování nebo jednotkového doplňku, můžeme k uvedeným operacím využít operaci EX-OR.

Použití – asembler:

Nulování registru R0 a vytvoření jednotkového doplňku registru R5

```
EOR R0,R0,R0  R0 = 0000 0000
              R5 = 0110 1001
EOR R5,#0FFh  R5 = 1001 0110
```

Použití – jazyk C:

Nulování proměnné test

```
Příkaz test ^ =test; bude
interpretován instrukcí
                R3 = 1001 0111
XOR R3,R3       R3 = 0000 0000
Pak R3 přesuhuto na odpo-
vídající paměťové místo
```

STM32F4xx – PŘIPOJENÍ JEDNOTLIVÝCH PERIFERIÍ KE SBĚRNICÍM

Před použitím jakékoliv periferie je potřeba provést - její vynulování (zajišťuje reset procesoru, ale nezajišťuje jej například programový restart uP), přivedení hodinového signálu a nastavením do požadovaného režimu. Činnost zajišťuje skupina registrů – **RCC** (**Reset and Clock Control**). Periferie procesoru STM32F4 můžeme rozdělit do skupin podle toho, ke které sběrnici jsou připojeny. Sběrnice můžeme rozdělit na **AHBx**, ke kterým jsou většinou připojeny pomocné jednotky usnadňující realizaci některých operací a **APBx**, ke kterým jsou připojeny periferní obvody.

AHB1 – USB, Ethernet MAC, DMA2, DMA1, CRC module, GPIOx
AHB2 – USB, Random generátor, Hash module, Cryptographic module, Camera interface
AHB3 – Flexible static memory controller

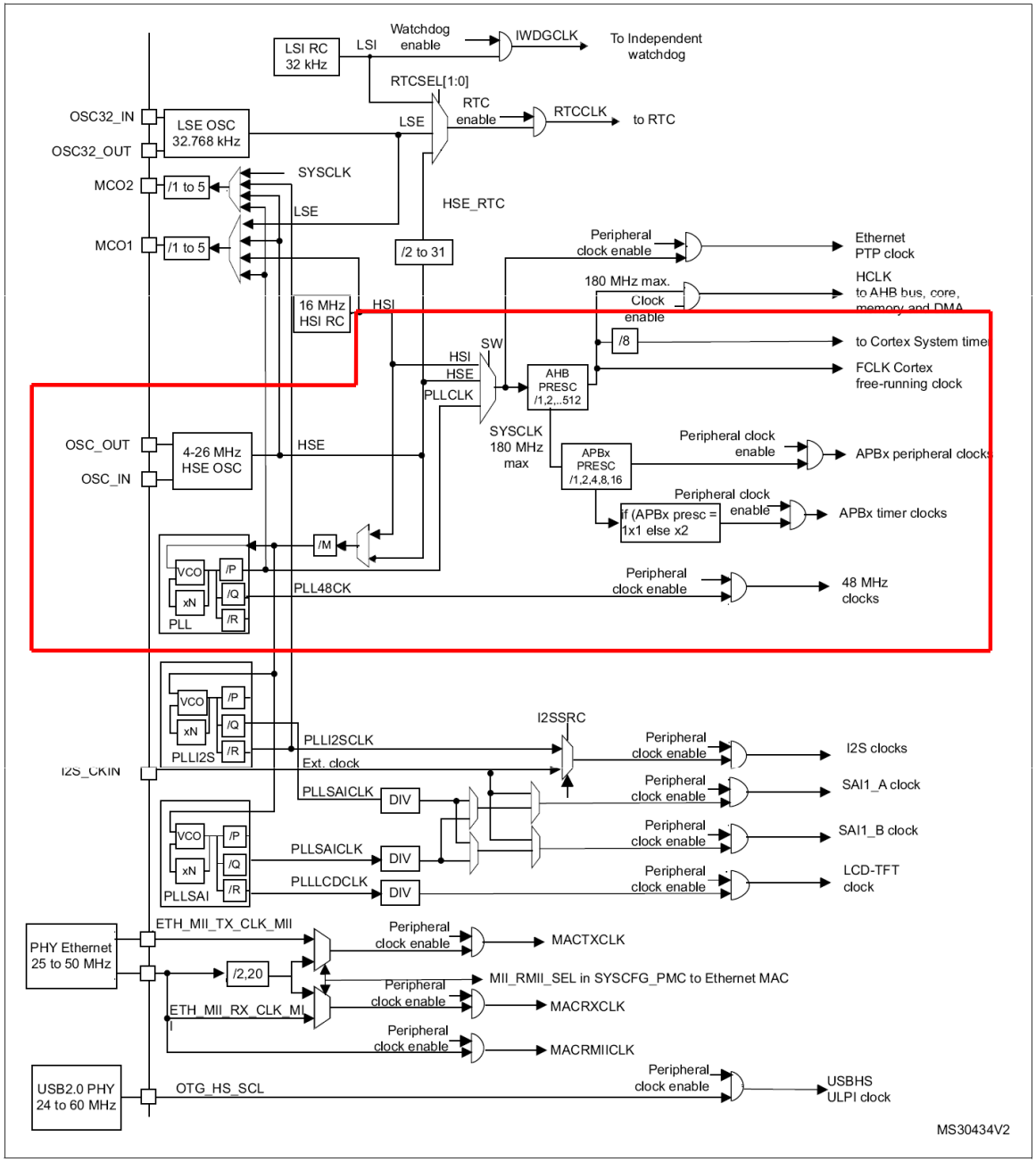
APB1 – DAC, Power interface, CAN2, CAN1, I2C2, I2C1, UART 2-5, SPI2, SPI3, Watchdog, TIM 2-7 a TIM12-14
APB2 – TIM 9-11, System configuration controller, SPI1, SDIO, ADC, USART 1 a 6, TIM 8 a 1

Každá skupina disponuje registry - pro nulování **AHBxRSTR** a **APBxRSTR,** pro povolení hodin **AHBxENR** a **APBxENR,** a **AxBxLPENR** nastavující povolený nebo zakázaný hodinový signál v režimu se sníženou spotřebou.

KONFIGURACE HODINOVÉHO SYSTÉMU PROCESORU

Pro první programy je důležitá červeně označená část v zapojení hodinového systému STM32-F401, která určuje hodinový signál pro procesorové jádro a časování sběrnic APBx.

Po připojení napájení je aktivován interní RC oscilátor HSI jako zdroj hodin CPU. Následně je v inicializačním programu nakonfigurován způsob čtení z paměti flash a hodinový signál pro sběrnice APB1 a APB2. Na závěr je zakázána činnost PLL, nakonfigurovány dělící poměry pro požadovaný hodinový kmitočet. Následně je PLL spuštěn, je vyčkáno na jeho stabilizaci a nakonec vybrán jako zdroj systémového kmitočtu. Celá konfigurace v jazyce C je na následující bláně.

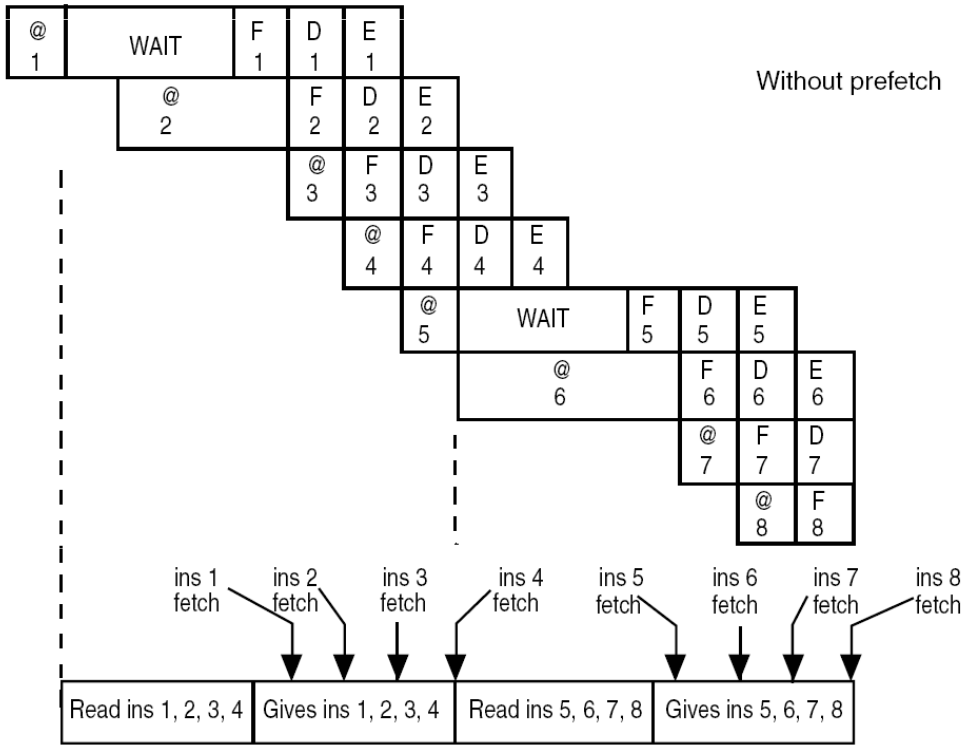


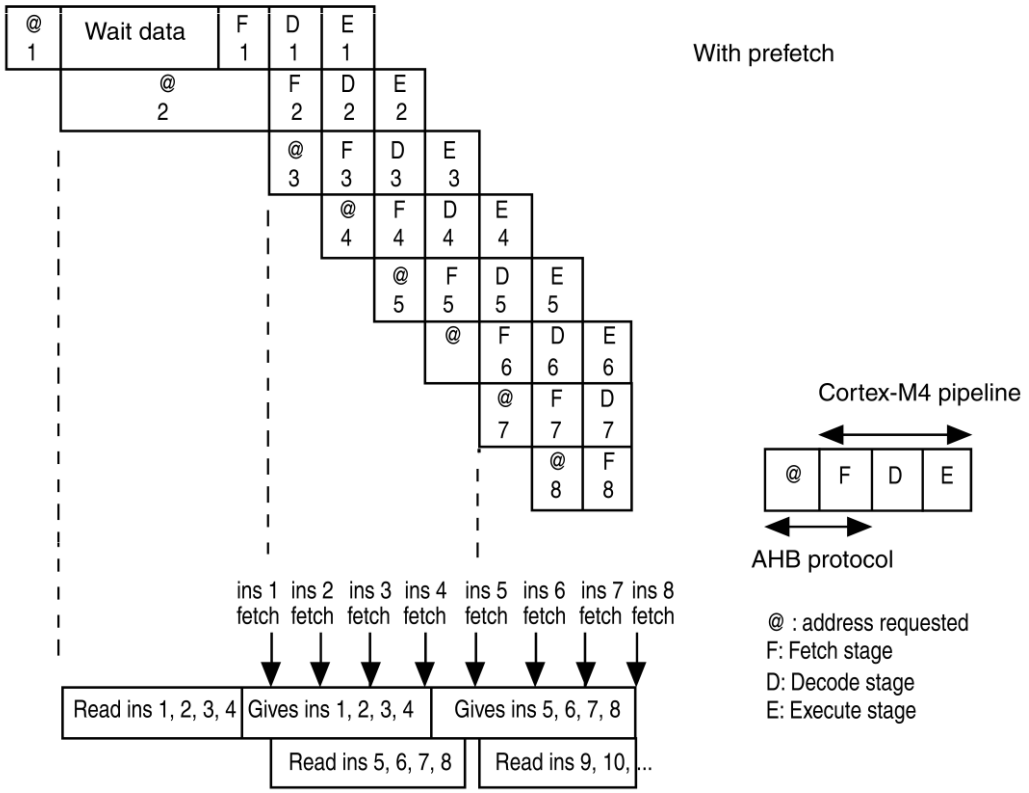
KONFIGURACE HODINOVÉHO SYSTÉMU PROCESORU F401

```
//     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
}
```

SYSTÉM ČTENÍ PROGRAMU Z PAMĚTI FLASH U PROCESORŮ F4

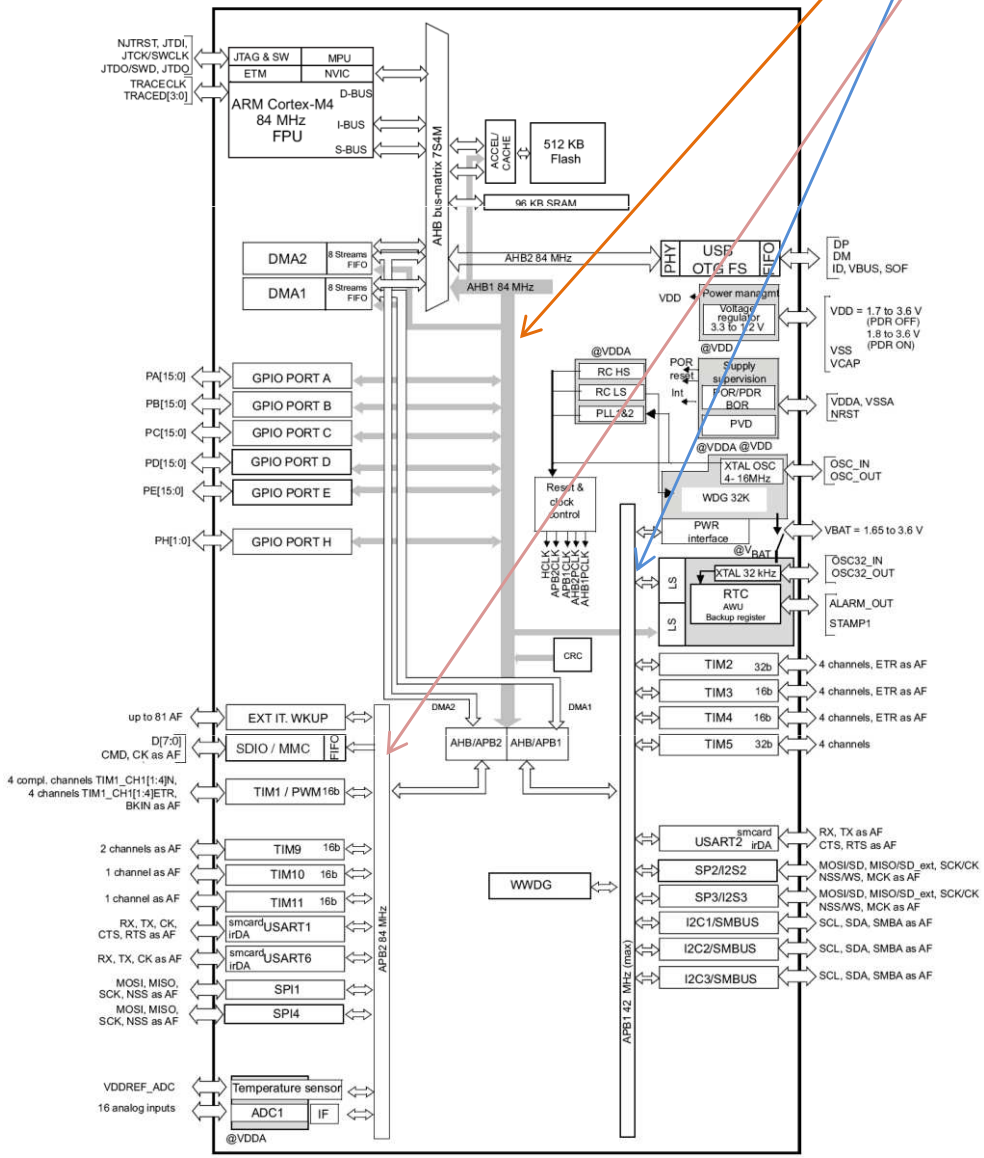
```
FLASH->ACR  = FLASH_ACR_PRFTEN;         // Enable Prefetch Buffer
FLASH->ACR |= FLASH_ACR_ICEN;           // Instruction cache enable
FLASH->ACR |= FLASH_ACR_DCEN;           // Data cache enable
FLASH->ACR |= FLASH_ACR_LATENCY_5WS; // Flash 5 wait state
```





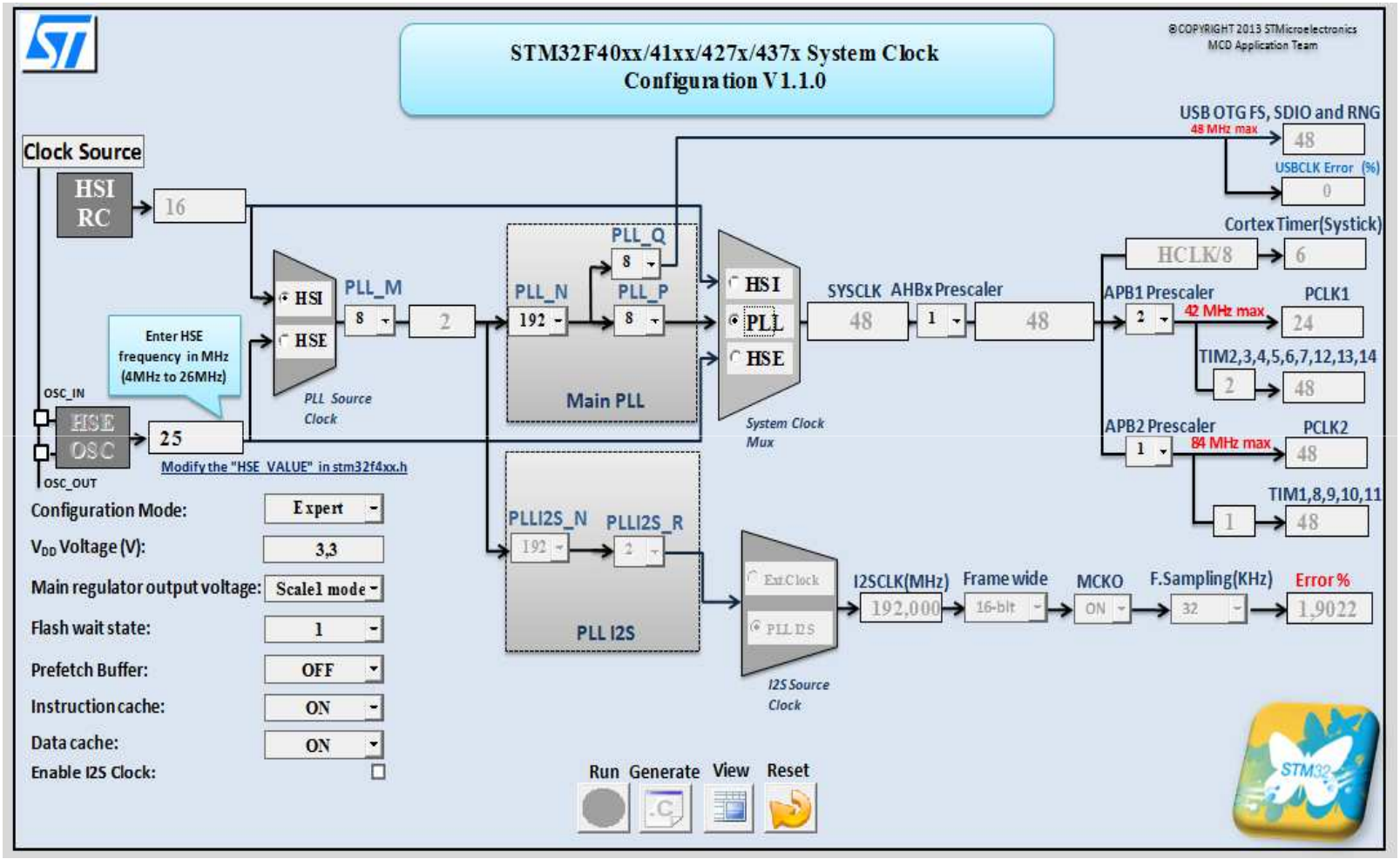
KONFIGURACE HODINOVÉHO SIGNÁLU PRO SBĚRNICE

```
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
```

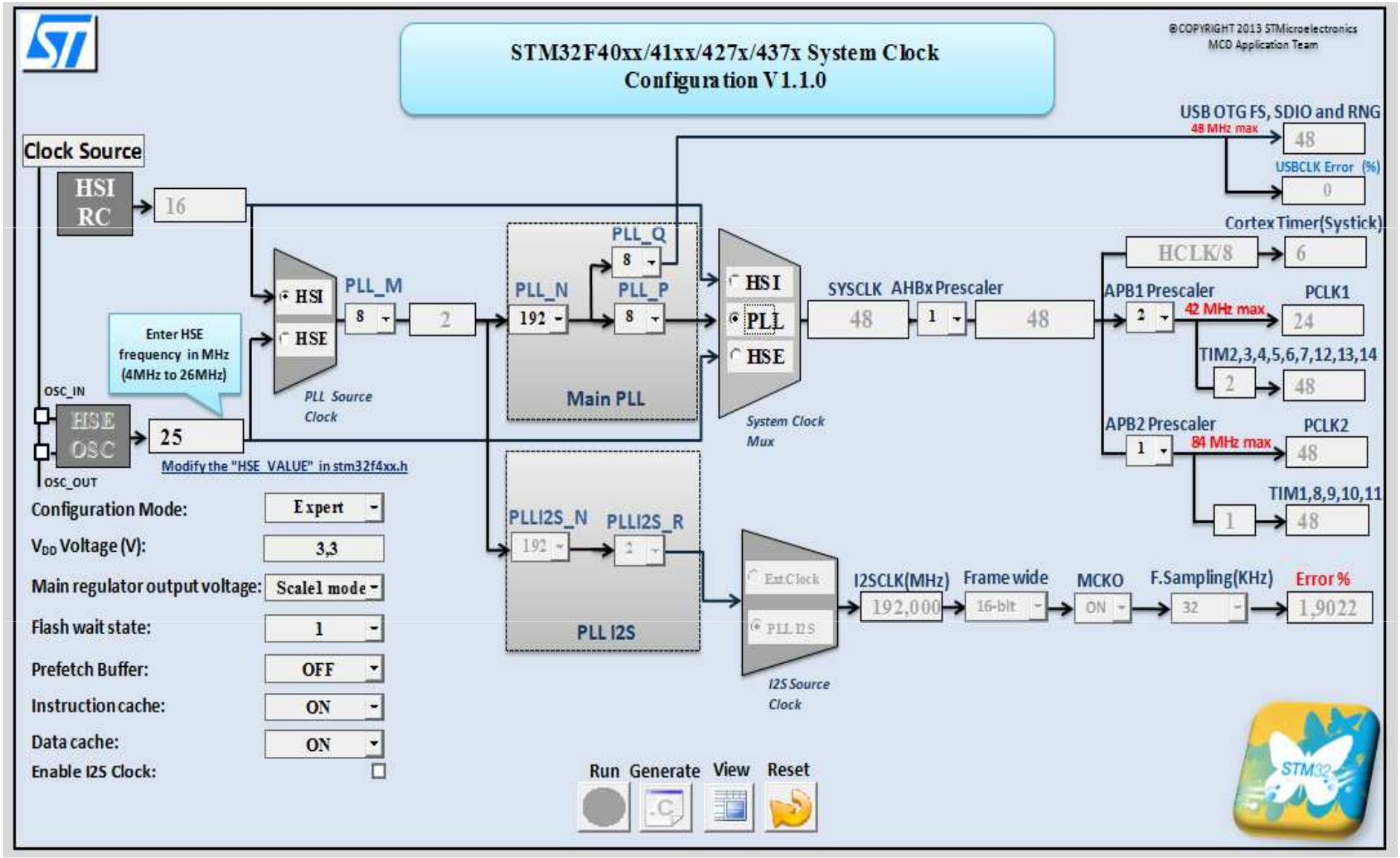


KONFIGURACE PLL HODINOVÉHO SYSTÉMU PROCESORU F401

```
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 HODINOVÉHO SYSTÉMU PROCESORU F401



KONFIGURACE HODINOVÉHO SYSTÉMU PROCESORU

V programu na před předcházejících blánách jsou realizovány logické operace se symbolickými hodnotami a nikoliv s číselnými hodnotami. Názvy se nachází v souboru

stm32f401xe.h

Budeme-li v programu používat tyto názvy, **nemusíme** při přechodu na jiný procesor od ST elektronic **ověřovat** umístění jednotlivých bitů v konfiguračních registrech. Při nastavování bitů pomocí následujících funkcí je pro každý procesor nutné zkontrolovat jejich umístění v manuálu.

```
#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))
```

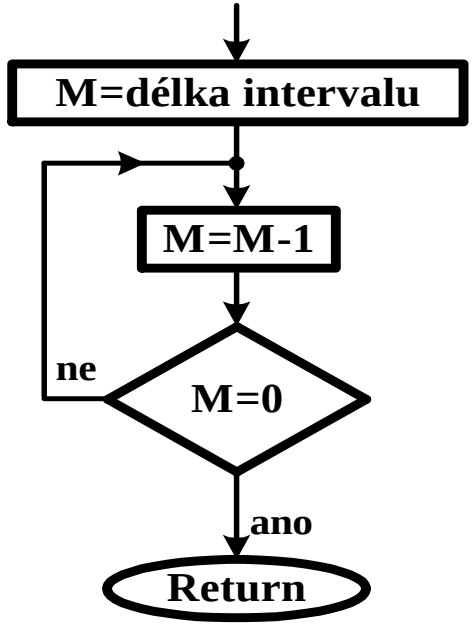
PROGRAMOVĚ GENEROVANÉ ZPOŽDĚNÍ

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 nastavíme hodnotu odpovídající požadovanému zpoždění. Ta je postupně dekrementována/ inkrementována za pomocí 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.

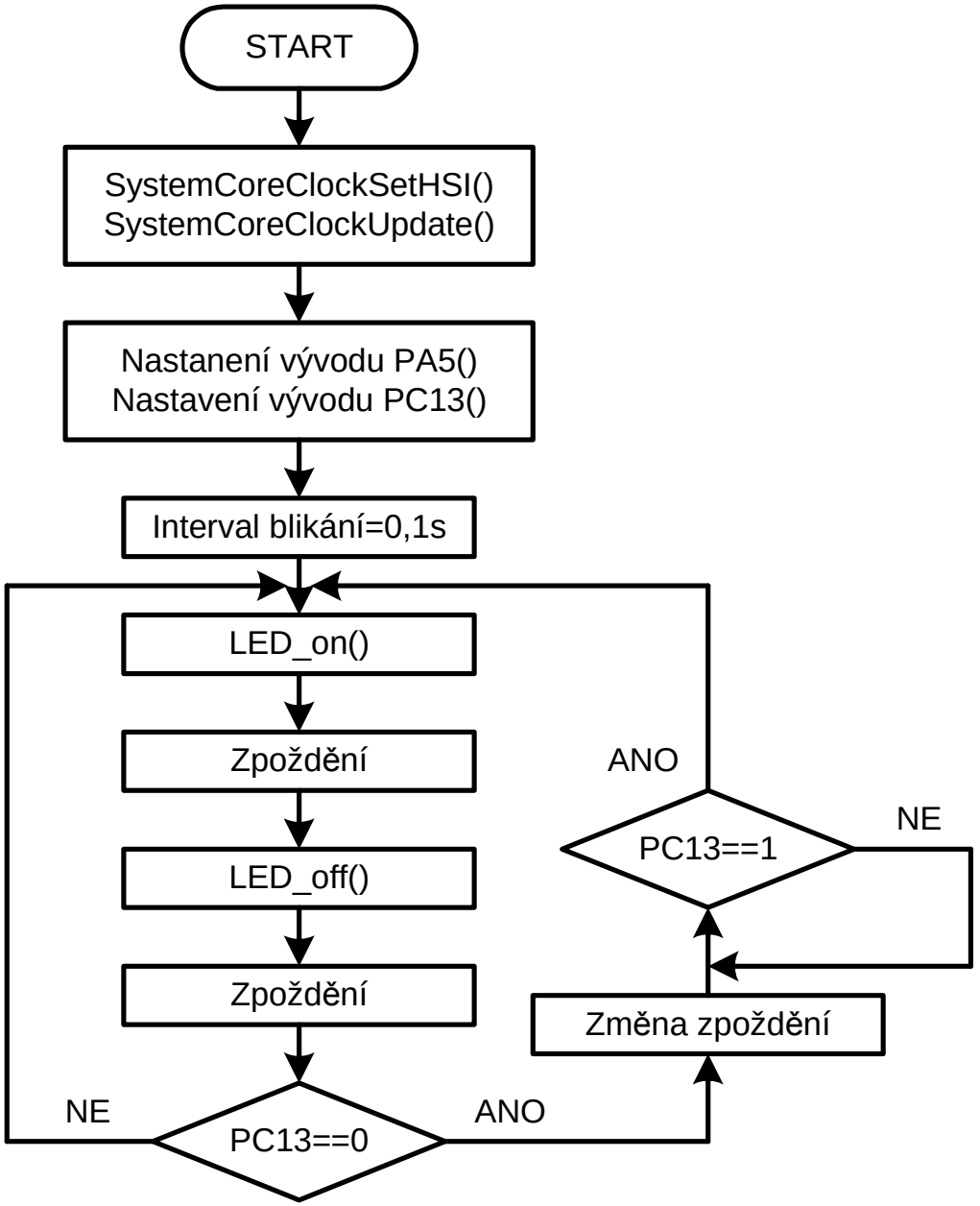


PROGRAMOVĚ GENEROVANÉ ZPOŽDĚNÍ V JAZYCE C A ASEMBLERU

```
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;}
}
```

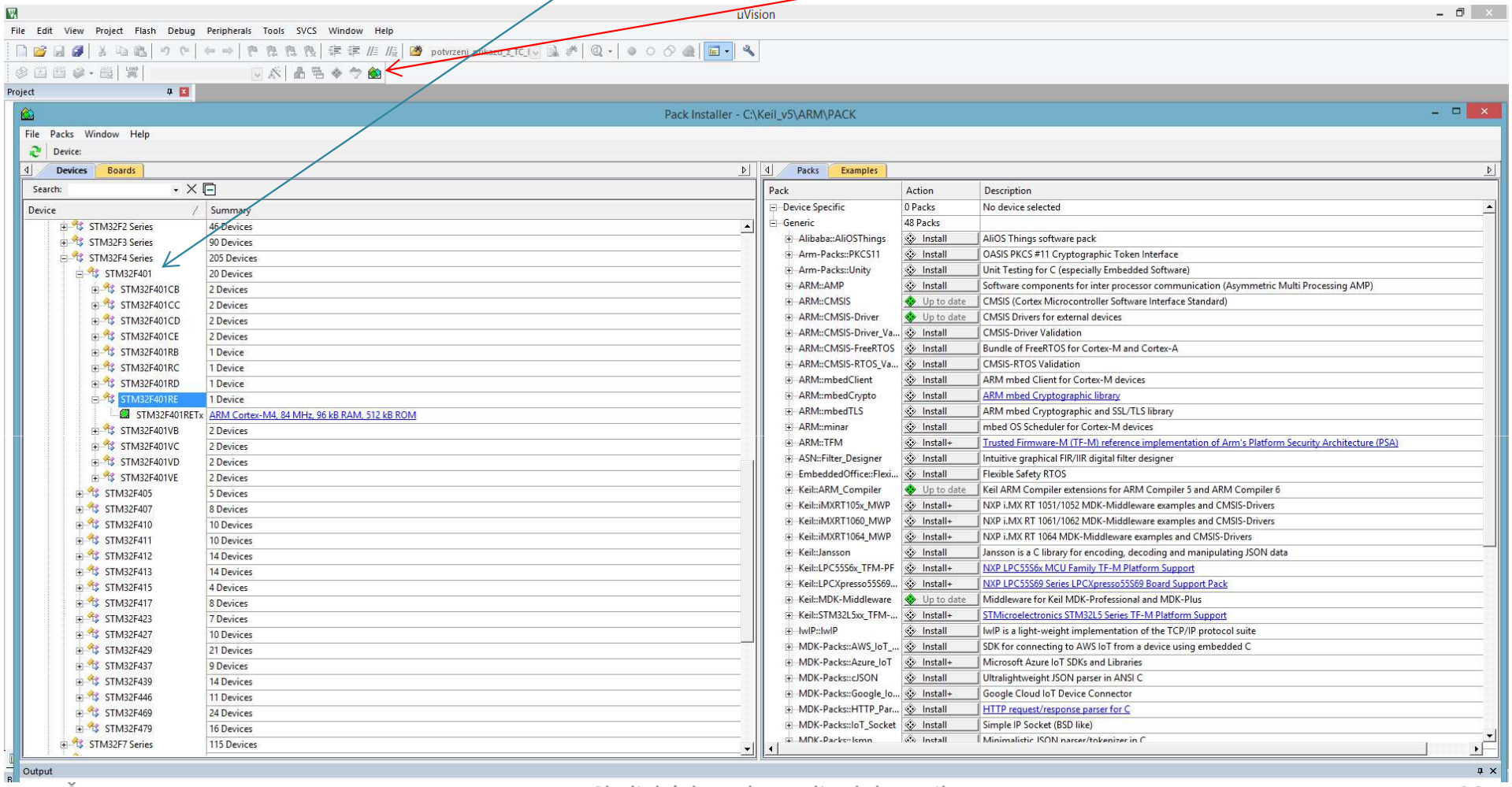
```
;*******************************************************************************
;* Jméno funkce           DELAY
;* Popis                  Softwarové zpožděni procesoru
;* Mění stav registrů     R0 a R3
;* Vstup                  R0 = počet opakování cyklu zpoždění
;* Vystup                 Žádný
;* Komentář               Podprogram zpozdí průběh vykonávaní programu
;*******************************************************************************
DELAY                     ; Navěstí začátku podprogramu
        PUSH    {LR}      ; Uloženi hodnoty návratové adresy - LR do zásobníku
WAIT1
        LDR     R3, =doba         ; Vloženi konstanty doba pro prodlevu do R3
WAIT    SUBS    R3, R3, #1        ; Odečteni 1 od R3,tj. R3 = R3 - 1 a nastaveni příznakového
                                  ; registru
        BNE     WAIT              ; Skok na navěstí při nenulovosti R3 (skok dle příznaku)
        SUBS    R0, R0, #1        ; Odečteni 1 od R0,tj. R0 = R0 - 1 a nastaveni příznakového
                                  ; registru
        BNE     WAIT1             ; dokud není nula v R0, skočí na wait1
        POP     {LR}              ; Návrat z podprogramu, obnoveni 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
```

IDEOVÉ ŘEŠENÍ ZÁKLADU ÚLOHY 1



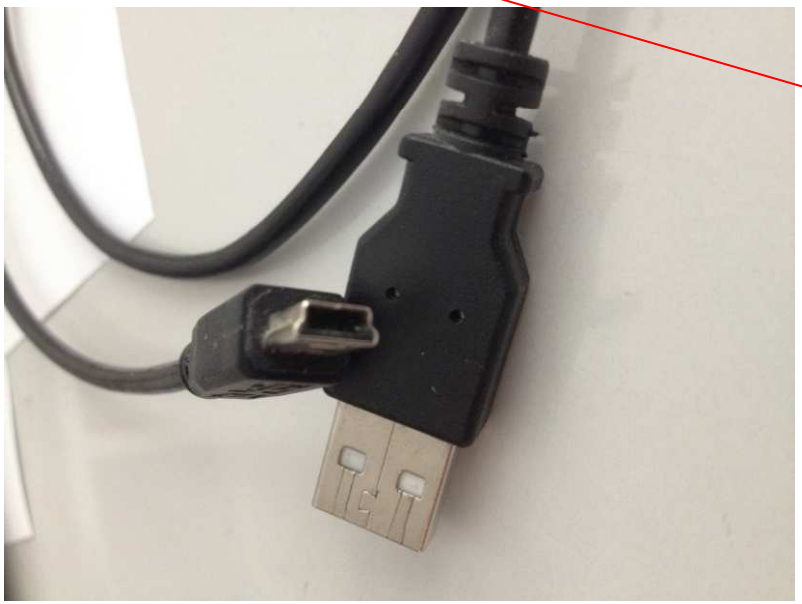
PRVNÍ KROKY K REALIZACI PROGRAMU

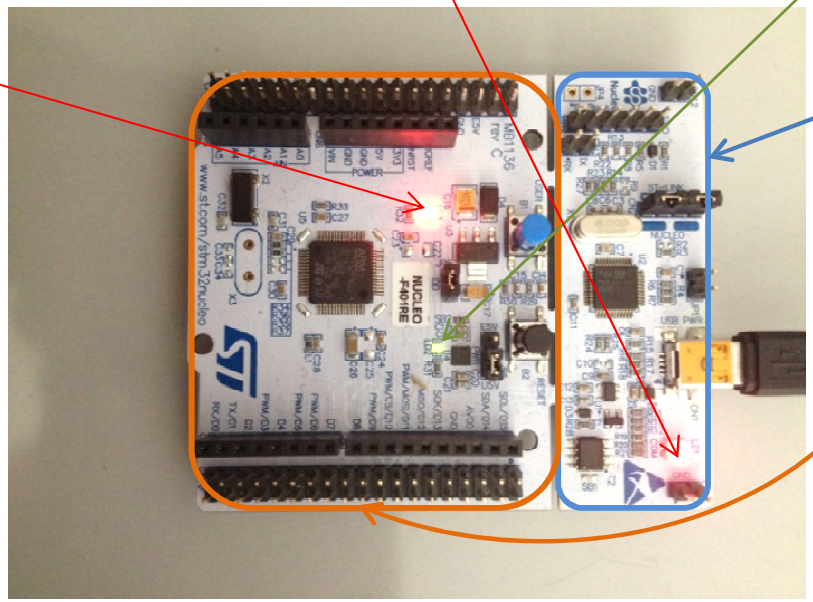
- Nainstalujeme si na PC program Keil uVision5 5.36 nebo vyšší a STM32 ST-LINK Utility V3.8.0, který je k dispozici na Moodle.
- Po instalaci nebo po prvním spuštění zmáčkneme ikonu programu pack. Vybereme ST electronic, STM32F4 a poklepeme na ni. Nahrají se potřebné knihovny.



PRVNÍ KROKY K REALIZACI PROGRAMU

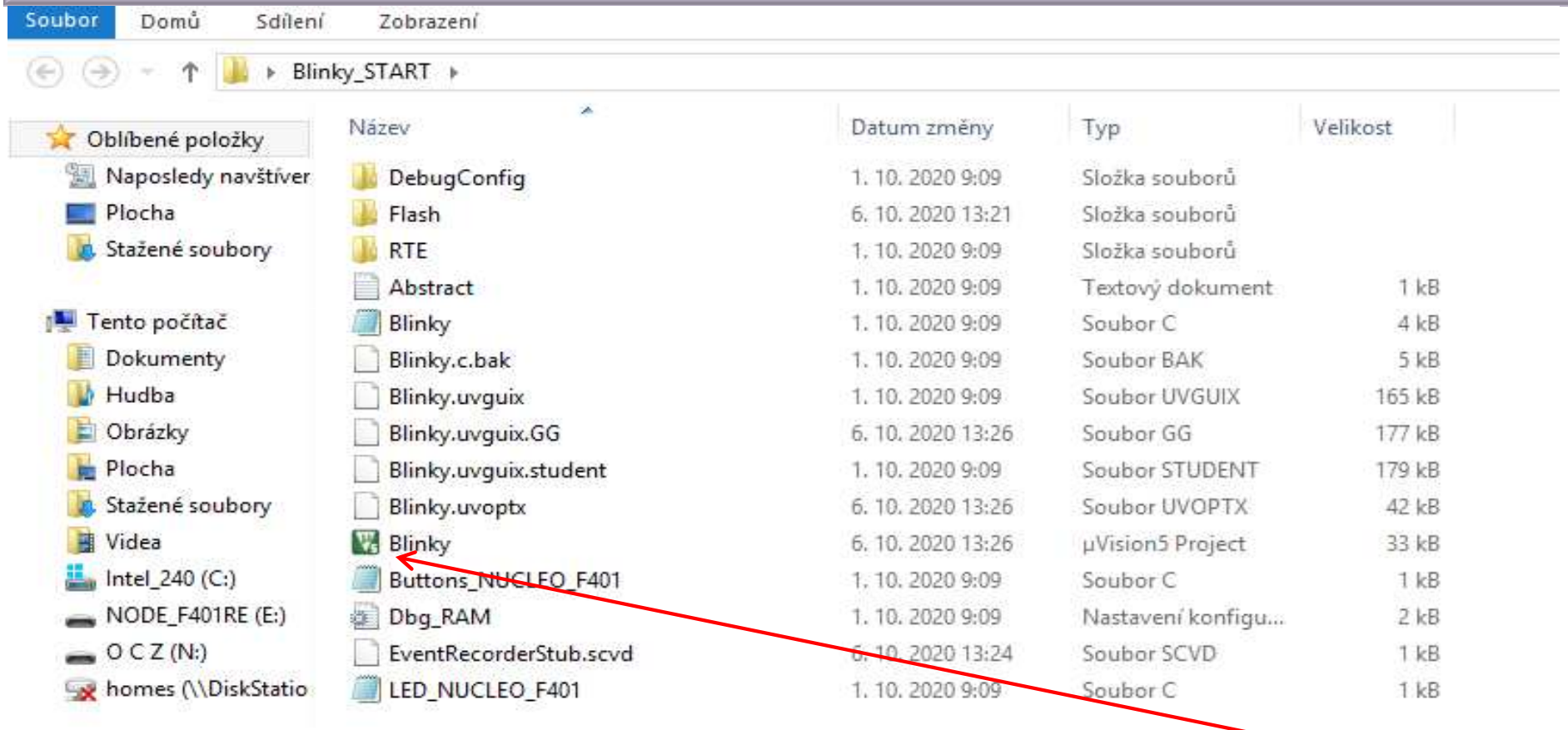
- Propojíme kabelem PC s vývojovým modulem. Na modulu se rozsvítí červená dioda a v části ST-Link svítí dioda též červeně viz. obrázek. Zelená dioda může a nemusí svítit, záleží na posledním uloženém programu.





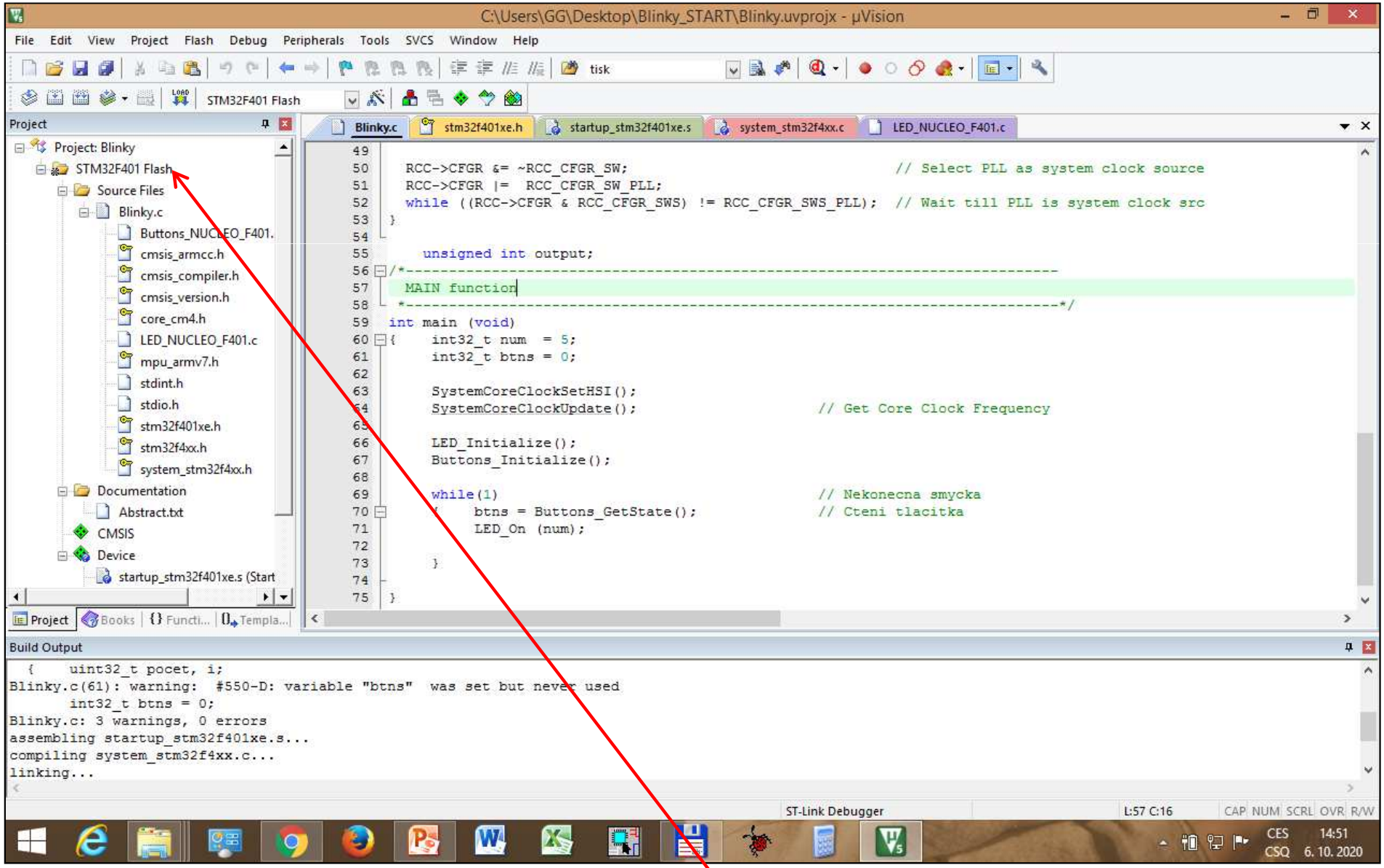
- Z Moodle si nakopírujeme soubor Blinky_START.zip, rozbalíme direktorář Blinky_START umístíme na disk.
- Otevřeme direktorář Blinky_START, kde budou umístěny následující programy.

PRVNÍ KROKY K REALIZACI PROGRAMU



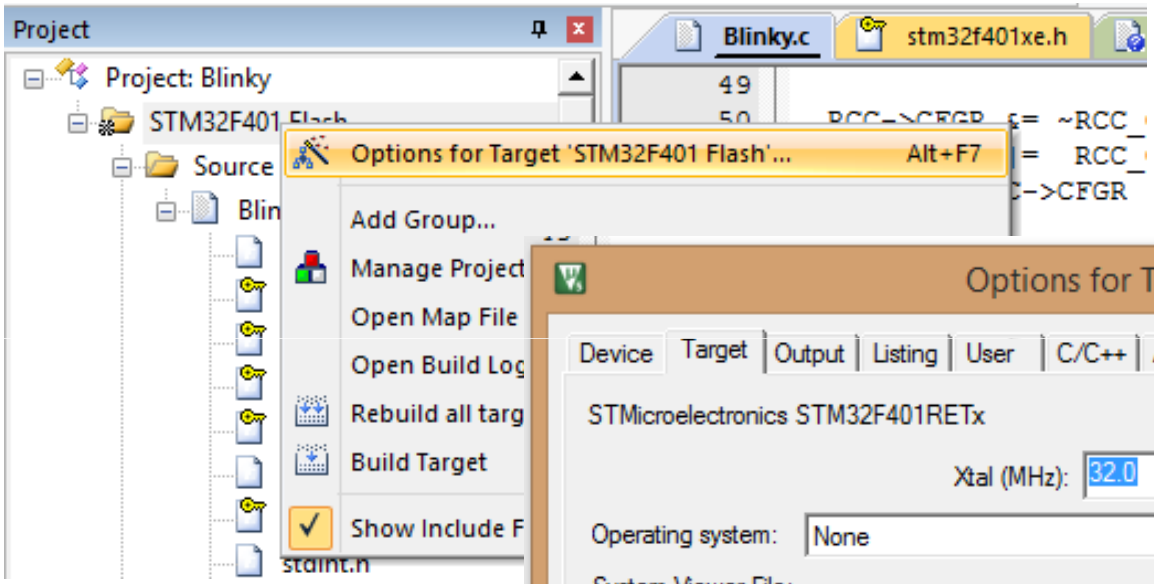
- Vývojové prostředí Keil uVision5 spustíme poklepáním na zelenou ikonku Blinky
- Pro seznámení se ze systémem a ověření funkčnosti bude potřeba doplnit programy Blinky.c, LED_NUCLEO_F401.c a Buttons_NUCLEO_F401.c. Dříve, něž k tomu přistoupíte, zkontrolujeme propojení modulu s vývojovým prostředím.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

PRVNÍ KROKY K REALIZACI PROGRAMU

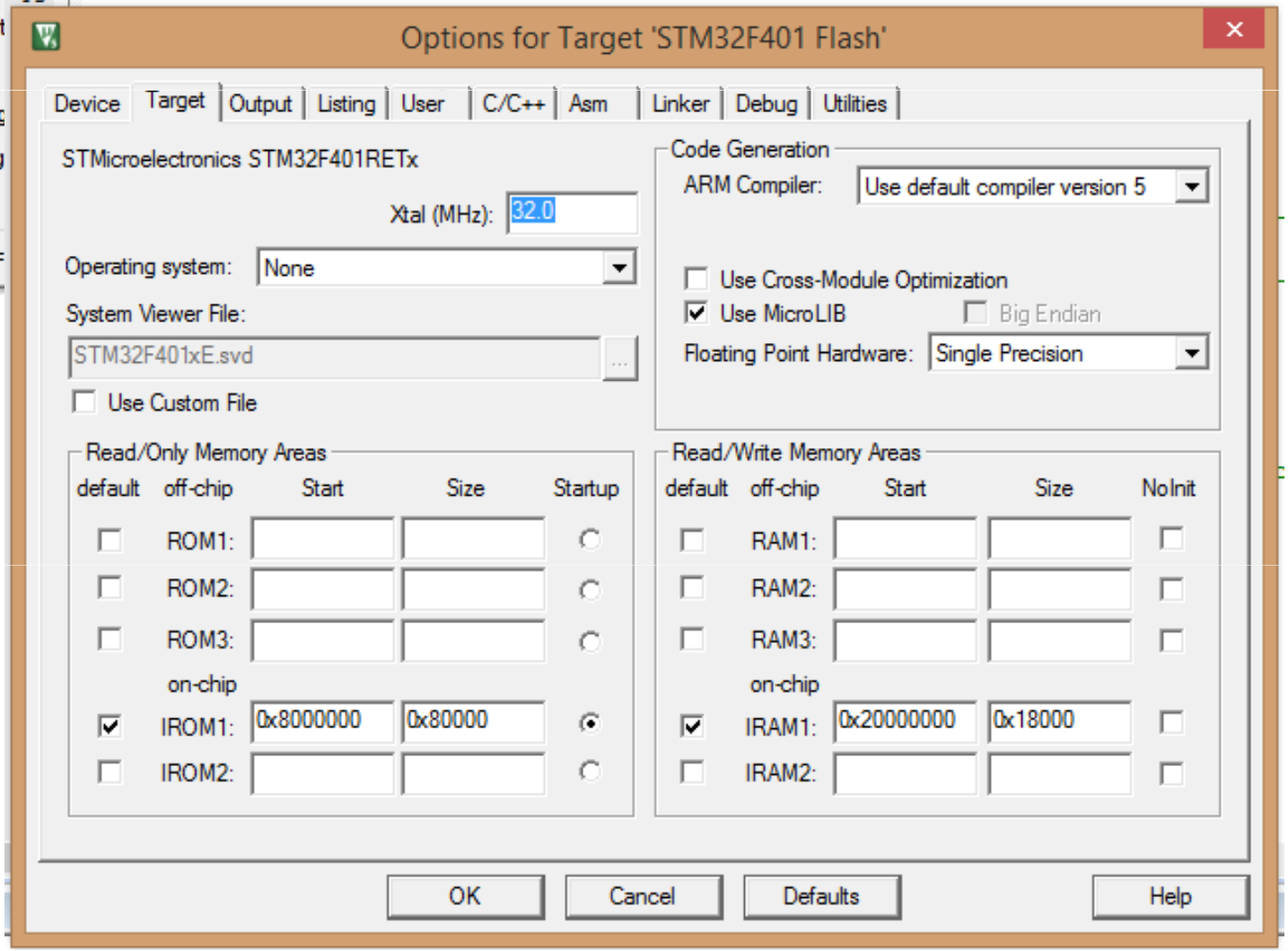


❖ Pravé tlačítko myši na název projektu

NEJDŮLEŽITĚJŠÍ NASTAVENÍ

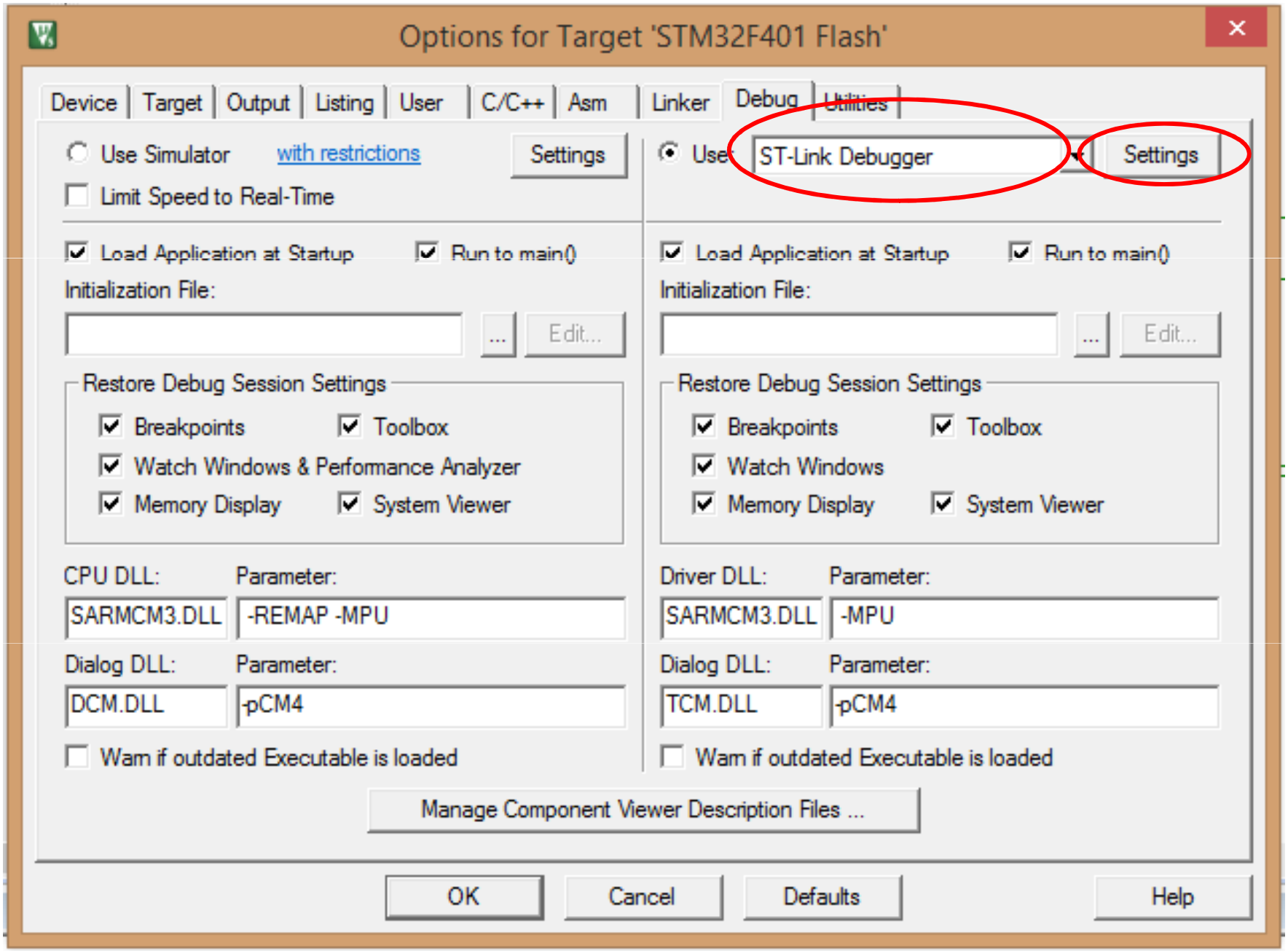


- Po vybrání položky Option for….
- Uvidíme okno

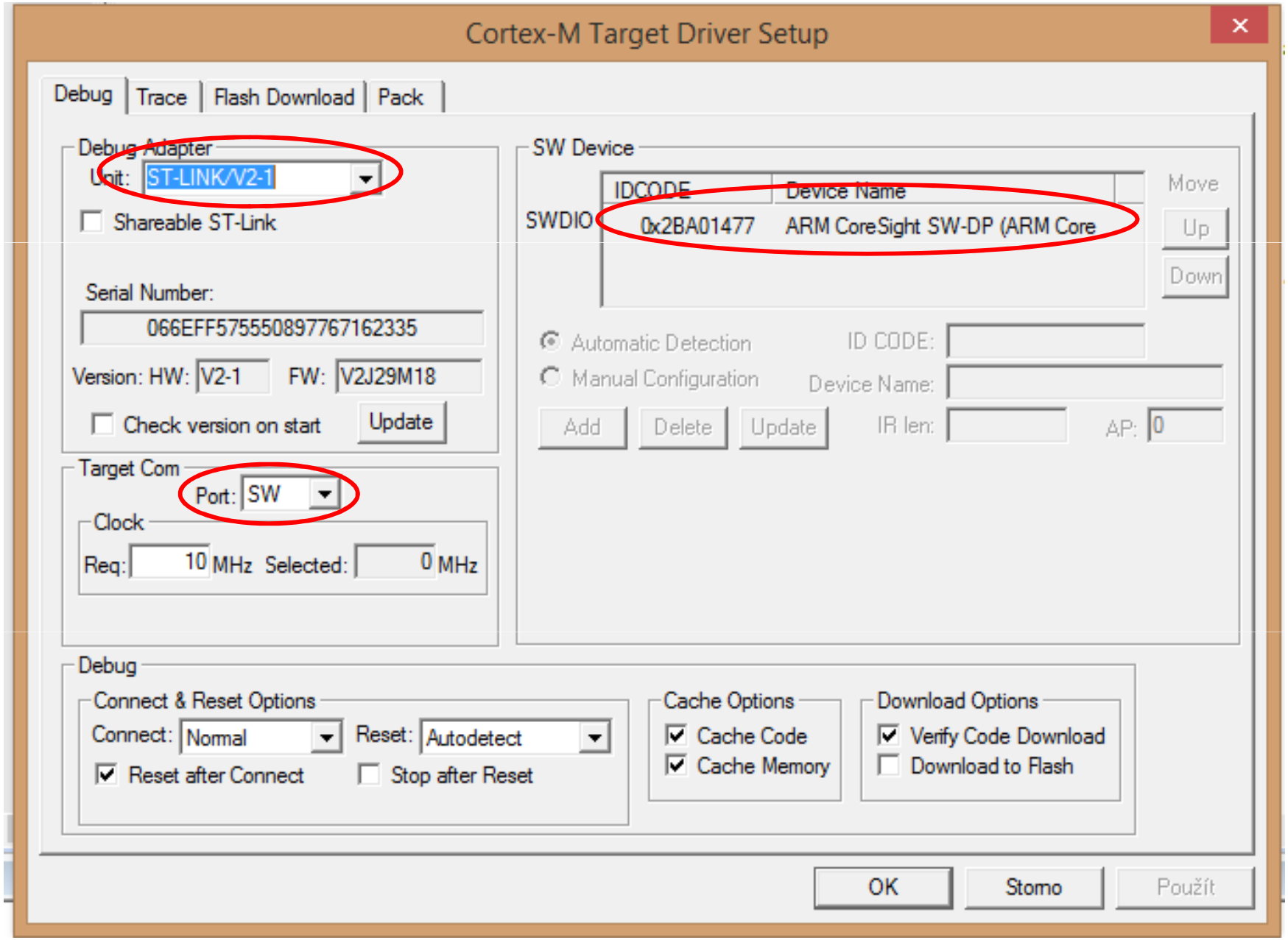


- Device – použitý procesor
- Target – hodinový kmitočet
- Debug – další stránka

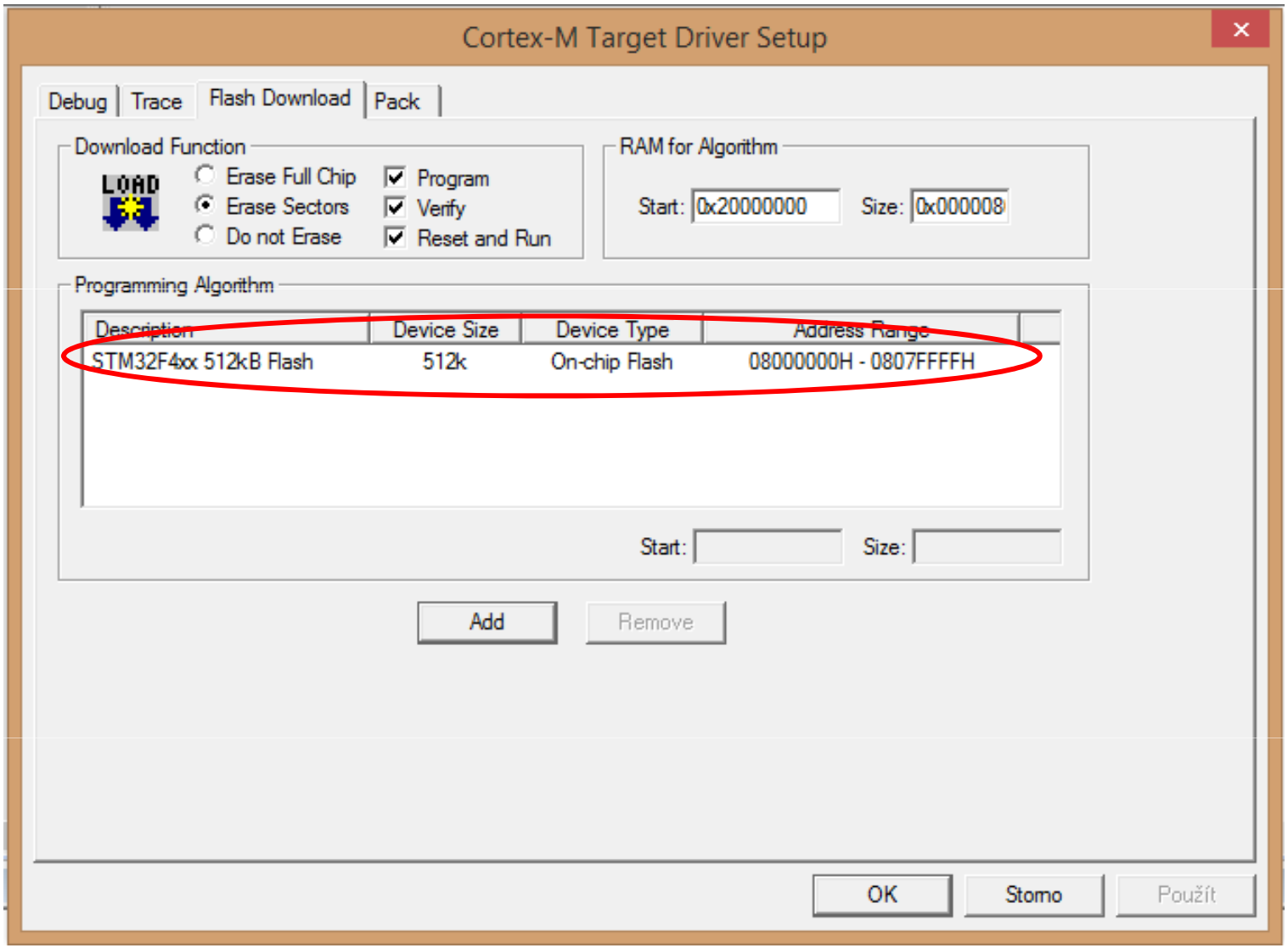
NEJDŮLEŽITĚJŠÍ NASTAVENÍ



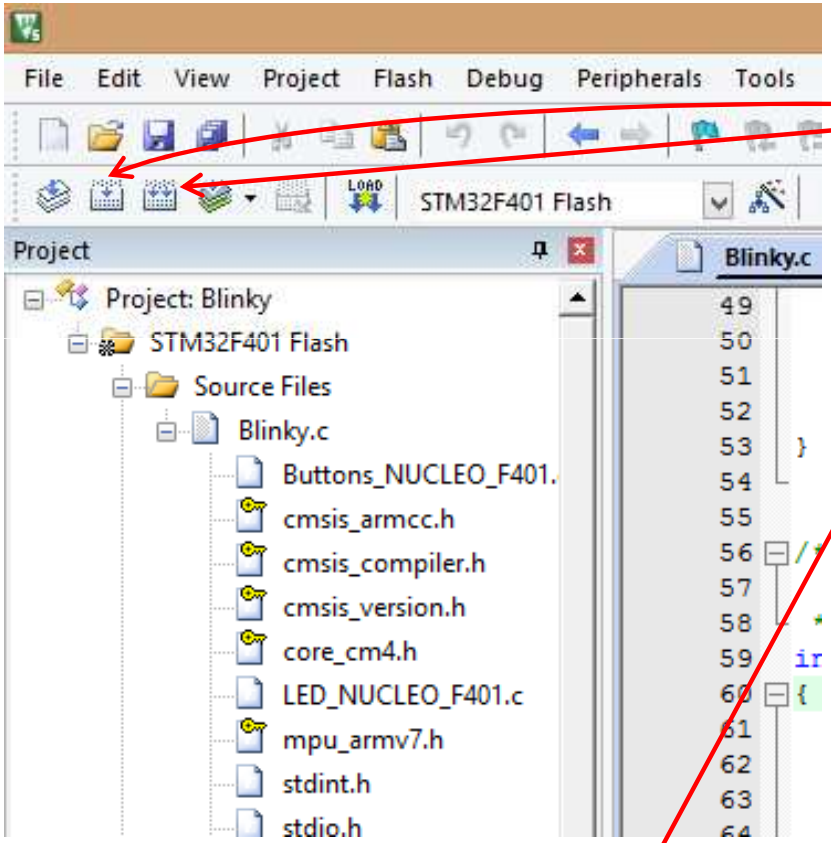
NEJDŮLEŽITĚJŠÍ NASTAVENÍ



NEJDŮLEŽITĚJŠÍ NASTAVENÍ



PŘEKLAD PROGRAMU

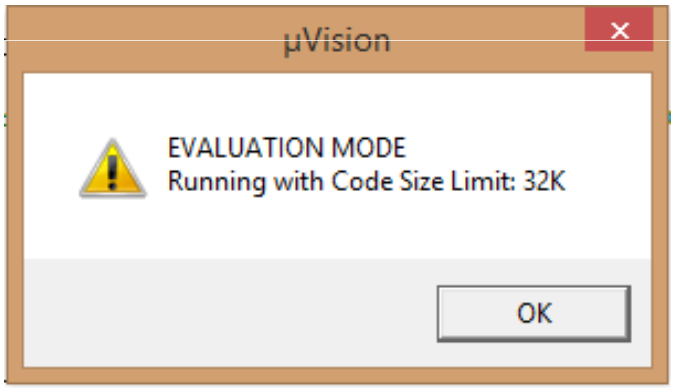


- Překlad programu zajistíme pomocí ikon
- Po překladu zkontrolovat zda je bez chyb, varování jsou většinou nevýznamná
- Code – velikost strojového kódu programu
- RO-data – velikost konstant v programu
- RW-data – velikost paměti pro proměnné

```
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
```

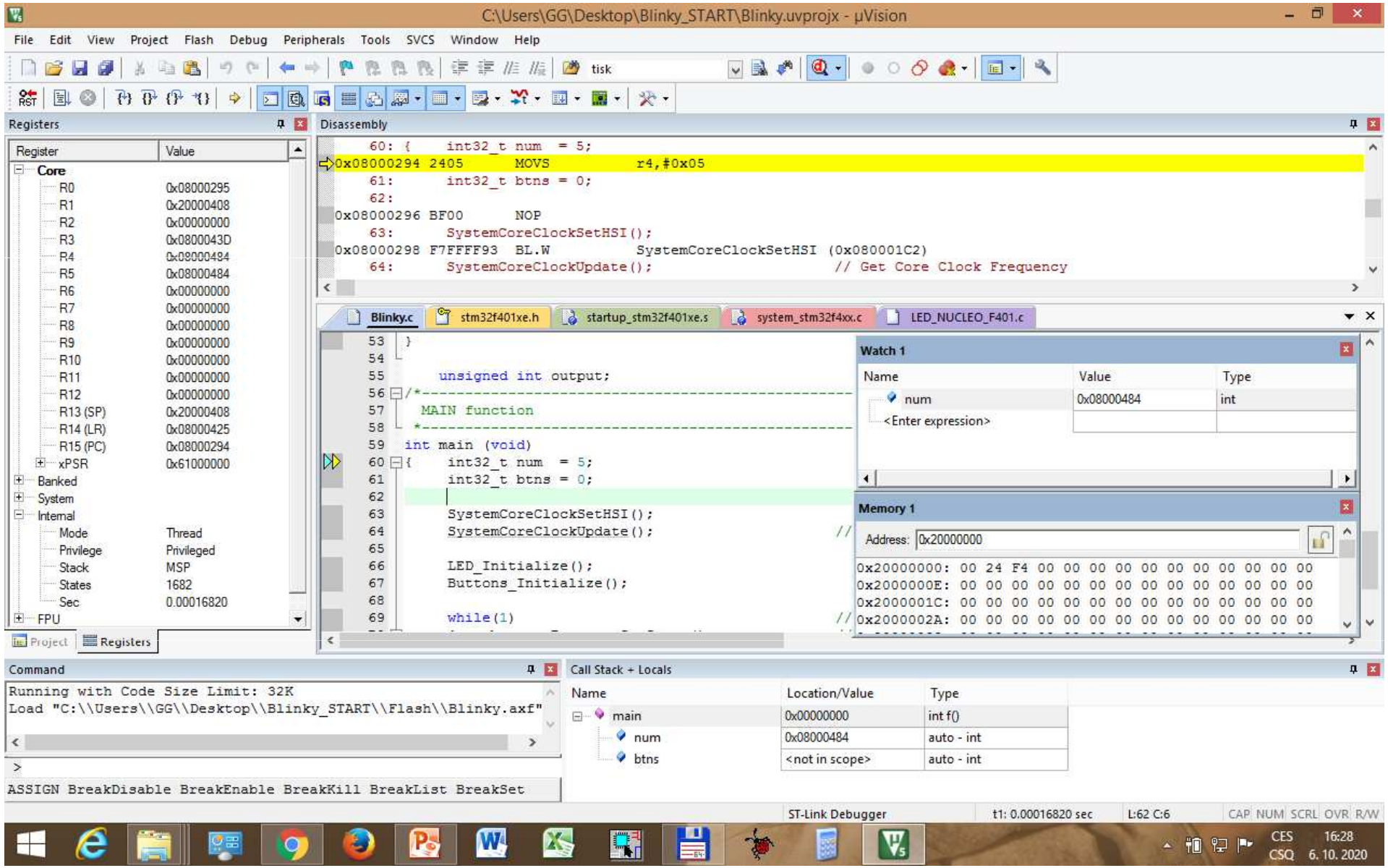
SPUŠTĚNÍ PROGRAMU

- 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



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 rozvícení a zhasnytí 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 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);         // Povoleni hodinového signálu pro GPIOA
       // Nastaveni vývodu PA.5 (Zelena LED)  na výstup push-pull
       // bez upnutí k napájení nebo zemi
       GPIOA->MODER   &= ~((3ul << 2*LED)); // Stav po nulovaní (rušeni předchozího stavu)
       GPIOA->MODER   |=  ((1ul << 2*LED)); // Vystup
       GPIOA->OTYPER  &= ~((1ul <<   LED)); // Push-Pull
       GPIOA->OSPEEDR &= ~((3ul << 2*LED)); // Rušeni 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);           // Povoleni hodinového signálu pro GPIOC
                                            // V registru AHB1ENR nastaven bit 2
                                            // (počítáno od 0)
     // Nastaveni vývodu PC.13 (Modré tlačítko) na vstup, bez upnutí k napájení nebo zemi
     GPIOC->MODER   &= ~(3ul << 2*13);      // Vstup
     GPIOC->OSPEEDR &= ~(3ul << 2*13);      // Rušeni 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);
}
```

VOLNÉ POUŽITELNÉ A REZERVOVANÉ VÝVODY NA NUCLEO F401

VOLNĚ POUŽITELNÉ GPIOx 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

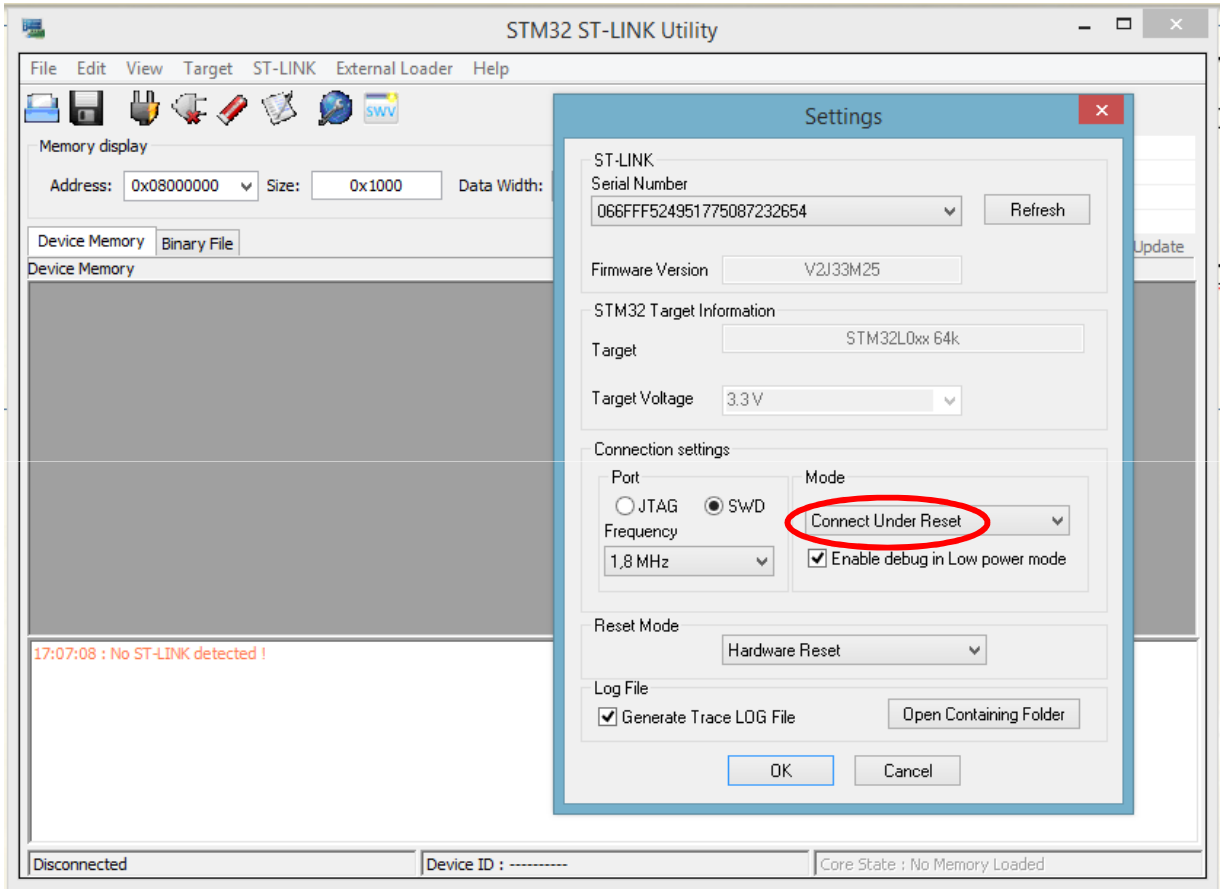
Komparační systém kanál 1 – PA6, PB4, PC6
kanál 2 – PA7, PB5, PC7

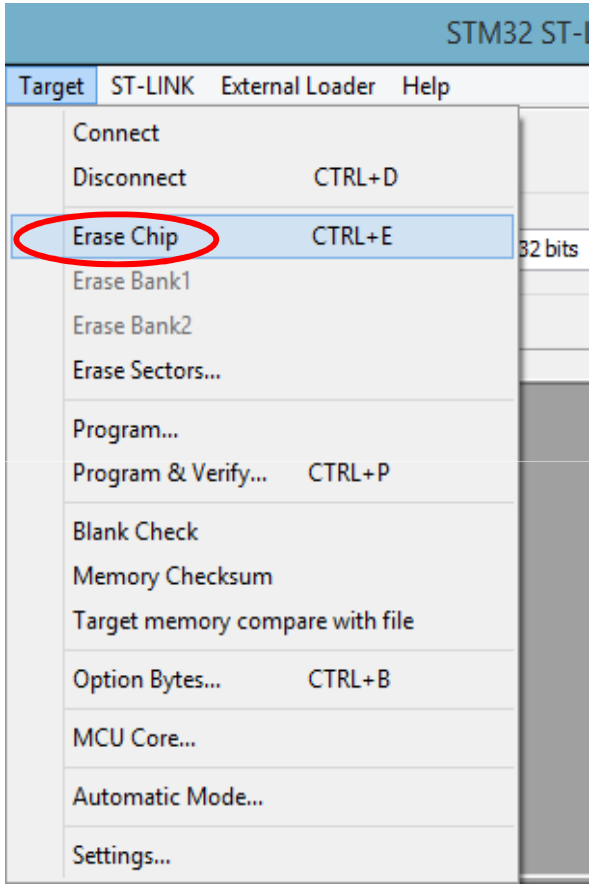
Záchytný systém kanál 1 – PA6, PB4, PC6

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: